International Journal on Software Tools for Technology Transfer manuscript No.
(will be inserted by the editor)

PolyGraph: A Data Flow Model with Frequency Arithmetic

Paul Dubrulle$^{(0000-0002-1158-6348)}$ · **Nikolai Kosmatov**$^{(0000-0003-1557-2813)}$ ·
Christophe Gaston$^{(0000-0001-6865-5108)}$ · **Arnault Lapitre**$^{(0000-0002-2185-4051)}$

Received: date / Accepted: date

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 static data flow paradigms, 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 illustrate how PolyGraph can be used in practice on a realistic Advanced Driver Assistance System (ADAS), and present a framework to check PolyGraph properties in the tool DIVERSITY, along with experiments on realistic and random models.

Keywords Dataflow · Real-time · Performance analysis · Formal semantic · Consistency · Liveness · Cyber-Physical System · Data fusion · Advanced Driver Assistance System

P. Dubrulle, N. Kosmatov, C. Gaston, A. Lapitre
CEA, List, 91191 Gif-sur-Yvette, France
E-mail: firstname.lastname@cea.fr

N. Kosmatov
Thales Research and Technology, 91120 Palaiseau, France
E-mail: nikolaikosmatov@gmail.com

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 designers. 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 of respecting the application's real-time requirements.

Data flow formalisms [6, 24, 28] can be used to carry out this kind of static analysis [7, 9, 17–19]. 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 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. Our goal is to extend an existing data flow formalism for which the consistency and liveness properties of a given model are decidable, in order to cover applicative real-time constraints. In doing so, we want to ensure that the expressiveness extension does not impact the decidability of these properties, and that the verification can be performed in abstraction of a particular implementation's characteristics (like execution times or mapping).

With this extension, the data flow performance analysis will benefit from the additional information on the system behavior to produce tighter bounds. Compared to approaches where all components are modeled as periodic and globally synchronous tasks [11,15], we want to give the possibility to enforce this behavior only for the subset of the modeled components with real-time constraints. By having strict timing constraints only where needed in the model, the behavior of modeled systems will be less constrained and we can expect the performance prediction to be less pessimistic [10].

Approach and main results. This paper introduces PolyGraph[1], a specification language extending some variants of Synchronous Data Flow (SDF) [6,24,28] for specification of frequency constraints on the components. We define an arithmetic based on rational numbers to reason about the synchronization of the frequency constrained components and data exchanges. We show that the theorems providing a theoretical foundation for practical verification of consistency and liveness for a static data flow model can be generalized to this new formalism. Finally, we propose a framework to decide the liveness of polygraphs, in a way similar to [24,18].

The contributions of this work include:

– a data flow formalism, called PolyGraph, extending variants of the well-known SDF [24] formalism, to support the synchronization of data production and consumption on a reference time scale;
– a demonstration that the decidability of two classical data flow properties, namely consistency and liveness, is preserved for this new formalism;
– an overview of a modeling methodology based on PolyGraph applied to a detailed and realistic use case;
– an algorithm to check the liveness of polygraphs and an implementation in the DIVERSITY tool;
– initial experiments with this tool to validate our approach.

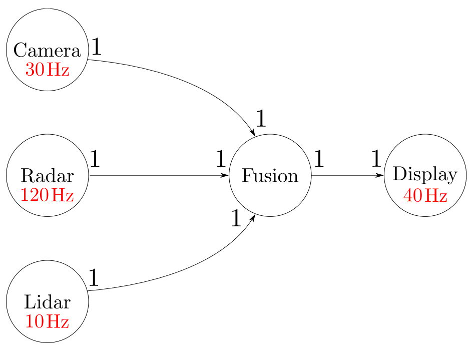
[1] We write "PolyGraph" (capitalized) for the specification formalism, and a "polygraph" for a specific model in it.

This paper is an extended version of a previous paper [14]. Extensions include a carefully revised, more detailed presentation of the motivation and the formalism, with a generalized version of some definitions for the synchronous constraints and many additional examples and useful properties. Detailed proofs (or, for some technical parts, proof sketches) are given for the theorems. The definitions and properties are illustrated with additional running examples and counter-examples. The extension proposes a detailed description of a realistic Advanced Driver Assistance System (ADAS) use case and a step by step explanation of a methodology to model it as a polygraph, with useful algorithms to assist system designers in their task. The tool implementation was optimized, leading to a 30% speedup compared to [14]. Finally, a new campaign of experiments has been performed, based in particular on an automated random polygraph generator, specifically implemented for this work.

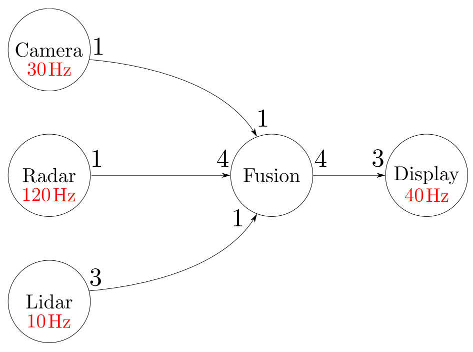
Outline. The remainder of this paper is organized as follows. Section 2 gives an informal introduction to the proposed modeling approach, with a step-by-step explanation relying on an illustrative system. In Sect. 3, we formalize PolyGraph. Section 4 provides the statements and proofs for the consistency and liveness theorems. Section 5 presents the algorithm to check liveness of a given polygraph and proves its properties. Section 6 describes the modeling activities to obtain a polygraph on a realistic use case. Section 7 presents our implementation of the liveness checking algorithm and its evaluation. In Sect. 8 and 9, we discuss related and future work, while Sect. 10 presents conclusion and perspectives.

2 Motivation and Illustrating Example

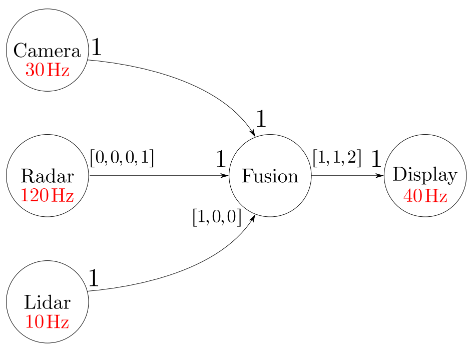
Motivating example. To introduce the modeling approach behind PolyGraph, we use a toy example of a data fusion system that could be integrated into the cockpit display of a car, depicted in Fig. 1. The system is composed of three sensors producing data samples to be used by a data fusion component, and 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



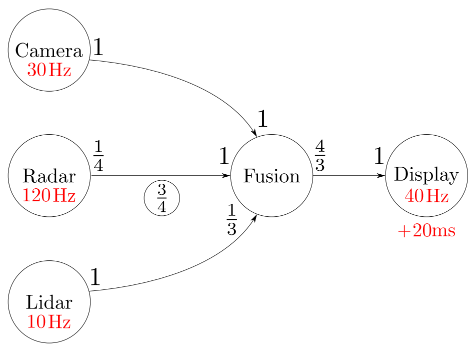
(a) Variant with inconsistent SDF rates



(b) Variant with consistent SDF rates



(c) Variant with CSDF rates



(d) Polygraph variant

Fig. 1: A data fusion system modeled as a data flow graph with frequencies on a subset of actors. The variants a) to d) specify an amount of data exchanged by the components in different variants of the model. In the polygraph variant d), there are two non-trivial initial conditions: 3/4 for the channel connecting the radar and fusion actors (denoted by a circled rational number) and a phase of 20ms for the display actor.

produce a descriptor of the closest detected obstacles. The fusion component uses this information to draw the obstacle descriptors on the corresponding frame.

We use this data fusion example to introduce classical data flow concepts step by step and explain the limitations of existing data flow models, preventing to capture all its requirements. At each step, we explain informally the extensions present in PolyGraph that overcome these limitations, and their impact on the decidability of data flow properties. The modeling of the toy example actually provides counter examples proving that existing statements for SDF and derived models are no longer sufficient to verify these properties.

2.1 Frequency Constrained Communication

Communication dependencies. The first step to model this system is to build a graph capturing data dependencies between the components. Each vertex models an *actor*, an abstract entity representing the function of a component. Each directed edge models a communication *channel*, the source actor being the producer of data consumed by the destination actor. The structure of the graphs in the variants of Fig. 1 illustrate 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.

In our example, we consider that a camera frame is a token for the two channels connecting the camera, fusion, and display actors. For the channels connecting the radar and lidar actors to the fusion actor, we consider a token suitable to store the number of obstacle descriptors drawn on each camera frame.

The actors communicate by *firing*, an atomic process during which they consume and produce a certain number of data *tokens* on the connected channels. With this policy, the graph can be assimilated to a Kahn Process Network (KPN) [21]. In a KPN, the communications are determinate, but in general it is not possible to statically analyze its performance.

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. For our example, we consider the frequency labeling illustrated by Fig. 1.

A *global clock* is used to synchronize the firing of frequency labeled actors. Since the synchronous constraints are expressed as frequencies, we have a multi-periodic system, and the period of the global clock is defined in the next section to observe the hyperperiod of that system (100ms in our example). In addition, to specify the acceptable latency between the firings of actors, the first firing of an actor can be delayed by a certain amount of time in the hyperperiod, called a *phase*.

Hence, in the hyperperiod of the system, there are *significant dates* at which some actors must fire. As formalized in the next section, the period of the global clock is divided by a sufficient number of ticks to identify all the significant dates in the hyperperiod. Due to the multi-periodic nature of the system and the phases, it is possible that only some ticks identify significant dates (we call them *significant ticks*).

In PolyGraph, the time it takes for an actor to fire is abstracted away, which is comparable in a sense to the synchronous hypothesis of synchronous languages [4]. We consider that a firing occurs at a tick if all the expected communications are completed as an atomic transaction on that tick.

In our example, we want to specify that the acceptable latency to draw the result of the fusion on the screen after capture by the sensors is 20ms. By adding that phase to the display actor, we specify that its first firing occurs 20ms after the beginning of the hyperperiod. The sensor actors do not have a phase, so they all fire synchronously at the start of the hyperperiod. Hence, the time elapsed between the firings of the sensing actors and a firing of the display actor that depends on the data they produce should be at most 20ms. Of course, in the mathematical formalism, the real-time units are abstracted.

Mixed-firing policy. Generally, in real-life systems, computation kernels compute asynchronously, as soon as input data is available, and they do not have frequency constraints. In our frequency labeling, the actors modeling such components can be left without a frequency label. Their firing is also an atomic transaction completing all the expected communications, but it can occur at any tick, significant or not. In our example, this is the case for the fusion actor.

The possibility to have unlabeled actors is an important and differentiating part of our approach, as further discussed in Sect. 8. 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 synchronous constraints that apply only where needed by the real-life system. In general, periodic synchronous constraints inflict a penalty on the performance of the system, delaying the execution of the constrained component. Relaxing these constraints where they are not necessary expands the search space of static analysis algorithms, and may allow them to produce tighter bounds in some cases (for example allowing bursts of executions for components with short execution times, reordering of unconstrained executions, *etc.*).

2.2 Memory-Bounded Periodic Schedule

Another characteristic of real-life software components in our context is that they require a fixed number of input samples from each source. For example, the fusion component requires one frame from the camera and obstacle detection descriptors from the radar and lidar sensors.

The constraint on the number of tokens to produce and consume can be captured by KPN restrictions, such as Synchronous Data Flow (SDF) [24]. Thanks to the linear behavior of a modeled system in these restrictions, they allow us to decide for any given system whether the communications can occur in bounded memory for an unbounded repetition of a periodic schedule of the modeled system. Systems respecting this property are said to be *consistent*.

Static rates. In SDF, both ends of each channel are assigned a *rate*, denoting a number of tokens. The rate at the origin of a channel is called an *output* (or *production*) *rate*, and that at the destination is an *input* (or *consumption*) *rate*. An actor thus has a rate for each channel it is connected to, and each of its firings consumes or produces the corresponding number of tokens on these channels.

Without taking frequencies into account, if we assign a rate of 1 to all actors on all channels as in the variant of Fig. 1a), the resulting SDF matches 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, given any marking of the graph with any numbers of tokens stored in the channels, if all actors fire 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 to respect the read-blocking policy of the channels. It will consume only 10 tokens from the camera and radar actors, leaving respectively 20 and 110 unconsumed tokens on their 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 adjusted in a periodic schedule.

A possible adaptation of those naive unitary rates, given in the variant of Fig. 1b), 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.

It is then important to state the difference between an actor constrained by a frequency label, and an actor inheriting a frequency by transitivity. An actor with a frequency label must fire synchronously with the ticks of the global clock. For an actor that inherits the frequency, the number of firings in a periodic schedule are constrained, but they can occur at any tick.

The rates of the SDF variant of Fig. 1b) are satisfactory to achieve consistency, but do not accurately reflect the expected behavior. The fusion actor would consume 4 tokens per firing from the radar actor, while in reality the component only requires 1. This would result for example in an inaccurate static analysis of the memory footprint required to store the samples.

Cyclo-Static Rates. It is possible to use Cyclo-Static Data Flow (CSDF) [6] to get closer to the real communication requirements. In CSDF, the successive firings of an actor cyclically consume or 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, 2 tokens for the fourth, 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.

In our context, 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 rate to a cyclo-static sequence accordingly. In other words, the second actor is in charge of resampling the token stream to cope with the difference in frequencies.

To illustrate this approach, the variant of Fig. 1c) uses cyclo-static sequences. The fusion actor requires one token from each sensor every firing. Since the component is synchronized on camera frames, we decide that the fusion actor's reference frequency should be the same as the camera actor (30Hz). For the channel connecting these two actors, the frequency constraints do not conflict with the expected communication behavior, and we assign a static rate of 1 to both ends.

Now, considering the radar actor, the fusion actor is taken as reference and 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. The radar actor is thus in charge of downsampling its output stream towards the fusion actor. For the lidar actor, the fusion actor 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. This time an upsampling is required, and the fusion actor is in charge of its implementation. A similar logic is applied for the connection to the display actor. The output rate of the fusion actor was chosen to illustrate an overproduction to balance the communication. Indeed, the fusion actor will produce one additional token every three firings.

With this variant, only the required tokens are exchanged on the channels, and the consistency property is preserved. 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 one out of four computed results and drop the others.

The solution with CSDF rates is satisfactory regarding expressiveness and property decidability. But in all generality, we believe that choosing appropriate cyclic rate sequences for all channels manually is not convenient for the system designer. Indeed, for large systems with a great number of different frequencies and connections, the repetitive task of manually specifying sequences satisfying the resampling requirements, in particular for non-trivial frequency ratios, may become tedious. In addition, we believe that human beings tend to make mistakes in such repetitive tasks.

Rational Rates. We propose instead to extend the SDF model with rational communication rates, as attempted in Fractional Rate Data Flow (FRDF) [28], where a rate $r = p/q$ specifies that the actor produces or consumes either a fraction p/q of token every firing or p tokens every q firings. In FRDF, the detailed semantic of the communication with a rational rate is not formalized. We formalize in the next section a semantic where p tokens are produced or consumed 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. As further detailed in Sect. 6, there is a unique default cyclo-static sequence that corresponds to a given rational rate, and Algorithm 2 provided in that section describes how to compute it. In some sense, the proposed semantic with a linear behavior over rational numbers will be used to approximate the desired non-linear cyclo-static behavior with a (possibly different) integer number of produced or consumed tokens for each firing, and will bring the benefit to ensure good properties of the system thanks to this linear approximating behavior. The proposed variant of the system for our example is shown in Fig. 1d). For now, we ignore the rational marking on the channel connecting the radar to the fusion actor. In this case, the default sequences for the rates given in this variant are those of the CSDF variant Fig. 1c).

We propose the same methodology as mentioned above for CSDF rates: for a given channel, the frequency and the rate of one actor are considered as a reference, while the other one adapts its rates according to that reference. The rate of the reference actor corresponds to the exact communication requirement of the corresponding software component, and is thus a static integer rate (the rates of 1 in our example, cf. Fig. 1a)). The rational rate for the other actor is computed using the ratio between the frequencies of the two actors. In our example, the adjusted output rate of the radar is thus $1 \times 30/120 = 1/4$, and the adjusted input rate of the fusion on the channel coming from the lidar is $1 \times 10/30 = 1/3$.

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, and the required resampling sequences can be be automatically computed from the rates and frequencies. We have seen a counter-example showing that the existing conditions for the consistency property are no longer sufficient when adding the frequency constraints. We provide (in Theorem 1 below) an extended statement to the existing consistency theorem, along with a proof that the required conditions are necessary and sufficient.

2.3 Causally Correct Schedule

Causality issues can appear in static data flow models without frequencies: in the case of cyclic graphs, the firings of the actors in a cycle all depend on each other. This means that without a sufficient number of tokens initially occupying the channels in a cycle, there is a deadlock in the scheduling of the modeled system. To prevent this, it is possible to *mark* the channels with a sufficient initial number of tokens, allowing the firings of all actors in the cycle. The *liveness* property of a static data flow graph is verified when all cycles in the graph are marked with enough tokens to prevent a deadlock [24,6].

Even though the graph of Fig. 1d) is acyclic, we can show that without introducing initial conditions it has causality issues, as illustrated by the timing diagram of Fig. 2. This diagram illustrates the expected timing of actor firings in our example, and the data dependencies between them, according to the semantic formalized in the next section. Note that the numbers of tokens produced or consumed by the firings for the graph of Fig. 1d) without initial conditions are exactly given by the cyclo-static rates of the variant of Fig. 1c).

The diagram represents the scheduling constraints for our example: a firing that must occur at a tick cannot happen at any other time, and for any data dependency, the source of the arc should occur before its destination, or at the same time. It is then obvious that the data dependencies marked by a cross in Fig. 2 are not satisfied in time since they go backwards on the time scale.

For example, the first firings of the display and camera actors (that is, D1 and C1) must occur at the first tick. The first firing of the fusion actor must thus occur at the first tick because of dependencies C1→F1→D1. This constraint for F1 is not compatible with its data dependency on the fourth firing of the radar actor (R4). Indeed, R4 must occur synchronously at a later tick.

We thus extend the notion of deadlock in data flow literature to this particular situation. It is arguable to call a situation where inputs are missing at some real-time date a deadlock. We make that choice because, in the formalization, in that particular case the clock

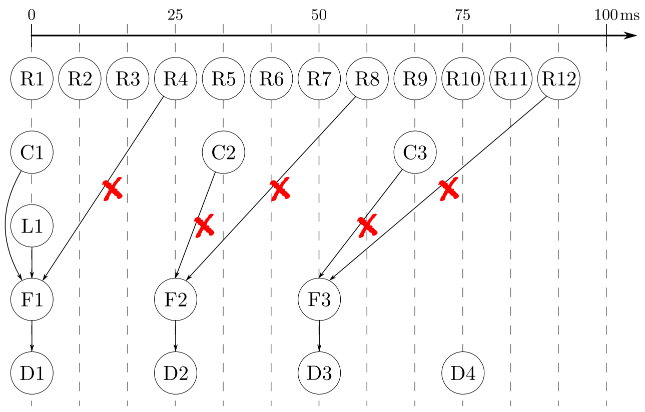


Fig. 2: Scheduling constraints of the example from Fig. 1d) considered without initial conditions. Firings are identified by the initial letter of the corresponding actor and the rank of the firing. Arrows show data dependencies between firings. The vertical dashed lines represent the significant ticks in the hyperperiod. The firings that must occur synchronously at one of them are centered on that tick. The data dependencies marked by a cross introduce a causality issue.

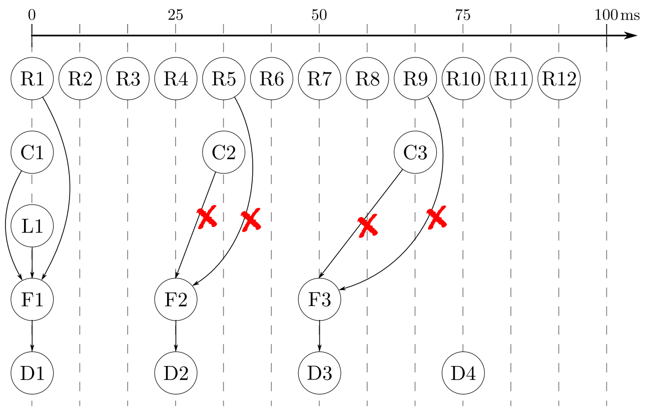


Fig. 3: Adjusted scheduling constraints from Fig. 2 taking into account the initial marking of 3/4 on the channel connecting the radar and fusion actors. The production of the first token by the radar occurs 3 firings earlier. Some data dependencies are still invalid.

cannot tick without breaking the validity of the execution, in the same way as the actor with missing inputs cannot fire, and the producing actor of the missing inputs cannot fire either because the clock cannot tick. The execution cannot progress one step further and remain valid due to starvation. We interpret this loop in causality between these events as a deadlock.

Hence, for the frequency extension we propose, while a sufficient marking for cycles is necessary and sufficient for SDF and CSDF models, it is not a sufficient condition to ensure a schedule without deadlocks for a polygraph. The additional condition we need is that the input tokens required by a firing can be produced at an earlier tick of the global clock, or at the same tick. We thus propose an extension to the initial marking of data flow graphs in order to shift production and consumption times.

Rational marking. One way to adjust production or consumption times is to change the default sequences defined by the rational rates, as explained in the following.

For this, we propose a rational initial marking of the graph. Each channel with natural rates at both ends can be marked with a positive integer giving the initial number of tokens, as in (C)SDF. Each channel with a rational rate $r = p/q$ on one of the ends can be initially marked with a rational number $n + k/q$ with $0 \leqslant k < q$, which indicates that the channel initially holds n tokens (as in (C)SDF), and, if $0 < k$, the default sequence is adjusted according to k. As detailed after the formal definitions in Sect. 6, if the rational rate is on the producer, the default sequence is rotated left by k positions; otherwise, it is rotated right by k positions.

To illustrate how this helps decreasing the risk of a deadlock, consider the model of Fig. 1 with the default sequences of variant c) and the corresponding rational rates of variant d), and let us illustrate the effect of the initial marking of 3/4 on the channel connecting the radar to the fusion actor in that variant. (The phases are considered to be 0 for the moment). This initial marking does not add a token, but rotates the default sequence $[0, 0, 0, 1]$ by 3 elements to the left, yielding the sequence $[1, 0, 0, 0]$. When comparing the schedules of Fig. 2 and Fig. 3, we can see that this marking shifts the dependency R4→F1 to R1→F1, R8→F1 to R5→F1, and so on. With this adjustment, the tokens are produced earlier by the radar actor, on time for D1 to fire synchronously with the first tick. However, some dependencies remain unsatisfied, and we explain below a complementary approach to the initial marking to correct this.

Note that there is a functional reason to add an initial marking, it is not just some mechanism to workaround causality issues. Indeed, in Fig. 3, all the firings of the sensing actors producing tokens used by the fusion actor occur at the same tick. This means that the data samples used by the actual component in the system have equivalent production dates, which is preferable for fusion algorithms. More details on this aspect are given in Sect. 6 after the formalization.

Relaxed phases. We proposed earlier to use a phase to express the maximal acceptable latency between firings of actors. Increasing this phase is a complementary approach to the initial marking of the channels in order to

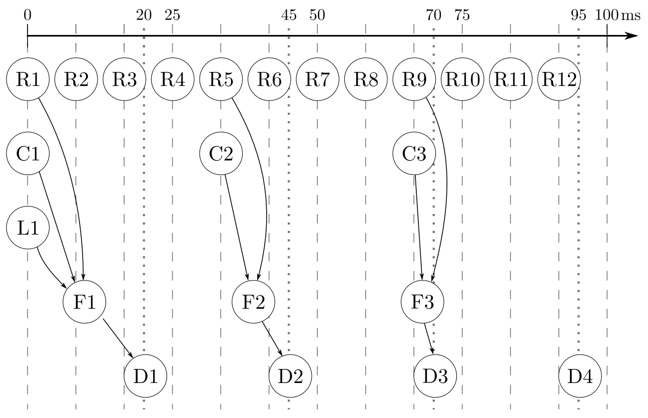


Fig. 4: Adjusted scheduling constraints from Fig. 3 taking into account the phase of 20ms for the display actor. All the firings of the display actor are delayed in the hyperperiod, which introduces new significant ticks (dotted lines). All the data dependencies are satisfied on time.

prevent causality issues. In other words, for some fixed rates and initial markings, there might be a minimal latency required to achieve a causally correct periodic schedule.

This is the case in our illustrative example of Fig. 1d) with the initial marking (and without the phase for the display actor), illustrated by the constraints of Fig. 3. The dependencies C2→F2→D2 and C3→F3→D3 are invalid. Let us show how this issue can be addressed by choosing a phase for the display actor. The longest delay between an activation of the display and the production of a token it requires by the camera is the one between D3 and C3. Relying on the period of the camera and display firings, the duration between these firings is easily inferred as 50/3ms, i.e. ≈ 17ms. The dependencies C3→F3→D3 cannot be satisfied unless the display actor has a phase greater or equal to that duration. Since with the initial marking, the firings R5 and R9 occur synchronously with C2 and C3, the minimal latency to satisfy the dependencies R5→F2→D2 and R9→F3→D3 is the same.

The chosen latency of 20ms in the initial conditions of Fig. 1d), chosen for functional reasons in the design process described earlier, is greater than this lower bound. Combined with the rational marking on the radar-fusion connection, chosen to have temporally correlated samples for the fusion algorithm, it is suitable to have a live schedule, as shown in Fig. 4.

The rational marking and phases provide two degrees of adjustment to synchronize the data exchanges. As for the consistency property, we have shown that with the synchronous firing constraints, the existing conditions for the liveness property are not sufficient. We provide (in Theorem 2 in Sect. 4) an extended statement to the existing liveness theorem, along with a proof that the required conditions are necessary and sufficient. A model verifying this property guarantees that the requirements captured by the rational markings and the phases are feasible in practice in a causally correct periodic schedule.

3 The PolyGraph Language

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. A number $r \in \mathbb{Q}$ rounded down (resp., up) to a closest integer is denoted by $\lfloor r \rfloor$ (resp., $\lceil r \rceil$).

For a set A, we denote A^* the set of all finite sequences of elements of A and A^+ the set of all nonempty sequences; in other words, A^* is the *free monoid* on A and A^+ is the *free semigroup* on A. For any sequence $w = a^1 \cdots a^n \in A^*$, the i^{th} element of w is denoted $w[i] = a^i$ and the length of w is denoted $|w| = n$. The concatenation of $w, w' \in A^+$ is denoted $w \cdot w'$.

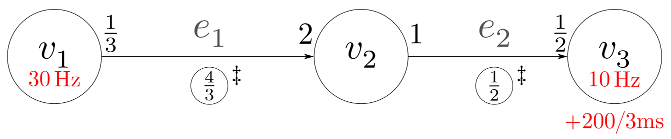
For any $1 \leqslant j \leqslant n$, we denote by $\mathbf{u}_j$ the vector $(u_i) \in \mathbb{B}^n$ such that $u_i = 1 \Leftrightarrow i = j$. If there is no risk of ambiguity on the number n of rows of a column vector, we denote by $\mathbf{0}$ the vector whose components are all 0. For a matrix $\mathbf{A} \in \mathbb{Q}^{n \times m}$, we denote the transpose of this matrix by $\mathbf{A}^{\mathrm{T}} \in \mathbb{Q}^{m \times n}$, which is mainly used to write vectors in a more compact form, e.g. for a three-element zero vector $\mathbf{0} = \begin{bmatrix}0 \ 0 \ 0\end{bmatrix}^{\mathrm{T}}$. Finally, for two vectors $\mathbf{a} = (a_j), \mathbf{b} = (b_j) \in \mathbb{Q}^n$ we write $\mathbf{a} \leqq \mathbf{b}$ (resp., $\mathbf{a} \geqq \mathbf{b}$) if for any j we have $a_j \leqslant b_j$ (resp., $a_j \geqslant b_j$).

3.1 System Components

Actors, channels, tokens, and firings. The function implemented by a component of the modeled system is represented by an abstract entity that we call an *actor*. Each function has expected inputs and outputs. Read-blocking FIFO *channels* represent the connections between outputs and inputs of different modeled functions.

There is an implicit data type for the communication over a given channel, and we call one instance of that type a *token*. A *firing* of an actor corresponds to one execution of the modeled function, which consumes the required input tokens and produces the resulting output tokens. These connections are modeled as a *(system) graph*.

Definition 1 (System graph) A *(system) graph* is a connected finite directed graph $G = (V, E)$ with set of nodes (or *actors*) V and set of edges (or *channels*) $E \subseteq V \times V$.



$$G = (V, E) \quad V = \{v_1, v_2, v_3\} \quad E = \{e_1, e_2\} \quad V_F = \{v_1, v_3\}$$

$$\mathbf{\Gamma} = \begin{bmatrix} \frac{1}{3} & -2 & 0 \\ 0 & 1 & -\frac{1}{2} \end{bmatrix} \quad \begin{aligned} \omega : V_F &\longrightarrow \mathbb{N} \\ v_1 &\longmapsto \omega_1 = 3 \\ v_3 &\longmapsto \omega_3 = 1 \end{aligned} \quad \begin{aligned} \pi &= 3 \\ \mathrm{T} &= \{0, 1, 2\} \end{aligned} \quad \begin{aligned} \varphi : V_F &\longrightarrow \mathrm{T} \\ v_1 &\longmapsto \varphi_1 = 0 \\ v_3 &\longmapsto \varphi_3 = 2 \end{aligned}$$

$$\Theta = \langle \omega, \pi, \varphi \rangle \quad \mathcal{P}^\dagger = \langle G, \mathbf{\Gamma}, \Theta \rangle \quad \mathbf{c}^\ddagger = \begin{bmatrix} \frac{4}{3} & \frac{1}{2} \end{bmatrix}^\mathrm{T}$$

Fig. 5: Graphical representation of a toy polygraph $\mathcal{P}^\dagger$ used as a running example, along with the corresponding formal definitions, explained throughout Sect. 3. The initial frequencies inside the actors are given in Hz (i.e. times per second), while the ω mapping is equivalently defined in decahertz (i.e. times per 100ms) with a time unit of 100ms adapted to have $\gcd(\omega_1, \omega_3) = 1$.

We consider that V and E are indexed respectively by $\{1, \cdots, |V|\}$ and $\{1, \cdots, |E|\}$, and denote by v_j the actor of index j and by e_i the channel of index i. For an actor v_j, let $\mathrm{in}(v_j) = \{\langle v_k, v_l \rangle \in E \mid l = j\}$ denote the set of *input channels* of v_j, and $\mathrm{out}(v_j) = \{\langle v_k, v_l \rangle \in E \mid k = j\}$ the set of *output channels* of v_j.

Throughout this section, we illustrate the definitions using the toy polygraph $\mathcal{P}^\dagger$, illustrated in Fig. 5. We will detail the definition of $\mathcal{P}^\dagger$ step by step when stating the corresponding formal definition.

3.2 Communication Constraints

Rates and topology matrix. For any pair of a channel e_i and an actor v_j, we associate a rate γ_{ij} which is a rational number whose absolute value defines the partial production or consumption effect on e_i of each firing of v_j, and whose sign indicates if the effect is a partial production ($\gamma_{ij} > 0$) or consumption ($\gamma_{ij} < 0$). Note that by partial, we do not mean that a portion of token is produced or consumed. As explained in Sect. 2, a non-zero fractional part for a rate γ_{ij} rather represents a resampling of the token stream on the channel, necessary when two connected actors do not operate at compatible frequencies. In practice, a rational rate should be assigned to the actor modeling the component in charge of the actual resampling (if it is necessary). Thus at most one of the rates on the two ends of a channel can be non-integer. The Condition (ii) of the following Def. 2 reflects this intended use. The two other Conditions (i) and (iii) state that, by convention, the rate γ_{ij} must be 0 when v_j is not connected to e_i and when v_j is connected to both ends of e_i.

Indeed, for a *self-loop* $e_i = \langle v_j, v_j \rangle$ connecting v_j to itself, the global production/consumption effect of v_j on the channel must be 0 for the model to be consistent. Therefore the associated production and consumption rates must be equal. Their exact value does not matter and can be any integer.

The rates are given by a topology matrix with one row per channel and one column per actor. For example, the matrix $\mathbf{\Gamma}$ of Fig. 5 gives the rates for our example $\mathcal{P}^\dagger$. In the graphical representation, only the non-zero rates γ_{ij} are shown for a channel e_i, near the end connected to the corresponding actor v_j.

Definition 2 (Topology matrix) A matrix $\mathbf{\Gamma} = (\gamma_{ij}) \in \mathbb{Q}^{|E| \times |V|}$ is a *topology matrix for a graph G* if for every channel $e_i = \langle v_k, v_l \rangle \in E$, we have:

(i) the rate $\gamma_{ij} = 0$ for all $j \neq k, l$;
(ii) if $k \neq l$, then the rates $\gamma_{ik} > 0$ and $\gamma_{il} < 0$ are irreducible fractions, and at least one of them has a denominator equal to 1 (i.e. is an integer); let $q_i \geqslant 1$ be the greatest of their denominators, we define $r_i = 1/q_i$ the *smallest fraction portable by* e_i;
(iii) if $k = l$, then $\gamma_{ik} = 0/1 = 0$, and we define $q_i = r_i = 1$.

Channel state. A channel state is a vector of rational numbers with one row per channel, where the component for channel $e_i = \langle v_j, v_k \rangle$ tracks the partial production or consumption effect of successive firings of v_j and v_k by addition of rates γ_{ij} and γ_{ik}. The component for e_i must thus be a multiple of its smallest portable fraction r_i. The number of tokens in a channel is defined as the integer part of its rational state, and a token is actually produced (resp. consumed) by a firing when this integer part increases (resp. decreases) at this firing. The channels are read-blocking, so in order to be valid, a channel state cannot have a negative component.

On the polygraph of Fig. 5, the channels are marked with a circled rational number $\textcircled{r}^{\ddagger}$. This denotes a non-zero initial channel state, and the corresponding vector is $\mathbf{c}^{\ddagger} = [\frac{4}{3}, \frac{1}{2}]^{\mathrm{T}}$. Hence, $\lfloor\frac{4}{3}\rfloor = 1$ token initially occupies channel e_1.

Definition 3 (Channel state) A vector $\mathbf{c} = (c_i) \in \mathbb{Q}^{|E|}$ is a *channel state* of a graph G with topology matrix $\mathbf{\Gamma}$ if for every channel $e_i = \langle v_j, v_k\rangle \in E$, we have $c_i = zr_i$ for some $z \in \mathbb{Z}$. We say that $\lfloor c_i \rfloor$ is the *number of tokens occupying channel* e_i. In addition, we say that channel state $\mathbf{c}$ is *valid* if for all $e_i \in E$ we have $c_i \geqslant 0$.

Remark 1 For any actor v_j, the j^{th} column of $\mathbf{\Gamma}$ gives the rate γ_{ij} for each channel e_i. Therefore, to extract that column, we can use the product $\mathbf{\Gamma} \cdot \mathbf{u}_j$. For any channel state $\mathbf{c}$, the new channel state after an atomic firing of v_j is given by $\mathbf{c} + \mathbf{\Gamma} \cdot \mathbf{u}_j$. For example, with the topology matrix for $\mathcal{P}^{\dagger}$ of Fig. 5, the rates of actor v_2 are given by:

$$\mathbf{\Gamma} \cdot \mathbf{u}_2 = \mathbf{\Gamma} \cdot \begin{bmatrix} 0 & 1 & 0 \end{bmatrix}^{\mathrm{T}} = \begin{bmatrix} -2 & 1 \end{bmatrix}^{\mathrm{T}} .$$

□

3.3 Synchronous Constraints

Timed actors, frequencies, and time unit. A non-empty subset $V_F \subseteq V$ of actors are *timed actors.* Each of them is constrained by a frequency, expressed as a strictly positive natural number. We use a frequency mapping $\omega : V_F \to \mathbb{N}^{>0}$ in order to map each timed actor v_j to its frequency $\omega(v_j)$, denoted ω_j. There is an implicit time unit, and each timed actor $v_j \in V_F$ is supposed to be fired exactly ω_j times per time unit. In order to have a minimal time unit suitable to express the frequencies of all timed actors, we assume that the greatest common divisor (or *gcd*) of the frequencies is 1. This is not limiting, since for any set of frequencies, a suitable time unit can be chosen to fit this constraint. Indeed, the exact value of the time unit is not essential for this formalization.

For example, with the frequencies indicated inside the actors of Fig. 5 and expressed in Hz, i.e. times per second, their gcd is 10. The frequency mapping ω of $\mathcal{P}^{\dagger}$ is defined with those same frequencies expressed in decahertz (daHz), i.e. times per 100ms (1daHz = 10Hz). In this case their gcd is 1, and the implicit time unit is 100ms.

Global clock, resolution, and ticks. In addition, timed actors must fire synchronously with respect to a periodic global clock. The *resolution* of the global clock is a number $\pi \in \mathbb{N}$ of ticks per time unit that is sufficiently frequent to associate to each tick the set of timed actors that must fire at the corresponding date. The resolution π must thus be a multiple of the least common multiple (or *lcm*) of the timed actor frequencies ω_j. We denote by $\mathrm{T}_\pi = \{ 0, 1, \ldots, \pi - 1 \}$ the global clock with resolution π, and by $\tau \in \mathrm{T}_\pi$ one of its ticks.

In our running example $\mathcal{P}^{\dagger}$, we choose the minimal possible resolution of the global clock $\pi = 3$. The period of that global clock is defined by the time unit of ω (i.e. 100ms), and the duration[2] between two consecutive ticks is 100/3ms.

Phase. In addition to a frequency, the timed actors have a phase. Since the firings are synchronous with the global clock, such a delay for the first firing of some timed actors allows us to specify an acceptable latency between the firings of two timed actors. Sometimes, relaxing this acceptable latency by increasing the phase is necessary to respect causality, as explained in Sect. 2.

We use a phase mapping $\varphi : V_F \to \mathrm{T}_\pi$ to map each timed actor v_j to its phase $\varphi(v_j)$, denoted φ_j. The first firing of each timed actor $v_j \in V_F$ then occurs at the tick $\tau = \varphi_j$. The latency is thus expressed as a duration, and the greater the resolution of the global clock, the finer the specification of the phases.

To respect the specified frequency, the firings of a timed actor v_j should occur ω_j times over the π ticks of the global clock, that is, every $(\pi/\omega_j)^{\text{th}}$ tick, starting from tick φ_j. This obviously requires that the phase φ_j of timed actor v_j be strictly less than π/ω_j. Hence, a timed actor v_j is expected to fire synchronously with ticks $\tau \in \mathrm{T}_\pi$ such that $\tau \equiv \varphi_j \pmod{\pi/\omega_j}$.

The phase mapping φ of $\mathcal{P}^{\dagger}$ in Fig. 5 is defined such that v_1 fires on tick 0 and v_3 fires on tick 2. Considering that the duration of a tick is 100/3ms, the phase of v_3 is thus given under the actor in the graphical representation as the duration $\varphi_2.100/3 \approx 67$ms.

Synchronous constraints and clock matrix. The full synchronous constraints are thus defined by a tuple of a frequency mapping, a global clock resolution, and a phase mapping.

Definition 4 (Synchronous constraints) For a graph G with a non-empty set of timed actors $V_F \subseteq V$, *(synchronous) constraints* are defined as a tuple $\Theta = \langle \omega, \pi, \varphi \rangle$

[2] In a practical implementation, the mechanisms deployed to cope with repeating decimals in such durations are implementation dependent (*e.g.* the objects *ratio* and *duration* of the C++ standard *chrono* library).

with a frequency mapping ω, a resolution π, and a phase mapping φ such that:

(i) $\gcd(\{\omega_j \mid v_j \in V_F\}) = 1$;
(ii) $\pi = k \cdot \mathrm{lcm}(\{\omega_j \mid v_j \in V_F\})$ for some $k \in \mathbb{N}^{>0}$;
(iii) $\forall v_j \in V_F,\ \varphi_j < \pi/\omega_j$.

Remark 2 To determine which timed actors v_j are expected to fire on a tick τ, we can use a vector $\mathbf{t}^\tau \in \mathbb{B}^{|V|}$ such that:

$$\forall v_j \in V_F, \quad t_j^\tau = 1 \quad \Leftrightarrow \quad \tau \equiv \varphi_j \ (\mathrm{mod}\ \pi/\omega_j),$$
$$\forall v_k \notin V_F, \quad t_j^\tau = 0.$$

For example, in $\mathcal{P}^\dagger$ of Fig. 5, we have:

$$\mathbf{t}^0 = \begin{bmatrix}1\ 0\ 0\end{bmatrix}^\mathrm{T}, \mathbf{t}^1 = \begin{bmatrix}1\ 0\ 0\end{bmatrix}^\mathrm{T}, \mathbf{t}^2 = \begin{bmatrix}1\ 0\ 1\end{bmatrix}^\mathrm{T}.$$

□

Remark 3 The sum of the vectors $\mathbf{t}^\tau$ for all τ gives a vector whose coordinates indicate the number of times each actor v_j fires synchronously during every period of π ticks. Hence, if we denote this sum $\mathbf{t}$, we have $t_j = \omega_j$ for all $v_j \in V_F$, and $t_k = 0$ for all $v_k \notin V_F$. With the vectors $\mathbf{t}^\tau$ of $\mathcal{P}^\dagger$, we have $\mathbf{t} = \mathbf{t}^0 + \mathbf{t}^1 + \mathbf{t}^2 = \begin{bmatrix}3\ 0\ 1\end{bmatrix}^\mathrm{T}$. □

Synchronous state, current tick, and tracking vector. A *synchronous state* is a tuple $\theta = \langle \tau, \mathbf{a} \rangle$ containing the current tick τ of the global clock, and a *tracking vector* $\mathbf{a} \in \mathbb{N}^{|V|}$ with one element per actor. The tracking vector is used to check that the timed actors respect their synchronous constraints and has the following semantic. An actor $v_j \notin V_F$ without a frequency constraint is ignored in $\mathbf{a}$, so we assume that $a_j = 0$. For a timed actor v_j, the component a_j counts how many times v_j fired at the current tick. So, for a synchronous state to be valid, each component of $\mathbf{a}$ must be less than or equal to the corresponding component in $\mathbf{t}^\tau$.

Definition 5 (Synchronous state) For a graph G with constraints Θ, a *synchronous state* is a tuple $\theta = \langle \tau, \mathbf{a} \rangle$ with a current tick $\tau \in \mathrm{T}_\pi$ and a tracking vector $\mathbf{a} = (a_j) \in \mathbb{N}^{|V|}$ such that for all $v_j \notin V_F$ we have $a_j = 0$. In addition, we say that synchronous state θ is *valid* if $\mathbf{0} \leqq \mathbf{a} \leqq \mathbf{t}^\tau$.

3.4 Language Semantic

Polygraph, state. We now define the notion of polygraph which introduces a basic communication topology, rational communication rates, and synchronous constraints for a subset of actors. A state of a polygraph is a combination of a channel state and a synchronous state.

Definition 6 (Polygraph, state) A *polygraph* is a tuple $\mathcal{P} = \langle G, \mathbf{\Gamma}, \Theta \rangle$ of a graph G, a topology matrix $\mathbf{\Gamma}$ for G, and constraints Θ for G. A *state* of a polygraph $\mathcal{P}$ is a tuple $s = \langle \mathbf{c}, \theta \rangle$ of a channel state $\mathbf{c}$ and a synchronous state θ. We say that state s is *valid* if both $\mathbf{c}$ and θ are valid, in other words, if $\mathbf{c} \geqq \mathbf{0}$ and $\theta = \langle \tau, \mathbf{a} \rangle$ satisfies $\mathbf{a} \leqq \mathbf{t}^\tau$. We denote by S the set of all possible states of $\mathcal{P}$.

State transitions. The only possible transitions from one state to another are the firing of an actor or a tick of the global clock. The effect of the firing of an actor on the channel state is to add its rates to the respective components of all the channels, as per Remark 1. In addition, the firing of a timed actor is tracked by an increment of its component in the tracking vector, and has no effect on the current tick. When the global clock ticks, the channel state is not changed, the current tick is adjusted, and the tracking vector is reset.

Definition 7 (Fire) For a polygraph $\mathcal{P} = \langle G, \mathbf{\Gamma}, \Theta \rangle$, the mapping fire : $V \times S \to S$ maps an actor v_j and a state $s = \langle \mathbf{c}, \langle \tau, \mathbf{a} \rangle \rangle$ to a state $s' = \langle \mathbf{c}', \langle \tau', \mathbf{a}' \rangle \rangle$ such that we have:

(i) $\mathbf{c}' = \mathbf{c} + \mathbf{\Gamma} \cdot \mathbf{u}_j$;
(ii) $\tau' = \tau$;
(iii) if $v_j \in V_F$, then $\mathbf{a}' = \mathbf{a} + \mathbf{u}_j$;
(iv) if $v_j \notin V_F$, then $\mathbf{a}' = \mathbf{a}$.

Remark 4 For two consecutive firings of any actors v_j and v_k from a state $s = \langle \mathbf{c}, \langle \tau, \mathbf{a} \rangle \rangle$, the resulting state $s'' = \langle \mathbf{c}'', \langle \tau'', \mathbf{a}'' \rangle \rangle$ does not depend on the order of the firings, and we have $\mathbf{c}'' = \mathbf{c} + \mathbf{\Gamma} \cdot (\mathbf{u}_j + \mathbf{u}_k)$ and $\tau'' = \tau$. In addition, if both v_j and v_k are timed actors, we have $\mathbf{a}'' = \mathbf{a} + (\mathbf{u}_j + \mathbf{u}_k)$. This property can be generalized to any finite number of consecutive firings. □

Definition 8 (Tick) For a polygraph $\mathcal{P} = \langle G, \mathbf{\Gamma}, \Theta \rangle$, the mapping tick : $S \to S$ maps a state $s = \langle \mathbf{c}, \langle \tau, \mathbf{a} \rangle \rangle$ to a state $s' = \langle \mathbf{c}', \langle \tau', \mathbf{a}' \rangle \rangle$ such that we have:
(i) $\mathbf{c}' = \mathbf{c}$;
(ii) $\tau' = (\tau + 1) \bmod \pi$;
(iii) $\mathbf{a}' = \mathbf{0}$.

Executions. The state of $\mathcal{P}$ can evolve by successive application of either $fire$ or $tick$. An *execution* of $\mathcal{P}$ is a finite sequence of such applications starting from an *initial state*. The following definition of theoretical executions allows for any ordering of the applications of $fire$ and $tick$, which is useful for reasoning about a polygraph in an abstract way. We refine executions to synchronous ones later in Def. 10, which contain valid synchronous states only (the timed actors fire exactly once at the expected ticks), and to non-blocking executions in Def. 11, where channels must always carry a positive number of tokens. Executions with both properties are then valid.

Definition 9 (Execution) An *execution* of a polygraph $\mathcal{P}$ is a sequence of states $\sigma = s^1 \cdots s^n \in S^+$, such that for $1 \leqslant l < n$ we have:

(i) $s^{l+1} = \text{fire}(v_j, s^l)$ for some $v_j \in V$;
(ii) or $s^{l+1} = \text{tick}(s^l)$.

Remark 5 From the conditions of Def. 7 and Def. 8, for a sequence of states $s^1 \cdots s^n$ we can uniquely determine for each state after s^1 if it results from an application of fire or tick on the previous state, and thus determine whether that sequence is an execution. For that reason, the execution notation includes only a sequence of states without fire or tick transitions between states. □

Remark 6 The number of firings of actors in an execution $\sigma = s^1 \cdots s^n$ can be represented by a tracking vector $\mathbf{y}^\sigma = (y_j^\sigma) \in \mathbb{N}^{|V|}$. For each v_j, the component y_j^σ gives the number of times v_j fires in σ. The number of ticks of the global clock in σ is denoted z^σ. In other words,

$$y_j^\sigma = \text{card}\,\{\, l \mid 1 \leqslant l < n,\ s^{l+1} = \text{fire}(v_j, s^l)\,\},$$
$$z^\sigma = \text{card}\,\{\, l \mid 1 \leqslant l < n,\ s^{l+1} = \text{tick}(s^l)\,\}.$$

To isolate the number of firings of the timed actors, we denote $\mathbf{a}^\sigma$ the tracking vector $(a_j^\sigma) \in \mathbb{N}^{|V|}$ such that $a_j^\sigma = y_j^\sigma$ if $v_j \in V_F$, and $a_j^\sigma = 0$ if $v_j \notin V_F$. □

Remark 7 Let $\sigma = s^1 \cdots s^n$ be an execution with initial state $s^1 = \langle \mathbf{c}^1, \langle \tau^1, \mathbf{a}^1 \rangle\rangle \in S$ and with final state $s^n = \langle \mathbf{c}^n, \langle \tau^n, \mathbf{a}^n \rangle\rangle \in S$. Using Remark 4 and Def. 8, Condition (i), since a tick does not modify a channel state, we can deduce that $\mathbf{c}^n = \mathbf{c}^1 + \mathbf{\Gamma} \cdot \mathbf{y}^\sigma$. Moreover, from Def. 8, Condition (ii), we deduce $\tau^n = (\tau^1 + z^\sigma) \bmod \pi$. By Def. 8, Condition (iii), the tracking vector $\mathbf{a}^n$ counts the firings of timed actors that occurred since the last tick in σ. If the very last transition in σ is a tick, then we obviously have $\mathbf{a}^n = \mathbf{0}$. □

Remark 8 For any two executions σ_1 and σ_2 of respective length n_1 and n_2, if we have $\sigma_1[n_1] = \sigma_2[1]$, we can see σ_2 as a continuation of σ_1, and we denote $\sigma_1 \rhd \sigma_2$ the execution $\sigma_3 = \sigma_1[1] \cdots \sigma_1[n_1] \cdot \sigma[2] \cdots \sigma_2[n_2]$. In this case, from Remark 6, we have $\mathbf{y}^{\sigma_3} = \mathbf{y}^{\sigma_1} + \mathbf{y}^{\sigma_2}$ and $z^{\sigma_3} = z^{\sigma_1} + z^{\sigma_2}$. □

The sequence of states in Fig. 6 represents an execution $\sigma_1 = s^1 \cdots s^{33}$ of the polygraph of Fig. 5, with the detail of the channel and synchronous states. At the end of the row for each state s^l, we give the tracking vector $\mathbf{y}^\sigma$ and tick count z^σ of the execution $\sigma = s^1 \cdots s^l$. Some lines are eluded, their contents can be inferred from the tracking vector and tick count of the following row, and from the corresponding path in Fig. 8.

Notice that if we denote $\sigma' = s^1 \cdots s^{17}$ and $\sigma'' = s^{17} \cdots s^{33}$, then σ_1 is built by continuing σ' with σ'' as per Remark 8, and $\sigma_1 = \sigma' \rhd \sigma''$. From Remark 6, we can see that the applications of *fire* and *tick* in σ'', eluded in Fig. 6, are the same as those in σ'. Indeed, the tracking vector and tick count for the row of s^{33} shows that the same number of firings and ticks[3] occurred since s^{17}. We then have $\mathbf{y}^{\sigma'} = \mathbf{y}^{\sigma''}$ and $z^{\sigma'} = z^{\sigma''}$, and by Remark 8, we have $\mathbf{y}^{\sigma_1} = \mathbf{y}^{\sigma'} + \mathbf{y}^{\sigma''}$ and $z^{\sigma_1} = z^{\sigma'} + z^{\sigma''}$.

However, in practice, with the synchronous constraints and the read-blocking policy on the channels, there are some restrictions, as illustrated in Fig. 7 with another execution of our running example $\mathcal{P}^\dagger$ from a different initial state. (To avoid any confusion, notice that the same notation s^i refers here to different states than in Fig. 6.)

Let us first explain when the firing of a timed actor v_j is admissible in a state $s = \langle \mathbf{c}, \langle \tau, \mathbf{a} \rangle\rangle$. In this case, v_j may fire only if the current tick τ is one of its firing ticks, i.e. $t_j^\tau = 1$ as per Remark 2. Since it must fire exactly once on such a tick, an additional constraint for a firing of v_j is that it has not fired yet, i.e. its component in the tracking vector $\mathbf{a}$ is $a_j = 0$. These constraints are respected if the resulting synchronous state after the firing is valid. Ticking the clock in s is also subject to constraints: each timed actor v_j supposed to fire synchronously with τ should have done so exactly once, i.e. its coordinate in the tracking vector $\mathbf{a}$ is $a_j = t_j^\tau = 1$. This constraint ensures that the clock cannot tick too early, i.e. before all expected timed actors fire. In a synchronous execution, these conditions must be verified for every transition.

Definition 10 (Synchronous execution) Consider an execution $\sigma = s^1 \cdots s^n \in S^+$ of a polygraph $\mathcal{P}$, with $s^l = \langle \mathbf{c}^l, \theta^l \rangle$ and $\theta^l = \langle \tau^l, \mathbf{a}^l \rangle$ for all $1 \leqslant l \leqslant n$. The execution σ is *synchronous* if synchronous state θ^1 is valid and for all $1 \leqslant l < n$ we have:

(i) $s^{l+1} = \text{fire}(v_j, s^l)$ for some $v_j \in V$ and synchronous state θ^{l+1} is valid;
(ii) or $s^{l+1} = \text{tick}(s^l)$ and $\mathbf{a}^l = \mathbf{t}^\tau$.

Since the novelty in PolyGraph resides mainly in the synchronous constraints, it is worthwhile to provide additional insight on their structure and properties in the

[3] In addition, $s1 = s17 = s33$ and the reader familiar with SDF literature will recognize the notion of *iteration*.

s^l	c_1^l:tokens	c_2^l:tokens	τ^l	a_1^l	a_2^l	a_3^l	$\mathbf{y}^\sigma$	z^σ
s^1	$\frac{4}{3}$: ①	$\frac{1}{2}$:	0	0	0	0	$[0,0,0]$	0
$s^2 = \text{fire}(v_1, s^1)$	$\frac{5}{3}$: ①	$\frac{1}{2}$:	0	1	0	0	$[1,0,0]$	0
$s^3 = \text{tick}(s^2)$	$\frac{5}{3}$: ①	$\frac{1}{2}$:	1	0	0	0	$[1,0,0]$	1
$s^4 = \text{fire}(v_1, s^3)$	$\frac{6}{3}$: ①②	$\frac{1}{2}$:	1	1	0	0	$[2,0,0]$	1
$s^5 = \text{tick}(s^4)$	$\frac{6}{3}$: ①②	$\frac{1}{2}$:	2	0	0	0	$[2,0,0]$	2
$s^6 = \text{fire}(v_2, s^5)$	$\frac{0}{3}$:	$\frac{3}{2}$: ①	2	0	0	0	$[2,1,0]$	2
$s^7 = \text{fire}(v_1, s^6)$	$\frac{1}{3}$:	$\frac{3}{2}$: ①	2	1	0	0	$[3,1,0]$	2
$s^8 = \text{fire}(v_3, s^7)$	$\frac{1}{3}$:	$\frac{2}{2}$: ①	2	1	0	1	$[3,1,1]$	2
$s^9 = \text{tick}(s^8)$	$\frac{1}{3}$:	$\frac{2}{2}$: ①	0	0	0	0	$[3,1,1]$	3
...	...	...	...	...	...	...	...	...
$s^{14} = \text{tick}(s^{13})$	$\frac{3}{3}$: ③	$\frac{2}{2}$: ①	2	0	0	0	$[5,1,1]$	5
$s^{15} = \text{fire}(v_1, s^{14})$	$\frac{4}{3}$: ③	$\frac{2}{2}$: ①	2	1	0	0	$[6,1,1]$	5
$s^{16} = \text{fire}(v_3, s^{15})$	$\frac{4}{3}$: ③	$\frac{1}{2}$:	2	1	0	1	$[6,1,2]$	5
$s^{17} = \text{tick}(s^{16})$	$\frac{4}{3}$: ③	$\frac{1}{2}$:	0	0	0	0	$[6,1,2]$	6
...	...	...	...	...	...	...	...	...
$s^{33} = \text{tick}(s^{32})$	$\frac{4}{3}$: ⑤	$\frac{1}{2}$	0	0	0	0	$[12,2,4]$	12

Fig. 6: A live execution $s^1 \cdots s^{33}$ of the polygraph $\mathcal{P}^\dagger$ from Fig. 5. For the channel states, there are as many circled numbers ⓝ as there are tokens occupying the channel in that state, and the number n represents the rank of the token in FIFO order on the channel.

s^l	c_1^l	c_2^l	τ^l	a_1^l	a_2^l	a_3^l	Comments
s^1	$\frac{2}{3}$	$\frac{0}{2}$	2	0	0	0	Initial state.
$s^2 = \text{fire}(v_1, s^1)$	$\frac{3}{3}$	$\frac{0}{2}$	2	1	0	0	Valid transition (synchronous and non-blocking)
$s^3 = \text{fire}(v_1, s^2)$	$\frac{4}{3}$	$\frac{0}{2}$	2	2	0	0	⚠ Non-synchronous firing of v_1 ($a_1^3 > t_1^2$)
$s^4 = \text{tick}(s^3)$	$\frac{4}{3}$	$\frac{0}{2}$	0	0	0	0	⚠ Non-synchronous tick ($\mathbf{a}^3 \neq \mathbf{t}^2$)
$s^5\text{fire}(v_2, s^4)$	$-\frac{2}{3}$	$\frac{2}{2}$	0	0	0	0	⚠ Blocking firing of v_2 ($c_1^5 < 0$)

Fig. 7: An execution $s^1 \cdots s^5$ of polygraph $\mathcal{P}^\dagger$ from Fig. 5, illustrating all possible kinds of invalid transitions: actor v_1 fires more than expected at the current tick $\tau = 2$ of state s^2; the tick in s^3 is invalid; and actor v_2 fires with an insufficient input channel state in state s^4. The first two cases are excluded for a synchronous execution (cf. Def. 10), while the last case is excluded for a non-blocking execution (cf. Def. 11).

following remarks. They all rely on the above definition of synchronous executions, and will be used in the proofs of the next section.

Remark 9 The firing of an actor $v_j \notin V_F$ without constraints does not impact the synchronous property of an execution, since it does not modify the synchronous state (cf. Def. 7, Conditions (ii) and (iv)). □

Remark 10 It follows from the Conditions (i) and (ii) in Def. 10 that the number of ticks in a synchronous execution is constrained by the number of firings of the timed actors, and conversely. For two synchronous executions σ and σ′ starting from the same initial state s, if they have the same number of ticks $z^\sigma = z^{\sigma'}$, we can easily show by induction for $1 \leqslant i \leqslant z^\sigma$ that by the i^{th} occurrence of a tick in σ and by the i^{th} occurrence of a tick in σ′, the timed actors fired the same number of times in both executions σ and σ′. The number of firings of the timed actors can thus differ only in the suffixes of σ and σ′ starting from the last *tick* application, and it is obviously bounded by the number of expected firings in the current tick. If the number of firing of timed actors is the same $\mathbf{a}^\sigma = \mathbf{a}^{\sigma'}$ in both executions, it is then clear that it is the same in these suffixes, and by Remark 4 the tracking vectors in their final states are equal. We know from Remark 7 that the current tick is also the same in their final states, and thus their final synchronous states are equal. Finally, if the number of firings of actors is the same $\mathbf{y}^\sigma = \mathbf{y}^{\sigma'}$ in both executions, also from Remark 7, their final chan-

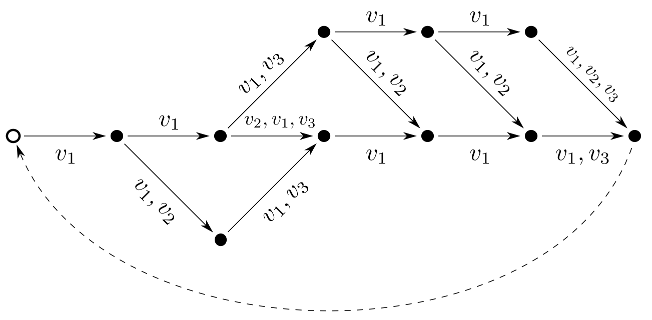


Fig. 8: Selected valid (and also live) executions of polygraph $\mathcal{P}^\dagger$ from Fig. 5 starting from the initial state s^1 of Fig. 6. The horizontal execution represents the execution $s^1 \cdots s^{17}$ of Fig. 6. The white circled dot represents that initial state, black dots represent tick applications, and plain arcs connecting them represent the firings of the actors mentioned in their label, in that order. The dashed arc illustrates that the execution can be repeated. (In this example the state after the last tick is the initial state.)

nel states are equal, and the final state is the same in both executions. □

To comply with the read-blocking policy on the channels, for an actor to fire, there must be enough tokens on its input channels. It can consume 0 tokens from a channel only when its rational communication rate allows such a firing. In any case, the resulting channel state cannot become strictly negative. Hence, to fire an actor v_j in a state $s = \langle \mathbf{c}, \langle \tau, \mathbf{a} \rangle \rangle$, for each input channel e_i of v_j, we require that the channel state c_i be large enough to avoid reaching a negative state, i.e. $c_i + \gamma_{ij} \geqslant 0$ (since the rate γ_{ij} is negative), or equivalently $c_i \geqslant |\gamma_{ij}|$. This constraint is respected if after the firing of v_j the resulting channel state is valid. No additional condition is required after a tick since it does not modify the channel state.

Definition 11 (Non-blocking execution) Let $\sigma = s^1 \cdots s^n \in S^+$ be an execution of a polygraph $\mathcal{P}$, with $s^l = \langle \mathbf{c}^l, \theta^l \rangle$ and $\theta^l = \langle \tau^l, \mathbf{a}^l \rangle$ for all $1 \leqslant l \leqslant n$. The execution σ is *non-blocking* if channel state $\mathbf{c}^1$ is valid and for all $1 \leqslant l < n$ we have:

(i) $s^{l+1} = \text{fire}(v_j, s^l)$ for some $v_j \in V$ and channel state $\mathbf{c}^{l+1}$ is valid;
(ii) or $s^{l+1} = \text{tick}(s^l)$.

An execution for which these conditions are not verified is said to be *blocking.*

Valid periodic schedule. In this section, a reader familiar with SDF will recognize a generalization of the definitions of the classical consistency and liveness properties of SDF graphs to polygraphs. For a given polygraph $\mathcal{P}$, finite executions that are both synchronous and non-blocking are called *valid.* The conditions for an execution to be valid impose a (partial) ordering of the actor firings and the global clock ticks. The timed actors must fire between the appropriate tick transitions (cf. Def. 10), and a producer must fire a sufficient number of times for a consumer to be able to fire in turn (cf. Def. 11).

To illustrate this, in Fig. 8, possible valid executions of the example polygraph $\mathcal{P}^\dagger$ from Fig. 5 are represented (to keep the representation compact, we limited it to some representative firing orderings). Notice how v_2 may fire anytime after the second firing of v_1, and must fire before the second firing of v_3, regardless of the current tick.

Definition 12 (Valid execution) In a polygraph $\mathcal{P}$, an execution σ is *valid* if it is both synchronous and non-blocking.

As mentioned in the introduction, only valid periodic executions are of practical interest in our context, as they can be statically analyzed. The results of that static analysis can be extrapolated to any number of repetitions of that period, giving a long run estimation. For a given implementation and the resources available on the physical platform running it, finding the permutation that yields the best performance for some criterion is also valuable, since scheduling the system according to this pattern will be more efficient.

To be scheduled in practice, a valid periodic execution should run in finite memory without deadlock, for any number of repetitions.

Without the synchronous and non-blocking properties, any execution $\sigma = s^1 \cdots s^n$ having at least one firing and returning to its initial state (that is, with $s^n = s^1$) can be repeated any number of times to produce an execution $\sigma' = \sigma \rhd \ldots \rhd \sigma$ (cf. Fig. 6 showing an execution built from two repetitions of another one, and Fig. 8 which illustrates how the concept can be repeated). This means that we can build such an execution σ' of arbitrary length, mastering the intervals of values of the channel states. Indeed, this interval will be exactly the one observed for σ. This notion corresponds to the classical consistency property of existing data flow formalisms.

In our context, as explained in Sect. 2, synchronous constraints must be taken into account to generalize the consistency property to polygraphs in Theorem 1.

Definition 13 (Consistent) In a polygraph $\mathcal{P}$, an execution $\sigma = s^1 \cdots s^n$ is called *consistent* if it is synchronous and we have $s^1 = s^n$ and $\mathbf{y}^\sigma \neq \mathbf{0}$. If there is such an execution for $\mathcal{P}$, we say that $\mathcal{P}$ is *consistent.*

Remark 11 The constraint for a consistent execution σ to contain at least one firing (i.e. $y_j^\sigma \neq 0$ for some actor v_j) implies that all actors will fire at least once in σ (i.e. $y_j^\sigma \neq 0$ for all actors v_j). Indeed, from Remark 7, for any channel $e_i = \langle v_j, v_k\rangle \in E$ with initial state c_i in σ, the final channel state is $c_i' = c_i + y_j^\sigma \gamma_{ij} + y_k^\sigma \gamma_{ik}$. Obviously, since in a consistent execution we have the same initial and final channel state, we must have $y_j^\sigma \gamma_{ij} + y_k^\sigma \gamma_{ik} = 0$. If $y_j^\sigma \neq 0$ for some actor v_j, then by Def. 2, Condition (ii), all the actors v_k in its neighborhood must fire a sufficient number of times $y_k^\sigma > 0$ to respect this balance equation, and since the system graph is connected, by transitivity, this is true for all actors. We will see in Th. 1 another interesting property of a consistent execution σ: the global clock ticks in σ some number r of complete periods, that is, $r \cdot \pi$ times, for some $r \in \mathbb{N}^{>0}$. $\square$

If a consistent execution σ is non-blocking, all read operations on the channels can finish without blocking, and this is true for any repetition σ'. The channel states remain strictly positive and bounded, thus the number of tokens occupying each channel at each moment of time remains bounded as well. This notion corresponds to the classical liveness property of existing data flow formalisms, that we generalize to polygraphs in Theorem 2. Since the synchronous constraints are already taken into account in the consistency property, the non-blocking property is all that is needed to have a periodic execution that can be analyzed and implemented in practice.

Definition 14 (Live) In a polygraph $\mathcal{P}$, an execution $\sigma = s^1 \cdots s^n$ is called *live* if it is consistent and non-blocking. In other words, σ is live if it is valid, and we have $s^1 = s^n$ and $\mathbf{y}^\sigma \neq \mathbf{0}$. If there is such an execution for $\mathcal{P}$, we say that $\mathcal{P}$ is *live from (initial state)* s^1.

For example, the polygraph $\mathcal{P}^\dagger$ from Fig. 5, with initial state $\langle \mathbf{c}^\ddagger, \langle 0, \mathbf{0}\rangle\rangle$, has at least one live execution, the one shown in Fig. 6.

4 PolyGraph Language Properties

As explained in the previous section, it is interesting to a system designer to be able to decide if the modeled system has live executions. We define here the theoretical foundation to check their existence for a given model in practice. The reader familiar with SDF will recognize statements from the theorems proved in [24], generalized to account for the synchronous property of executions introduced in PolyGraph.

4.1 Consistency Property

Based on existing results from [24, Th.1] and [28, Th.1], if we denote $s^1 = \langle \mathbf{c}^1, \theta^1\rangle$ and $s^n = \langle \mathbf{c}^n, \theta^n\rangle$, we know that a necessary and sufficient condition to build an execution $s^1 \cdots s^n$ such that $\mathbf{c}^1 = \mathbf{c}^n$ is that there is a non-trivial solution $\mathbf{x}$ to $\mathbf{\Gamma} \cdot \mathbf{x} = \mathbf{0}$. Indeed, for any such solution $\mathbf{x}$, for any synchronous execution σ with a number of firings $y_j^\sigma = x_j$ per actor v_j, from Remark 7, the resulting channel state is the same $\mathbf{c}^1 + \mathbf{\Gamma} \cdot \mathbf{x} = \mathbf{c}^n$.

To extend this result to polygraphs, we need to make sure that it is possible to build a synchronous execution also returning to the initial synchronous state $\theta^1 = \theta^n$ (that should be thus valid). We will prove that since it comes back to the initial state (and therewith, to the initial tick), that execution must have exactly some strictly positive number r of repetitions of global clock periods. For that execution to be synchronous, the firings of timed actors must be synchronous with the associated ticks (given by the vectors $\mathbf{t}^\tau$ and $\mathbf{t}$, inferred from the definition of $\mathcal{P}$ as per Remarks 2 and 3). The other actors $v_k \notin V_F$ do not have additional constraints in a synchronous execution, they just have to fire a number x_k of times. To separate their components in $\mathbf{x}$, we define the set $Y \subset \mathbb{N}^{|V|}$ of vectors $\mathbf{y}$ such that $y_k = 0$ for any $v_k \in V_F$.

For example, in the case of $\mathcal{P}^\dagger$ from Fig. 5, we know from Remark 3 that $\mathbf{t} = \begin{bmatrix}3 \ 0 \ 1\end{bmatrix}^\mathrm{T}$. With $r = 2$ and $\mathbf{y} = \begin{bmatrix}0 \ 1 \ 0\end{bmatrix}^\mathrm{T}$, the vector $\mathbf{x} = \mathbf{y} + r \cdot \mathbf{t} = \begin{bmatrix}6 \ 1 \ 2\end{bmatrix}^\mathrm{T}$ satisfies $\mathbf{\Gamma} \cdot \mathbf{x} = \mathbf{0}$. Notice that in Fig. 6, in state s^{17} the tracking vector (cf. column $\mathbf{y}^\sigma$) equals $\mathbf{x}$, and there are $2 \cdot \pi = 6$ ticks (cf. column z^σ). In s^{33}, the tracking vector equals $2 \cdot \mathbf{x} = \begin{bmatrix}12 \ 2 \ 4\end{bmatrix}$, and there are $4 \cdot \pi = 12$ ticks.

Theorem 1 (Consistency theorem) *For a polygraph $\mathcal{P}$, the following conditions are equivalent:*

(i) $\mathcal{P}$ is consistent;
(ii) there exists a non-trivial solution $\mathbf{x} \in \mathbb{N}^{|V|}$ to the equation $\mathbf{\Gamma} \cdot \mathbf{x} = \mathbf{0}$ such that $\mathbf{x} = \mathbf{y} + r \cdot \mathbf{t}$ for some $\mathbf{y} \in Y$ and $r \in \mathbb{N}^{>0}$.

Any such solution $\mathbf{x}$ is called a repetition vector *of $\mathcal{P}$. If polygraph $\mathcal{P}$ is consistent, there exists a* minimal repetition vector $\mathbf{x}$ *such that for any other repetition vector $\mathbf{x}'$, $\mathbf{x}' = p \cdot \mathbf{x}$ for some $p \in \mathbb{N}^{>0}$.*

Proof First, we prove that (ii) implies (i). Suppose that there exists such a solution $\mathbf{x}$. Then we can decompose it, for some $\mathbf{y} \in Y$ and $r \in \mathbb{N}^{>0}$, as follows:

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

The proof is organized as follows. The required consistent execution will be obtained by constructing sub-executions corresponding to this decomposition, and then using them to build the consistent execution.

To build the sub-executions, we rely on the fact that given a vector with one component per actor, it is possible to define an execution with as many firings of each actor as specified by that vector, and show that, under certain conditions, such an execution is synchronous (Claim 1 below). Based on this result, we construct a synchronous execution built from a valid initial synchronous state using vector $\mathbf{y}$ (Claim 2). Then, we construct a synchronous execution from a valid initial synchronous state $\theta = \langle \tau, \mathbf{0} \rangle$ using a vector $\mathbf{t}^\tau$ and ending with an additional tick (Claim 3). Finally, we compose these sub-executions according to the decomposition of $\mathbf{x}$ given above, and show that it is synchronous and returns to the initial state (Claim 4 and the end of the proof). Note that we do not care about the ordering of the firings of producers with respect to consumers, since we do not need the execution to be non-blocking.

Claim (1) Let $s = \langle \mathbf{c}, \theta \rangle \in S$ such that synchronous state $\theta = \langle \tau, \mathbf{a} \rangle$ is valid, and $\mathbf{w} \in \mathbb{N}^{|V|}$ a vector such that for all $v_j \in V_F$, we have $a_j + w_j \leqslant t^\tau_j$. Let σ be an execution such that $\sigma[1] = s$, $\mathbf{y}^\sigma = \mathbf{w}$, and $z^\sigma = 0$. Then σ is synchronous.

In any such $\sigma = s^1 \cdots s^n$ with $s^1 = s$, there are w_j firings of each actor $v_j \in V$ (without any constraint on their order), and there is no tick (since $z^\sigma = 0$). Let us denote $s^l = \langle \mathbf{c}^l, \theta^l \rangle$ and $\theta^l = \langle \tau^l, \mathbf{a}^l \rangle$ for $1 \leqslant l \leqslant n$. To prove that σ is synchronous, since it has no ticks, we just have to show (by Def. 10) that synchronous state θ^l is valid for any $1 \leqslant l \leqslant n$. For $\theta^1 = \theta$, it is assumed in the statement.

First of all, since there is no tick in σ, and since the *tick* operation is the only one modifying the current tick (cf. Def. 8, Condition (ii)), the current tick is the same $\tau^l = \tau$ in all its states.

For any $1 \leqslant l \leqslant n$, for any timed actor $v_j \in V_F$, we have $a_j = a^1_j \leqslant a^l_j \leqslant a^n_j = a_j + w_j$ by construction of σ (cf. Def. 7, Condition (ii)) since a component of the tracking vector $\mathbf{a}$ is incremented when the corresponding actor fires, and v_j fires w_j times. Since by assumption $a_j + w_j \leqslant t^\tau_j$, we deduce $a^l_j \leqslant t^\tau_j$. Hence, $\mathbf{a}^l \leqq \mathbf{t}^\tau$, and synchronous state θ^l is valid (cf. Def. 5).

Notice that for a timed actor $v_j \in V_F$, the assumption $a_j + w_j \leqslant t^\tau_j$ implies $w_j \leqslant 1$, that is, a timed actor v_j fires in σ at most once, as it cannot fire several times on the same tick.

Claim (2) For any $\mathbf{y} \in Y$ and any state $s^1 = \langle \mathbf{c}^1, \theta^1 \rangle \in S$, such that synchronous state θ^1 is valid, there exists a synchronous execution $\sigma = s^1 \cdots s^n$ with final state $s^n = \langle \mathbf{c}^n, \theta^n \rangle$ such that $\mathbf{c}^n = \mathbf{c}^1 + \mathbf{\Gamma} \cdot \mathbf{y}$ and $\theta^n = \theta^1$.

Consider any execution σ such that $\sigma[1] = s^1$, $\mathbf{y}^\sigma = \mathbf{y}$, and $z^\sigma = 0$. Since $\mathbf{y} \in Y$, the components corresponding to timed actors are 0, thus we can apply Claim 1 with $\mathbf{w} = \mathbf{y}$ and deduce that σ is synchronous. The resulting channel state is $\mathbf{c}^1 + \mathbf{\Gamma} \cdot \mathbf{y}$ as per Remark 7. The resulting synchronous state is θ^1 since the firings of actors $v_j \notin V_F$ do not modify the synchronous state as per Def. 7, Condition (ii) and (iv).

Claim (3) For any state $s^1 = \langle \mathbf{c}^1, \theta^1 \rangle \in S$ with synchronous state $\theta^1 = \langle \tau, \mathbf{0} \rangle$ (that is obviously valid), there exists a synchronous execution $\sigma = s^1 \cdots s^n$ with final state $s^n = \langle \mathbf{c}^n, \theta^n \rangle$ such that $\mathbf{c}^n = \mathbf{c}^1 + \mathbf{\Gamma} \cdot \mathbf{t}^\tau$ and $\theta^n = \langle (\tau + 1) \bmod \pi, \mathbf{0} \rangle$.

Consider any execution σ' such that $\sigma'[1] = s^1$, $\mathbf{y}^{\sigma'} = \mathbf{t}^\tau$, and $z^{\sigma'} = 0$. With $\mathbf{w} = \mathbf{t}^\tau$, the assumptions of Claim 1 are verified, and we deduce that σ' is synchronous. Let us denote the resulting state of σ' by $s' = \langle \mathbf{c}', \langle \tau, \mathbf{a}' \rangle \rangle$. We have $\mathbf{c}' = \mathbf{c}^1 + \mathbf{\Gamma} \cdot \mathbf{t}^\tau$ and $\mathbf{a}' = \mathbf{0} + \mathbf{t}^\tau = \mathbf{t}^\tau$ as per Remark 7. We extend σ' by a tick and obtain a new execution $\sigma = s^1 \cdots s^n$ with $s^{n-1} = s'$ and $s^n = \text{tick}(s^{n-1})$. Since $\mathbf{a}' = \mathbf{t}^\tau$, by Def. 10, σ is synchronous as well. In addition, we have $s^n = \langle \mathbf{c}^n, \langle \tau^n, \mathbf{a}^n \rangle \rangle$ with $\mathbf{c}^n = \mathbf{c}'$, $\tau^n = (\tau + 1) \bmod \pi$, and $\mathbf{a}^n = \mathbf{0}$ as per Def. 8.

Claim (4) For any state $s^1 = \langle \mathbf{c}^1, \theta^1 \rangle \in S$ with synchronous state $\theta^1 = \langle 0, \mathbf{0} \rangle$ (that is obviously valid), there exists a synchronous execution $\sigma = s^1 \cdots s^n$ with final state $s^n = \langle \mathbf{c}^n, \theta^n \rangle$ such that $\mathbf{c}^n = \mathbf{c}^1 + \mathbf{\Gamma} \cdot \mathbf{t}$ and $\theta^n = \theta^1$.

Let $\sigma_0, \ldots, \sigma_{\pi-1}$ be the synchronous executions constructed as per Claim 3 for ticks $\tau = 0, \ldots, \pi - 1$, where the initial state of σ_0 is s^1, and for any $i > 0$, the initial state of σ_i is the final state of σ_{i-1}. Let $\sigma = s^1 \cdots s^n$ denote the combined execution $\sigma_0 \rhd \ldots \rhd \sigma_{\pi-1}$. It is synchronous since all $\sigma_0, \ldots, \sigma_{\pi-1}$ are synchronous. It follows from Claim 3 that $\mathbf{c}^n = \mathbf{c}^1 + \mathbf{\Gamma} \cdot (\mathbf{t}^0 + \cdots + \mathbf{t}^{\pi-1})$, and $\theta^n = \theta^1$. Equivalently, from Remark 3, we have $\mathbf{c}^n = \mathbf{c}^1 + \mathbf{\Gamma} \cdot \mathbf{t}$.

We are ready to prove (ii) implies (i). Consider some state $s^1 = \langle \mathbf{c}^1, \theta^1 \rangle \in S$ with synchronous state $\theta^1 = \langle 0, \mathbf{0} \rangle$ (that is obviously valid). Let σ_0 be the synchronous execution built with initial state s^1 as per Claim 2 with $\mathbf{y}^{\sigma_0} = \mathbf{y}$, and let $\sigma_1, \ldots, \sigma_r$ be the synchronous executions constructed as per Claim 4, where for any $i > 0$, the initial state of σ_i is the final state of σ_{i-1}. By Claim 2 and 4, each of these initial states is indeed of the form $s = \langle \mathbf{c}, \langle 0, \mathbf{0} \rangle \rangle \in S$ for some channel state $\mathbf{c}$.

Let $\sigma = s^1 \cdots s^n$ denote the combined execution $\sigma_0 \rhd \ldots \rhd \sigma_r$. It is synchronous since all $\sigma_0, \ldots, \sigma_r$

are synchronous. Denote $s^n = \langle \mathbf{c}^n, \theta^n \rangle$. It follows from Claims 2, 3 and the condition (ii) that

$$\mathbf{c}^n = \mathbf{c}^1 + \mathbf{\Gamma} \cdot (\mathbf{y} + r \cdot \mathbf{t}) = \mathbf{c}^1 + \mathbf{\Gamma} \cdot \mathbf{x} = \mathbf{c}^1$$

and $\theta^n = \theta^1$. Therefore, $s^n = s^1$.

Let us give a sketch of proof that (i) implies (ii). Assume there is a consistent execution $\sigma = s^1 \cdots s^n$ for $\mathcal{P}$, let us denote its initial state by $s^1 = \langle \mathbf{c}^1, \theta^1 \rangle \in S$ with a synchronous state $\theta^1 = \langle \tau^1, \mathbf{a}^1 \rangle$. Since σ returns to its initial channel state, by Remark 7, the tracking vector $\mathbf{y}^\sigma$ verifies $\mathbf{\Gamma} \cdot \mathbf{y}^\sigma = \mathbf{0}$. Without ticks, the firings of timed actors $v_j \in V_F$ monotonically increase the components of the tracking vector in the successive states (cf. Def. 7, Condition (iii)). As σ has to return to the initial tracking vector, it must have ticks since the only way to reset the tracking vector is to tick the global clock (cf. Def. 8, Condition (iii)). As σ has to return to the initial tick, from Def. 8, Condition (ii), σ must have $r \cdot \pi$ ticks in order to return to the same tick with $\tau^1 = (\tau^1 + r \cdot \pi) \bmod \pi$. Then there must be $\sum_{\tau=0}^{\pi-1} r \cdot \mathbf{t}^\tau = r \cdot \mathbf{t}$ firings of timed actors (see Remark 3). Indeed, this is obvious if $\mathbf{a}^1 = 0$ and $\mathbf{t}^{\tau^1}$ firings of timed actors occur before the very first tick. Otherwise notice that in σ, the $\mathbf{t}^{\tau^1} - \mathbf{a}^1$ firings of timed actors before the very first tick, together with the $\mathbf{a}^1$ firings of timed actors (required to come back to the initial synchronous state) after the very last tick, give the sum of $\mathbf{t}^{\tau^1}$.

Combining these constraints, in order to return to both initial channel state and synchronous state, σ must satisfy $\mathbf{\Gamma} \cdot \mathbf{y}^\sigma = \mathbf{0}$, $y_j^\sigma = r \cdot t_j$ for all $v_j \in V_F$, and $z^\sigma = r \cdot \pi$ ticks. Thus $\mathbf{y}^\sigma = (\mathbf{y}^\sigma - r \cdot \mathbf{t}) + r \cdot \mathbf{t}$ is the required solution with $(\mathbf{y}^\sigma - r \cdot \mathbf{t}) \in Y$.

The existence of a minimal solution immediately follows from the fact that in this case $\mathrm{rank}(\mathbf{\Gamma}) = |V| - 1$ according to [24, Corollary of Lemma 2]. □

Theorem 1 can be used to establish an explicit link between consistent executions and repetitions vectors.

Corollary 1 *Let $\mathcal{P}$ be a consistent polygraph with repetition vector $\mathbf{x} = \mathbf{y} + r \cdot \mathbf{t}$ for some $\mathbf{y} \in Y$ and $r \in \mathbb{N}^{>0}$, and let $s = \langle \mathbf{c}, \theta \rangle \in S$ be a state with a valid synchronous state θ. Then there is a consistent execution σ with initial state $\sigma[1] = s$ such that $\mathbf{y}^\sigma = \mathbf{x}$, $\mathbf{a}^\sigma = r \cdot \mathbf{t}$, and $z^\sigma = r \cdot \pi$.*

Proof. In the proof of (ii)⇒(i), for a given repetition vector, we actually constructed a consistent execution from any state $s' = \langle \mathbf{c}, \langle 0, \mathbf{0} \rangle \rangle$ with a trivial synchronous state $\langle 0, \mathbf{0} \rangle$. In the general case $s = \langle \mathbf{c}, \theta \rangle$, the required execution can also start by the firings of all non-timed actors, then it basically rearranges sub-executions in a different order, performing first all firings until the next tick, then all firings and ticks until the end of the period, then the $r-1$ remaining periods, and finally the remaining firings and ticks of the incomplete period put in the beginning. The detailed proof of the general case proceeds similarly and is left to the reader, since Claim 1 does not require a trivial synchronous state. □

Definition 15 (Minimal consistent execution) Let $\mathbf{x}$ be the minimal repetition vector of a consistent polygraph $\mathcal{P}$. A consistent execution σ of $\mathcal{P}$ with $\mathbf{y}^\sigma = \mathbf{x}$ is called a *minimal consistent execution.*

The following result shows that a minimal consistent execution is a consistent execution of minimal length.

Corollary 2 *Let $\mathcal{P}$ be a consistent polygraph with a consistent execution σ and a minimal repetition vector $\mathbf{x}$. Then σ defines a repetition vector $\mathbf{y}^\sigma$ having the form $\mathbf{y}^\sigma = \mathbf{y} + r \cdot \mathbf{t}$ for some $\mathbf{y} \in Y$ and $r \in \mathbb{N}^{>0}$, and σ has $z^\sigma = r \cdot \pi$ ticks. Moreover, for any minimal consistent execution σ', we have $\mathbf{y}^\sigma = p \cdot \mathbf{y}^{\sigma'} = \mathbf{x}$ and $z^\sigma = p \cdot z^{\sigma'}$ for some $p \in \mathbb{N}^{>0}$.*

Proof. The first part follows from the proof of (i)⇒(ii) of Th. 1. The second part follows from the last part of Th. 1 and Def. 15. Notice that σ and σ' are not required to have the same initial state. □

Finally, the following result shows an equivalence between consistent executions.

Corollary 3 *Let $\mathcal{P}$ be a consistent polygraph. If a synchronous execution σ has the same number of ticks and actor firings as a consistent execution σ', i.e. $\mathbf{y}^\sigma = \mathbf{y}^{\sigma'}$ and $z^\sigma = z^{\sigma'}$, then σ is consistent as well.*

Proof. By Corollaries 2 and 1, we can assume that we have a consistent execution σ' starting from the same initial state $s = \sigma[1]$ as σ. Then by Remark 10, σ and σ' have the same final state, so σ is consistent as well. □

4.2 Liveness Property

In Def. 14, a polygraph $\mathcal{P}$ from an initial state s is called live when it has a live (i.e. consistent and non-blocking) execution starting at s. Since by definition a live execution is consistent, $\mathcal{P}$ must be consistent in order to be live from s.

One goal of this section is to show a remarkable result: when verified, the liveness of polygraph $\mathcal{P}$ from initial state s guarantees that any valid execution starting from s can be extended indefinitely without deadlock. This stronger property better justifies the term *$\mathcal{P}$ is live from s.*

In a way similar to [24, Th. 3], we will to show that if there is a live execution, then any valid execution starting from s can be extended to a live execution. We also show that there is a minimal consistent execution that is live. In other words, the modeled system can be executed indefinitely in finite memory without blocking when its initialization corresponds to s.

For example, we have seen earlier that polygraph $\mathcal{P}^{\dagger}$ from Fig. 5 is consistent. Considering the initial state s^1 of the execution shown in Fig. 7, the only possible valid extension from s^1 is the firing of v_1, and there is no valid extension after that. Indeed, it is not possible to fire v_1 a second time without another tick, it is not possible to tick without a firing of v_3, which is not possible without firing v_2, which is not possible without firing v_1. There is no live execution from that initial state s^1.

On the other hand, the initial channel state $\mathbf{c}^{\ddagger}$ from Fig. 5 allows us to build a live execution, as the one from Fig. 6 up to state s^{17}, with $s^{17} = s^1$. Repeating that execution once more (or continuing with any of the executions shown in Fig. 8) leads to s^{33}, again with $s^{33} = s^1$, and so on.

In a new iteration, the order of transitions can be modified, but as long as the execution remains valid and the numbers of *fire* and *tick* applications remain the same, it leads to the same state, and results in a live execution.

Theorem 2 (Liveness theorem) *Let $\mathcal{P}$ be a consistent polygraph with minimal repetition vector $\mathbf{x}$, and $s \in S$ a valid state of $\mathcal{P}$. The following statements are equivalent:*

(i) there is a non-blocking execution σ such that $\sigma[1] = s$ and σ is a minimal consistent execution;
(ii) $\mathcal{P}$ is live from s;
(iii) any valid execution σ with $\sigma[1] = s$ can be extended to a live execution;
(iv) any valid execution σ such that $\sigma[1] = s$, $\mathbf{y}^{\sigma} \leqq \mathbf{x}$ and $z^{\sigma} \leqslant z$, where z denotes the number of ticks in a minimal consistent execution, can be extended to a live execution that is a minimal consistent execution.

Proof. It will be convenient first to show the following fact.

Lemma 1 *Let σ^1 and σ be two valid executions from initial state s such that $\mathbf{y}^{\sigma^1} \leqq \mathbf{y}^{\sigma}$ and $z^{\sigma^1} \leqslant z^{\sigma}$. Assume that $|\sigma^1| < |\sigma|$, i.e. σ^1 is strictly shorter than σ. Then σ^1 can be extended to a valid execution σ^2 such that $|\sigma^1| + 1 = |\sigma^2|$, $\mathbf{y}^{\sigma^2} \leqq \mathbf{y}^{\sigma}$ and $z^{\sigma^2} \leqslant z^{\sigma}$.*

We show that σ^1 can be extended by one step. Indeed, consider the set of prefixes σ' of σ such that $\mathbf{y}^{\sigma'} \leqq \mathbf{y}^{\sigma^1}$ and $z^{\sigma'} \leqslant z^{\sigma^1}$, in other words, such that all applications of *tick* or *fire* in σ' also occur in σ^1. Thus we have $|\sigma'| \leqslant |\sigma^1|$. This set is non-empty as it contains at least the trivial execution with only the initial state s. Let σ'' be the maximal (i.e. the longest) of such prefixes σ', and suppose its length is $|\sigma''| = n$.

Since $|\sigma''| \leqslant |\sigma^1|$ and $|\sigma^1| < |\sigma|$, we deduce $|\sigma''| < |\sigma|$. We extend the execution σ^1 by the same operation (a tick or a firing of an actor) as the operation immediately following the prefix σ'' in σ, i.e. between states $\sigma[n]$ and $\sigma[n+1]$. Let σ^2 be the resulting execution, we have $|\sigma^2| = |\sigma^1| + 1$. Since the added operation was already present in σ, we have $\mathbf{y}^{\sigma^2} \leqq \mathbf{y}^{\sigma}$ and $z^{\sigma^2} \leqslant z^{\sigma}$.

It can be proved from the validity of σ^1 and σ and the construction of σ'' that the extended execution σ^2 is indeed valid, that is, synchronous and non-blocking, since the preconditions for that are necessarily satisfied. We consider two cases.

Consider first the case when $\sigma[n+1] = \text{tick}(\sigma[n])$. As $z^{\sigma''} \leqslant z^{\sigma^1}$ and $z^{\sigma''} + 1 > z^{\sigma^1}$ by maximality of σ'', the same number of ticks occur in σ^1 and σ''. Let us explain the main steps of the argument. As σ^1 and σ are synchronous, by Remark 10, for any $1 \leqslant i \leqslant z^{\sigma''}$, by the i^{th} tick in σ^1 and σ'' each timed actor has fired the same number of times. As σ is synchronous, Condition (ii) of Def. 10 is verified for that tick just after the prefix σ'' in σ. Since all firings of timed actors that occur in σ'' also occur in σ^1 (as $\mathbf{y}^{\sigma''} \leqq \mathbf{y}^{\sigma^1}$ by construction of σ''), Condition (ii) of Def. 10 for σ^2 to be synchronous is necessarily verified at the end of σ^1 as well. Thus σ^2 constructed by adding a tick at the end of σ^1 is synchronous. In this case, no additional condition for σ^2 to be non-blocking is required, cf. Def. 11.

If $\sigma[n+1] = \text{fire}(v_j, \sigma[n])$ for some actor v_j, since all ticks and firings of actors occurring in σ'' also occur in σ^1 (by construction of σ''), we can show that the conditions for the extended execution σ^2 to be both synchronous and non-blocking are verified. The detailed proof relies on the definitions and is left to the reader.

To prove the theorem, we will show the implications: (i) $\Rightarrow$ (ii) $\Rightarrow$ (iii) $\Rightarrow$ (ii) $\Rightarrow$ (i), and (i) $\Rightarrow$ (iv) $\Rightarrow$ (ii). The implication (i)$\Rightarrow$(ii) directly follows from Def. 14.

Let us show that (ii)$\Rightarrow$(iii). Assume we have a live execution σ from initial state s, and a valid execution σ^1 from s that we need to extend to a live execution. In other words, we need to extend σ^1 to a valid execution coming back to state s.

Notice first that without loss of generality, we can assume that σ is long enough to satisfy $\mathbf{y}^{\sigma^1} \leqq \mathbf{y}^{\sigma}$ and $z^{\sigma^1} \leqslant z^{\sigma}$. Indeed, by Remark 11, all elements in $\mathbf{y}^{\sigma}$ are strictly positive, and we saw in the proof of (ii)$\Rightarrow$(iii) in Th. 1 that $z^{\sigma} > 0$ as well. By repeating live execution σ a sufficient number of times, we can obtain a longer live execution satisfying both inequalities.

If $\mathbf{y}^{\sigma^1} = \mathbf{y}^{\sigma}$ and $z^{\sigma^1} = z^{\sigma}$, then σ^1 is consistent by Cor. 3, and thus live by Def. 14. Otherwise, we can apply Lemma 1 for σ^1 and σ to obtain a longer valid execution σ^2 such that $|\sigma^1| < |\sigma^2| \leqslant |\sigma|$, $\mathbf{y}^{\sigma^2} \leqq \mathbf{y}^{\sigma}$ and $z^{\sigma^2} \leqslant z^{\sigma}$. We can iterate this procedure by progressively extending σ^1 until we obtain a valid execution σ^n such that $|\sigma^n| = |\sigma|$, $\mathbf{y}^{\sigma^n} \leqq \mathbf{y}^{\sigma}$ and $z^{\sigma^n} \leqslant z^{\sigma}$. Therefore, $\mathbf{y}^{\sigma^n} = \mathbf{y}^{\sigma}$ and $z^{\sigma^n} = z^{\sigma}$. Using Cor. 3, we deduce that σ^n is a live execution extending σ^1.

To prove the implication (iii) ⇒ (ii), we apply (iii) to extend the trivial valid execution σ with only the initial state s to obtain a live execution σ′ from s.

Let us prove the implication (ii) ⇒ (i). By (ii), there is a live execution σ′ from s. Let $\mathbf{x}$ be the minimal repetition vector of $\mathcal{P}$. Our goal is to construct a minimal consistent execution σ from s that is also non-blocking. By Cor. 2, such a minimal consistent execution σ will satisfy $\mathbf{y}^{\sigma'} = p \cdot \mathbf{y}^{\sigma} = p \cdot \mathbf{x}$ and $z^{\sigma'} = p \cdot z^{\sigma}$ for some $p \in \mathbb{N}^{>0}$. The number p is thus uniquely defined by the equality $\mathbf{y}^{\sigma'} = p \cdot \mathbf{x}$.

Let the sequence of operations of σ be constructed from the sequence of operations of σ′ by keeping the first $y_j^{\sigma'}/p$ firings of each actor v_j and the first $z^{\sigma'}/p$ ticks in σ′, in their order, and by erasing all other operations. We have to show that σ is synchronous and non-blocking. We give only the main steps of the proof, which mostly uses the same routine verifications as previously.

Consider the longest prefix σ^0 of σ that is both synchronous and non-blocking, and assume $n = |\sigma^0| < |\sigma|$. Let us show that the operation between $\sigma[n]$ and $\sigma[n+1]$ cannot be a tick: it would mean that a longer prefix σ^1—with this additional tick at the end of prefix σ^0—would not be synchronous and non-blocking. Indeed, σ already contains all firings of timed actors that are present in σ′ and necessary for σ^1 to be synchronous. By Def. 11, Condition (ii), an additional tick can never prevent an execution from being non-blocking.

Assume that the operation between $\sigma[n]$ and $\sigma[n+1]$ is a firing of an actor v_k, i.e. $\sigma[n+1] = \text{fire}(v_k, \sigma[n])$. If a longer prefix σ^1 obtained by adding such a firing at the end of prefix σ^0 is blocking, it means that there is an insufficient channel state for some channel $e_i = \langle v_j, v_k\rangle \in E$ in state $\sigma[n]$. This can only occur if some firing of the producer v_j was present in σ′ but erased in σ before the state $\sigma[n]$, and as it was erased, there are already $y_j^{\sigma'}/p$ firings of v_j in σ before the state $\sigma[n]$. Hence, after the $y_k^{\sigma'}/p$ firings of the consumer v_k, the resulting channel state for e_i becomes even smaller, and remains negative. Therefore, since the initial channel state was non-negative, the global effect of σ on channel e_i is negative: $(y_j^{\sigma'}/p)\gamma_{ij} + (y_k^{\sigma'}/p)\gamma_{ik} < 0$. That is contradictory with the constraint $0 = y_j^{\sigma'}\gamma_{ij} + y_k^{\sigma'}\gamma_{ik}$ that follows from the fact that σ′ returns to the initial channel state for this channel (cf. Remark 7). Similarly, we can show that adding a firing of v_k in σ^0 cannot prevent it from being synchronous.

We deduce by contradiction that σ^0 must be equal to σ. Therefore, σ is synchronous and non-blocking. By Corollary 3, the resulting state of σ is s. We deduce by construction of σ that σ is a minimal consistent execution from s that is also non-blocking.

Let us now prove the implication (i) ⇒ (iv). Let σ be a live execution from initial state s that is a minimal consistent execution (thus, by Cor. 2, $\mathbf{y}^{\sigma} = \mathbf{x}$ and $z^{\sigma} = z$). Assume that we have a valid execution σ^1 from s with additional conditions $\mathbf{y}^{\sigma^1} \leqq \mathbf{x}$ and $z^{\sigma^1} \leqslant z$, that we need to extend to a live execution that is a minimal consistent execution. If the additional conditions are both equalities, by Corollary 3, σ^1 is already a minimal consistent execution. Otherwise we can reason like in the proof of (ii)⇒(iii) and iteratively apply Lemma 1 until obtaining the required execution (by Corollary 3).

Finally, the implication (iv) ⇒ (ii) is proved like the implication (iii) ⇒ (ii) above by extending the trivial valid execution σ with only the initial state s to obtain a live execution σ′ from s. □

5 Liveness Checking

Thanks to Theorem 2, we can provide an algorithm to automatically verify whether a given consistent polygraph $\mathcal{P}$ with a given valid state s^1 is live from s^1. If so, the algorithm builds a live execution that is a minimal consistent execution of $\mathcal{P}$. It extends to PolyGraph the PASS algorithm for SDF graphs [24].

Although the theorem is not constructive, it ensures that if $\mathcal{P}$ is live, progressively extending a valid execution (starting from the initial state s^1) by ticks and firings chosen *in an arbitrary order* provides the required execution. We must just take care to do at most as many firings and ticks as in a minimal consistent execution (cf. Th. 2(iv)). Of course, if $\mathcal{P}$ is not live, it will be impossible to finish the extension: it will block at some stage.

Algorithm 1 proposes one possible extension strategy, that we could call "ticks go first". On line 1, it initializes the current state s (i.e. the last state of σ) to the initial state s^1, initializes σ to that initial state, and introduces notation for the components of s. Next, it executes ticks as long as possible (in a loop on lines

Algorithm 1: Liveness test.

```
Data: a consistent polygraph $\mathcal{P}$, $\mathbf{x}$ its minimal repetition vector, $z$ the number of ticks in a minimal consistent execution, $s^1$ a valid state of $\mathcal{P}$.
Result: true if $\mathcal{P}$ is live from $s^1$, false otherwise.
1  $s \leftarrow s^1$ ;   $\sigma \leftarrow s$ ;   $\langle \mathbf{c}, \langle \tau, \mathbf{a} \rangle \rangle \leftarrow s$ ;
2  // Extend $\sigma$ by ticks as long as possible
3  while $\mathbf{a} = \mathbf{t}^\tau \wedge z^\sigma < z$ do
4      $s \leftarrow \text{tick}(s)$ ;   $\sigma \leftarrow \sigma \cdot s$ ;   $\langle \mathbf{c}, \langle \tau, \mathbf{a} \rangle \rangle \leftarrow s$ ;
5  end
6  Allowed$\leftarrow (V \backslash V_F) \cup \{v_j \in V_F \mid t^\tau_j = 1 \wedge a_j = 0\}$ ;
7  Enabled$\leftarrow \{v_j \in V \mid \forall e_i \in \text{in}(v_j), c_i \geqslant \gamma_{ij}\}$ ;
8  Waiting$\leftarrow \{v_j \in V \mid y^\sigma_j < x_j\}$ ;
9  // Extend $\sigma$ by a firing and ticks in a loop
10 while (Enabled $\cap$ Allowed $\cap$ Waiting) $\neq \varnothing$ do
11     // Fire an actor that can fire
12     Choose $v_k \in$ (Enabled $\cap$ Allowed $\cap$ Waiting) ;
13     $s \leftarrow \text{fire}(v_k, s)$ ;   $\sigma \leftarrow \sigma \cdot s$ ;   $\langle \mathbf{c}, \langle \tau, \mathbf{a} \rangle \rangle \leftarrow s$ ;
14     // Do ticks as long as possible
15     while $\mathbf{a} = \mathbf{t}^\tau \wedge z^\sigma < z$ do
16         $s \leftarrow \text{tick}(s)$ ;   $\sigma \leftarrow \sigma \cdot s$ ;   $\langle \mathbf{c}, \langle \tau, \mathbf{a} \rangle \rangle \leftarrow s$ ;
17     end
18     Allowed$\leftarrow (V \backslash V_F) \cup \{v_j \in V_F \mid t^\tau_j = 1 \wedge a_j = 0\}$ ;
19     Enabled$\leftarrow \{v_j \in V \mid \forall e_i \in \text{in}(v_j), c_i \geqslant \gamma_{ij}\}$ ;
20     Waiting$\leftarrow \{v_j \in V \mid y^\sigma_j < x_j\}$ ;
21 end
22 return $\mathbf{y}^\sigma = \mathbf{x} \wedge z^\sigma = z$;
```

3–5) to obtain a valid execution, but takes care to do at most z ticks. Its validity follows from the loop condition. Then it computes the sets of actors that are allowed to fire in the current state from the point of view of synchronous constraints (Allowed), that have enough input tokens to fire (Enabled), and finally that have not yet fired enough (Waiting). They are used in the main loop (lines 10–21). As long as their intersection has an actor v_k (cf. lines 10, 12), we can extend σ to a longer valid execution by a firing of v_k (cf. line 13). Its validity follows from the definition of sets Allowed and Enabled. The definition of Waiting ensures that σ has at most as many firings of each actor as in $\mathbf{x}$. Then again, the algorithm executes ticks as long as possible (in a loop on lines 15–17). At the end of the iteration, the three sets are recomputed (lines 18–20). In any step, σ has at most as many firings and ticks as in a minimal consistent execution.

The actor (arbitrarily) chosen on line 12 can be the actor in the intersection with the smallest index. This *ticks-go-first* and *smallest-index-first* strategy applied to the polygraph $\mathcal{P}^\dagger$ from Fig. 5 will produce the execution in Fig. 8 passing through the three top nodes. (The execution in Fig. 8 passing through the lowest node would be obtained by another extension strategy: *firings-go-first* and *smallest-index-first.*)

Proposition 1 *Let $\mathcal{P}$ be a consistent polygraph with a valid initial state s^1. Algorithm 1 terminates. It is a sound and complete algorithm to check if $\mathcal{P}$ is live from s^1. If it is live, the execution σ constructed by the algorithm is a live execution from s^1 that is a minimal consistent execution of $\mathcal{P}$.*

Proof. Each iteration of each loop extends σ by at least one transition. The algorithm terminates since the length $|\sigma| = 1 + \sum_j y^\sigma_j + z^\sigma$ increases at each extension, and cannot exceed the length of a minimal consistent execution.

Assume the true verdict is returned (by line 22), that is, a valid execution σ with the same number of ticks and firings as in a minimal consistent execution was constructed. Then σ is consistent by Cor. 3. Thus σ is a live execution that is a minimal consistent execution. Therefore, $\mathcal{P}$ is live from s^1.

To show completeness, assume $\mathcal{P}$ is live from s^1. Then by Th. 2(iv), the extension of a valid execution σ is always possible by a tick or by a firing. After any extension by a new transition (at line 4 or 13 or 16), the algorithm always tries first to add a tick (at line 4 or 16), and when it is not possible any more, a firing (at line 13). Thus the algorithm cannot return false, as it would mean that at some stage it could execute neither a tick nor a firing, contrary to Th. 2(iv). The details are left to the reader. □

Algorithm 1 is an extension to PolyGraph of the PASS algorithm for SDF graphs [24], taking into account the tick event and the synchronous property of polygraph executions. It can be optimized by recomputing the sets on lines 18–20 using their previous values and pre-computed *significant* ticks at which some timed actors are expected to fire, and updating only data for the impacted channels or actors. For simplicity we presented here a straightforward version.

Remark 12 It is important to note that in a consistent polygraph $\mathcal{P}$, there is an initial channel state $\mathbf{c}$ that guarantees liveness from any valid state $s = \langle \mathbf{c}, \langle \tau, \mathbf{a} \rangle \rangle$. This channel state $\mathbf{c}$ is defined by $c_i = x_k \gamma_{ik}$ for any channel $e_i = \langle v_j, v_k \rangle$. Indeed, in that case, the consumer v_k has enough tokens on e_i for all its x_k firings along the minimal consistent execution, even in the worst case, when all firings of the producer v_j happen after them. Thus, Enabled $= V$ at any step of the algorithm, so the timed actors can always fire as expected on the corresponding tick without blocking. Of course, this over-feeded initial state $\mathbf{c}$ may introduce too much latency in the system to be useful in practice. We use it in polygraphs we generate to test our implementation of liveness checking algorithm, as explained in Section 7. □

Complexity. The main difference between Algorithm 1 and the PASS algorithm of SDF is the handling of ticks. As mentioned above, by precomputing significant ticks, the loop (lines 3–5, 15–17) executing the next ticks where no timed actor is expected to fire can be optimized and turned into a constant-time operation, executing the following non-significant ticks at once, along with the preceding significant one.

As for the PASS algorithm, the main complexity factor for Algorithm 1 is thus the total number of firings in a minimal consistent execution of the input polygraph $\mathcal{P}$. It determines the number of iterations of the main loop. This number is given by the sum of elements $\sum_j x_j$ of the minimal repetition vector $\mathbf{x}$ of $\mathcal{P}$. Each iteration firing an actor v_j needs to update the status of its $|\mathrm{in}(v_j) \cup \mathrm{out}(v_j)|$ incident channels, that leads to $O(|V|)$ operations per iteration, taking into account the worst case of a complete system graph.

For a topology matrix $\mathbf{\Gamma}$ with integral elements (in particular, for SDF) the bound on the number of firings can be derived from the maximum M_2 of the absolute values of the minors of order $|V| - 1$ in the matrix $\mathbf{\Gamma}$: we can deduce the bound $\sum_j x_j \leqslant |V| \cdot M_2$ from [8, Th. 1]. The upper bound of the overall complexity is thus $O(|V|^2 \cdot M_2)$. Since a minor (and thus M_2) is exponential in the size $|V|$, it shows that, without any restriction on the elements of the matrix $\mathbf{\Gamma}$, this bound is in general exponential.

Indeed, the time complexity of the PASS algorithm is exponential in the worst case, which motivated the investigation of alternative methods. One example is checking the initial number of tokens occupying the channels in cycles of the data flow graph [6], but as demonstrated in Sect. 2 this is not applicable to polygraphs, and anyhow suffers from an exponential worst-case for a complete graph. Other approaches simplify the problem and check a sufficient condition in polynomial time [26].

PolyGraph inherits this well-known issue for SDF graphs and similar models. However, in practice, SDF is widely used because the number of firings in real life models remains reasonable compared to $|V|$, making the liveness check a rather fast operation. This practical observation has been confirmed for PolyGraph by our experiments and industrial use cases, as we show below.

6 Methodology and Tool Support

In practice, the engineers designing a system do not necessarily need to know all theoretical aspects of the underlying formal model of computation. Based on a realistic use case, we illustrate in this section the main steps in a modeling methodology using PolyGraph, that do not require a deep theoretical knowledge of the formalism.

The considered use case is an Advanced Driver Assistance System (ADAS). Several sensors are used to determine the vehicle speed and perceive the environment, in order to render information on a cockpit display and regulate speed.

The main steps in the methodology include an identification of the functions and their dependencies, the conversion of this informal specification to a polygraph, and the refinement of constraints.

6.1 Functional Specification

We describe the architecture of the ADAS use case in a way inspired by a state-of-the-art methodology for service-oriented automotive systems [23].

Functions. In the ADAS use case, three sensors are used. An odometer (ODM) measures the current speed, a lidar (LDR) perceives nearby obstacles, and a stereo camera films the road ahead. The stereo streams are produced independently (LCM for left and RCM for right).

The data produced by these functions is used by six perception kernels, to transform it into more abstract objects usable by decision and rendering processes (cf. columns 2–3 in Fig. 9). These six perception functions detect the following elements in the environment: near-field obstacles (OBD), traffic signs (TSD), road mask (RMD), traffic lanes (TLD), pedestrians (PDD), and depth map (DMD), where the last letter “D” stands for detection.

The object descriptors produced by the perception kernels are then used by fusion kernels to have refined information based on context. Two such kernels are present in our use case. The results of PDD, RMD, and DMD are combined to perform an advanced pedestrian detection (APD), to identify pedestrians that are both close to the vehicle and standing on the road. Data from ODM combined with the results of TSD and OBD are used to perform speed control (SPC), to determine if the current speed is below the speed limit and whether there is a risk of collision with a nearby obstacle.

Some software components interact with the physical world. An emergency braking system (EBS) may be triggered by information received from speed control SPC. Finally, most of the available results are analyzed and rendered on an information display (IFD).

Interfaces. In service-oriented architectures, software components exchange data through typed *input/output interfaces.* An output interface declares the publication of

Type	Producers	Consumers	Comment
Left Frame	LCM	TSD, PDD, RMD, DMD, TLD	Camera frame (left stream).
Right Frame	RCM	DMD	Camera frame (right stream for stereo vision).
Speed	ODM,TSD	SPC	Speed in km/h.
Grid	LDR	OBD	Occupancy grid from lidar samples.
Obstacle	OBD	SPC	Range to closest relevant obstacle.
Brake	SPC	EBS	Boolean requiring emergency braking when set.
InfoSpeed	SPC	IFD	Current speed, speed limit, and braking status.
Mask	PDD,RMD,DMD	APD	Descriptors to draw
Mask	TLD,APD	IFD	on a camera frame.

Fig. 9: Types used in the ADAS use case, with a listing of the producing and consuming software components per type.

samples of a certain data type, and an input interface declares the subscription to the stream of typed samples produced by an output interface. For the sake of simplicity, we define very coarse-grained sample types to specify the interfaces between functions in Fig. 9. This way, each software component requires one sample from its input interfaces in order to compute, and produces one sample per computation on its output interfaces.

Timing constraints. Data fusion requires the input data from different sources to be temporally correlated. The corresponding timing constraints are generally derived from functional requirements rather than sensor specifications. For example, regarding the acquisition of speed by the odometer ODM, the system designers want to determine a sampling frequency suitable to observe speed variations with a precision that suits the needs of the system.

In the domain of data fusion, these timing constraints are often expressed as frequencies, in Hertz (Hz) or frames per second (fps). In our use case these constraints were expressed in this way by the engineers developing the system according to their expertise.

The sensing functions are constrained to produce a regular stream of data: odometer ODM at 10Hz, lidar LDR at 30Hz, and both cameras LCM and RCM at 10fps. The functions having an effect on the physical world are also constrained. For the IFD display, its constraint is to render the gathered information on the screen at 10fps. The minimal inter-arrival time for emergency braking commands to EBS is 100ms, as such it polls for input commands at 1/100ms = 10Hz.

The computation kernels are indirectly constrained by these frequencies. Most of the image processing kernels are applied to each frame produced by LCM, at 10fps. However, due to the required computational load, the road mask is expected to be produced (by RMD) every two frames of LCM (hence, at 5fps), and the depth map (by DMD) every five frames of LCM and RCM (2fps). The APD fusion kernel will work on every descriptor produced by PDD, at 10fps, and use the information produced by RMD and DMD when it is available. The occupancy grid produced by OBD is refreshed from samples produced by LDR at 30Hz. The speed information and emergency braking commands are produced by SPC at 10Hz, and it needs only the most recent grid from OBD.

In addition to these frequency constraints, this kind of system also has the notion of end-to-end latency. It is important to the system designers to determine the acceptable delay to have a response to an input of the system. Again, by experience, the end-to-end latencies are specified from sensor to actuator, in standard time units like milliseconds (ms).

As such, two end-to-end latencies are specified for our ADAS use case: the information for frames acquired by LCM and RCM must be rendered on screen by IFD at most 50ms after acquisition, and an emergency braking command resulting from data acquired by odometer ODM and lidar LDR must be taken into account by EBS at most 20ms after acquisition.

6.2 Polygraph Modeling

The above specification elements can be translated directly into a polygraph, whose graphical representation is given in Fig. 10. It shows the information required by a tool to build the formal representation of the polygraph. The steps to obtain it are the following.

System graph. The system graph in Fig. 10 is actually a direct translation of the specification. Each function is an actor, and each communication dependency is a channel. The only exception is the stream from LCM, which is used by several functions. It is represented by

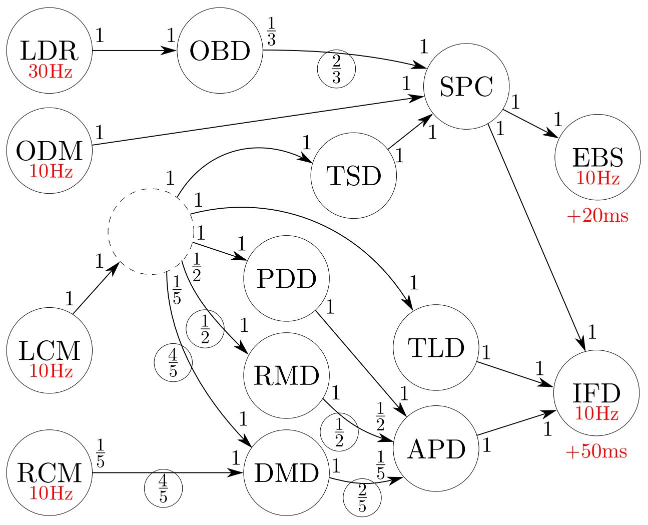


Fig. 10: Graphical representation of a polygraph modeling the ADAS use case.

a dashed vertex in the graph, modeling a function in charge of distributing the data to different consumers. Data flow programming languages [1,20] make use of such actors to identify a data sharing pattern in the graph, and enable compile time optimizations [29].

Frequencies. Frequency labels are added to the actors modeling sensor and actuator functions that lie at the interface with the physical world. These timed actors receive a label that corresponds exactly to the informal specification, and they will have a globally synchronous periodic behavior.

All other functions are left without a frequency label, since they are computation kernels reacting to input data. This allows a less constrained asynchronous behavior for their firing. As mentioned in Sect. 2, this will allow performance analysis to be less pessimistic by enabling additional optimizations that would otherwise not be possible if they were constrained to a globally synchronous periodic behavior.

Finally, the expressed end-to-end latencies are captured by phases added to EBS and IFD. The phases are expressed exactly as in the informal specification. Since none of the sensing function actors receive a phase, they all fire at the first tick of the global clock. The phases expressed for EBS and IFD are thus duration offsets from that reference starting date.

Rates. As explained in Sect. 2.1, the granularity of data exchange on each channel is captured by the notion of token. In our example, for every channel, a token is an instance of the type specified for the connected interfaces in Fig. 9. A good starting point is then to set all rates on all channel endpoints according to the communication requirements of the connected functions. In our example, that means initially setting all rates to 1.

To fit the specification, some adjustments are required though. The actors RMD and DMD model computation kernels and as such do not have a frequency constraint. However, a frequency was expressed in the specification, relative to that of the stereo camera. The rational rates on channels leading to RMD and DMD in Fig. 10 directly follow from that informal specification. Indeed, RMD will compute on 1 frame out of 2 from the left stream, and DMD will compute on 1 out of 5 frames from both streams. Since APD works on every frame mask provided by PDD at 10fps, it will have a frame mask from RMD every other firing and one from DMD every five firings. The rate of $1/3$ on the output channel of OBD follows from a similar reasoning.

Initial marking. RMD and DMD are expected (based on the expertise of the system designers) to take longer to produce results, and this may cause undesirable latency in the system since APD depends on their result to provide the advanced pedestrian detection mask to IFD. The rational rates on channels connected to RMD and DMD specify a resampling of the data streams coping with their reduced throughput, but they also leave a degree of liberty on the synchronization of communications to absorb their latency, thanks to a rational initial marking of the channels.

Without an initial marking, actors like RCM with a rational rate $1/q$ on an output channel produce a token only after q firings. Indeed, the successive firings add the rate to an initial channel state 0, and the number of tokens occupying the channel after $q-1$ firings is $\lfloor (q-1)/q \rfloor = 0$ (cf. Def. 3 and 7). The integer part of the channel state increases only after the q^{th} firing: $\lfloor q/q \rfloor = 1$. Hence, increasing the fractional part of an initial channel state to k/q makes the production of a token occur k firings earlier (this was illustrated earlier in Figures 2 and 3).

In our example, an earlier firing of RMD, DMD, and SPC is enabled by the initial marking (shown by circled numbers) of their input channels, which come from the corresponding sensing functions (cf. Fig. 10). The maximum possible fractional part is used to have the earliest release, and RMD and DMD will start processing on the first frame produced by LCM and RCM.

Following the same logic, without an initial marking, actors like APD with a rational rate $1/q$ on an input channel consume a token on their first firing, and then nothing up to the $(q+1)^{\text{th}}$ firing. Indeed, in a valid execution, such an actor fires only when the channel state is greater than $1/q$. With an initial channel

Algorithm 2: Conversion of a rational rate to a cyclo-static rate sequence

```
Data: a rate $0 \neq \gamma = \frac{p}{q} \in \mathbb{Q}$ with $q > 0$, an initial
      channel state $c \in \mathbb{Q}$.
Result: $n_1 \cdots n_q \in \mathbb{N}^+$, with $n_i$ the number of tokens
      produced (if $\gamma > 0$) or consumed (if $\gamma < 0$) by the $i^{\text{th}}$
      firing.
1 $r \leftarrow c - \lfloor c \rfloor$ ;
2 for $i = 1$ to $q$ do
3     if $\gamma > 0$ then $n_i \leftarrow \lfloor i\gamma + r \rfloor - \lfloor (i-1)\gamma + r \rfloor$ ;
4     else $n_i \leftarrow \lceil i|\gamma| - r \rceil - \lceil (i-1)|\gamma| - r \rceil$ ;
5 end
6 return $n_1 \cdots n_q$
```

state of 0, when such an actor fires in this condition, the channel state has to be an integer n since by Def. 2 the producer's rate is an integer. Removing $1/q$ from that state necessarily decreases the integer part of the channel state by 1 since $\lfloor n - 1/q \rfloor = n - 1$. This leaves a fractional part of $(q-1)/q$, and the following $q-1$ firings deplete that fractional part down to 0. Only the $(q+1)^{\text{th}}$ firing finally decreases the integer part by one again. Hence, increasing the fractional part of an initial channel state to k/q has the opposite effect when the rational rate is on the input endpoint, and makes the consumption of the first token occur k firings later.

Therefore, in our example (cf. Fig. 10), the initial marking of the input channels of APD delays the requirement to obtain data from RMD and DMD to the second and third firings respectively. Using the maximum possible fractional part would delay the input requirement to the latest possible instant. However it is not used in this case, to ensure a suitable temporal correlation of the tokens consumed by firings of the fusion APD. The time elapsed between two frames on each of LCM and RCM streams is 100ms. It can be checked that with the chosen initial markings, APD always consumes data derived from samples sensed at most 200ms apart (this could be an additional requirement of the system).

The computation of numbers of tokens produced or consumed by successive firings can be easily generalized to any rational rate p/q and initial state c. Algorithm 2 returns the sequence $n_1 \cdots n_q \in \mathbb{N}^+$ of numbers of tokens produced or consumed by q successive firings. That sequence is equivalent to a cyclo-static rate as explained in Sect. 2. The algorithm uses the fact [13, Prop.1] that after the first i firings, a producer produces (resp., a consumer consumes) $\lfloor i\gamma + r \rfloor$ tokens (resp, $\lceil i|\gamma| - r \rceil$ tokens) in total, where r is the fractional part of c. Taking the difference between these cumulative numbers after i firings and after $(i-1)$ firings gives the desired number for the i^{th} firing. (The implementation of Algorithm 2 can obviously be optimized to avoid redundant computations.) An automatic tool can thus help to easily switch between the rational specification and that sequence, making it clearer to the system designers which firing produces or consumes tokens.

This concludes the explanation of how the specification elements are converted into the representation of Fig. 10. It can be automatically converted to a polygraph by automated tools, which can also further assist the system designers in checking its liveness.

6.3 Assisted Property Checking

The graphical representation of Fig. 10 provides sufficient information to automatically build the underlying polygraph, and assist the system designers in the verification of their model.

Conversion to a polygraph. The construction of actor set V and channel set E is trivial, as well as verifying that they form a system graph $G = (V, E)$ according to Def. 1. The set of timed actors V_F contains all actors from V for which a frequency label was specified. Following the conditions of Def. 2, defining a topology matrix for G is also straightforward.

An automatic tool can then be used to deduce synchronous constraints. Given the graphical representation, frequency labels and phases should first be converted into compatible units, say, frequencies in kilohertz and phases in milliseconds. Algorithm 3 can then be applied to compute a tuple of synchronous constraints Θ for V_F respecting the conditions of Def. 4.

The algorithm is given in the general case for rational frequences and phases, but its first steps follow the explanations given (for integer frequences and phases) in Sec. 3.3. First it computes a suitable hyperperiod duration (in ms) for the global clock, and normalized frequencies (which belong to $\mathbb{N}^{>0}$) with respect to this hyperperiod. The greatest common divisor is extended from integers to rational numbers by defining $\gcd(\frac{p_1}{q}, \frac{p_2}{q}) = \frac{\gcd(p_1, p_2)}{q}$ for two rational numbers $\frac{p_1}{q}, \frac{p_2}{q} \in \mathbb{Q}$ brought to a common divisor $q > 0$, and can be generalized to a finite number of arguments.

Then we compute the minimal global clock resolution (i.e. its number of ticks) suitable to capture by its ticks all firings without phases (π_f), all phases (π_p) and finally both firings and phases (π). The resulting tick duration $c = h/\pi$ allows us to compute the normalized phases (which belong to $\mathbb{N}^{\geqslant 0}$). The proof that the conditions of Def. 4 are satisfied is left to the reader.

The result of that conversion is a polygraph $\mathcal{P} = \langle G, \mathbf{\Gamma}, \Theta \rangle$, according to Def. 6. We explain in the follow-

Algorithm 3: Synchronous constraints.

```
Data: set of timed actors $V_F$, frequencies
      $f : V_F \to \mathbb{Q}^{>0}$, phases $p : V_F \to \mathbb{Q}^{\geqslant 0}$, in
      compatible units (e.g. kHz and ms), assuming
      $\forall v, p(v) < 1/f(v)$.
Result: synchronous constraints $\Theta$ for $V_F$ (cf. Def.4)
1  //Suitable hyperperiod h (in ms) for global clock:
2  $a \leftarrow \gcd(f(v) | v \in V_F)$;
3  $h \leftarrow 1/a$;
4  //Normalized frequencies w.r.t. hyperperiod h:
5  $\omega : \forall v \in V_F,\ \omega(v) \leftarrow f(v)/a$;
6  //Min. resolution to express all firings (w/o phases)
7  $\pi_f \leftarrow \mathrm{lcm}(\omega(v) | v \in V_F)$;
8  //Min. resolution to express phases as tick offsets:
9  $b \leftarrow \gcd(h, \gcd(p(v) | v \in V_F))$;
10 $\pi_p \leftarrow h/b$;
11 //Min. resolution to express all firings with phases:
12 $\pi \leftarrow \mathrm{lcm}(\pi_f, \pi_p)$;
13 //Normalized phases w.r.t. global clock ticks:
14 $c \leftarrow h/\pi$;
15 $\varphi : \forall v \in V_F,\ \varphi(v) \leftarrow p(v)/c$;
16 return $\Theta = \langle \omega, \pi, \varphi \rangle$
```

ing paragraphs how a tool could benefit from that representation and Theorems 1 and 2 to assist the system designers in correcting errors in their specification, before starting the process of predicting its performance.

Rate and frequency refinement. From $\mathcal{P}$, the vector $\mathbf{t}$ can be inferred as per Remark 3. Thanks to Th. 1, an automatic tool has enough information to check whether $\mathcal{P}$ is consistent, using for example a Cholesky decomposition to solve the corresponding system of equations.

In the case of our example of Fig. 10, the model is consistent. However, its variants with a minor specification error or a mistake in building the input might lead to a negative verdict (for example forgetting to set the rate of 1/3 on the output channel of OBD), helping the system designer to identify the issue and fix the model. Developing advanced mechanisms to infer correct rates and suggest them to the system designers is an interesting future work direction, as discussed in Sect. 9.

Refinement of initial conditions. For the ADAS system depicted in Fig. 10, an initial state can be directly inferred as follows. Regarding the channel state $\mathbf{c}$, for a channel e_i, if the corresponding arc in the graphical view has no initial marking we set $c_i = 0$; otherwise, the available rational marking is used as initial channel state c_i. Checking that channel state $\mathbf{c}$ is valid according to Def. 3 is trivial. We set to zero the global tick τ and tracker vector $\mathbf{a}$, so the inferred initial state is thus $s = \langle \mathbf{c}, \langle 0, \mathbf{0} \rangle \rangle$. Applying Algorithm 1 on the polygraph $\mathcal{P}$ with initial state s returns *true*.

The test may fail, for example, if one forgets to mark the channel between OBD and SPC with 2/3. As for consistency, automated assistance to correct the model in this case should be possible, as discussed in Sect. 9.

7 Implementation and Experiments

DIVERSITY is a customizable model analysis tool based on symbolic execution [16], available in the *Eclipse Formal Modeling Project* [33]. 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 [16,3], Communicating STS [2], and abstractions of hybrid systems [27]. In this work, we use it to analyze polygraphs and check their liveness. All experiments were run on an *Intel core i7-7920HQ @ 3.10 GHz, RAM 32 GB.*

7.1 xLIA State Machines for Polygraph Analysis

The root entity in an xLIA model is a so-called *system*. A system is an executable entity that can be atomic (state-machine), compositional or hierarchical. A polygraph translated to xLIA is a system where the actors are state-machines with input/output ports associated with channel endpoints. 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 11 represents such a state machine for any actor of a polygraph. Transitions are 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 of the actor (i.e. the actor belongs to the Enabled set of Algorithm 1), *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 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. A sequence of actors to fire is associated with each tick of a clock, which corresponds to building the Allowed set in Algorithm 1.

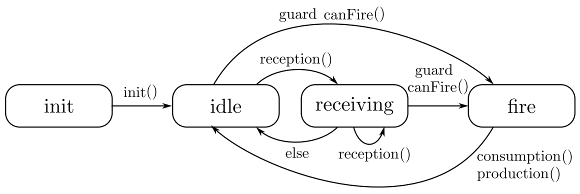


Fig. 11: xLIA state machine pattern for an actor of a polygraph

example	number of (timed) actors	built exec. length	verdict	time
Fusion (a)	(4) 5	23	Yes	8ms
Fusion (b)	(4) 5	14	No	5ms
ADAS (a)	(7) 18	207	Yes	111ms
ADAS (b)	(7) 18	10	No	6ms
MP4-SP (a)	(2) 5	299	Yes	152ms
MP4-SP (b)	(2) 5	130	No	60ms

Fig. 12: Results of experiments, where (a) marks live examples, and (b) marks manual modifications of examples to break the liveness property. The third column indicates the length of the constructed minimal live execution (when found) or of the tentative execution (when a deadlock occurs).

Like Algorithm 1, the current implementation tries to build a live execution that is a minimal consistent execution. When it succeeds, the true verdict along with the constructed execution is returned. As explained in Sect. 5, various generation strategies can be possible. The implemented strategy is optimized and slightly different from Algorithm 1. The significant ticks at which each timed actor is supposed to fire are precomputed. When attempting to fire a timed actor, at most one firing is triggered, but when attempting to fire a non-timed actor, as many firings of the actor as possible are triggered at the same time. That limits the number of necessary iterations and updates of the channel states. Hence, the timed actors can only fire at the expected tick. For any actor, a counter limits its number of firings to its coordinate in the minimal repetition vector (that corresponds to checking if it belongs to the Waiting set in Algorithm 1).

The two subsections below describe two campaigns of experiments. The first one, on a set of small (fragments of) real-life examples is aimed at evaluating the correctness of our tool. The purpose of the second one, on a large set of random polygraph models generated automatically, is to evaluate the performance of the tool.

7.2 Experiments on Real-Life Examples

We have applied our algorithm to different small examples inspired by real-life systems and summarized the results in Fig. 12. The model $Fusion$ (a) is depicted in Fig. 1d) and represents a live polygraph. The model $Fusion$ (b) is the same model, with a shorter phase for the *display* actor, which blocks its execution on the first tick on which it is supposed to fire (this is illustrated earlier in Fig. 3). The model $ADAS$ (a) is a variant of the one detailed illustrated by Fig. 10 and explained in Sect. 6. For the correctly marked model $ADAS$ (a), we find a live execution sequence in 111ms. $ADAS$ (b) is a modification of $ADAS$ (a), obtained by setting to 0 the initial channel states, provoking a deadlock execution which was identified in 6 ms. The example denoted $MP4-SP$ (a) is a translation to PolyGraph of a static version of the classical $MPEG4-SP$ SADF decoder [34] (it was converted to a polygraph to illustrate dynamic reconfigurations in an extension to the formalism, the dynamic polygraph is available in [13]). The polygraph $MP4-SP$ (a) has cycles, and has sufficient initial conditions to be live. The modified version $MP4-SP$ (b) has a shorter phase for an actor, as for the $Fusion$ (b), in order to break the liveness property.

All the results provided by the algorithm were confirmed manually. The results of experiments confirm that the proposed tool correctly verifies liveness on realistic small-scale models. Compared to the results published in [14], our optimized implementation performs 30% better thanks to better engineering of the management of clock ticks, that were evaluated at each step in the previous version.

7.3 Polygraph Generator

We have also developed and used a model generator to automatically generate polygraphs, in order to increase the number of available models and to draw conclusions on the performance of the approach. As explained below, by construction, some generated models are known to be live, and all the others are at least consistent. The generator has some degrees of randomness, and the principles are as follows.

The generator is configured to generate models with a given number of actors, and connect them with a given number of channels. A given percentage of actors are timed and receive a random frequency chosen within a given range. Among these timed actors, a fixed percentage receives a random phase in addition to their frequency, chosen within the range of acceptable phases according to Def. 4. The channels are generated by choosing randomly a pair of actors, and assigning random rates on the channels.

As discussed earlier in Sect. 5, the main complexity factor is the number of firings in minimal consistent executions. This number does not depend on the number of actors or channels (except for its lower bound). As such, generating random models of a fixed size with a random number of firings for the actors would not provide more insights than generating random models of increasing size with a bounded number of firings. With the latter approach though, the measures show the impact of the size of the instance on the execution time. For this reason, one of the goals of the random generator is thus to bound the number of firings in the minimal consistent execution.

The frequencies are chosen randomly from a configurable range. In the benchmark we generated, that range was 10–50Hz, with a step of 10, to have a measure of control on the duration of one period of the global clock. Since the hyperperiod of the system is the least common multiple of the timed actor periods, by choosing these values, in the worst generation scenario we have a global clock period of 100ms.

The generator first builds a connected graph, then generates additional channels in that single connected component. As we focus on liveness detection and since by Theorem 2 consistency is a prerequisite, the generation of channel rates is implemented to guarantee the consistency property of the generated polygraphs.

The condition in Th. 1 defines balance equations, one for each channel $e_i = \langle v_j, v_k \rangle$, relating the expected number of firings x_j, x_k of the connected actors in a consistent execution and their respective rates γ_{ij}, γ_{ik} on e_i. These balance equations are of the form $x_j\gamma_{ij} + x_k\gamma_{ik} = 0$. In addition, for a timed actor v_j, its variable x_j must be a multiple of the number of expected firings in the hyperperiod.

The generator randomly chooses one actor of a channel as a reference and randomly generates a rate for the new connection. Then, relying on the balance equations, it computes a suitable rate for the actor on the other end of the channel. We distinguish the following cases:

- when the connected actors are both timed actors, their variables x_j, x_k are constrained by their respective repetitions over the global clock period, and the generator chooses the rates to satisfy this constraint;
- when the connected actors belong to the same connected component, their variables x_j, x_k are constrained by the existing balance equations in the component, and the generator chooses the rate to preserve the validity of these equations;
- in any other case, the variables x_j, x_k are independent, and the generator chooses the rates randomly, with an objective to bound the values of x_j, x_k.

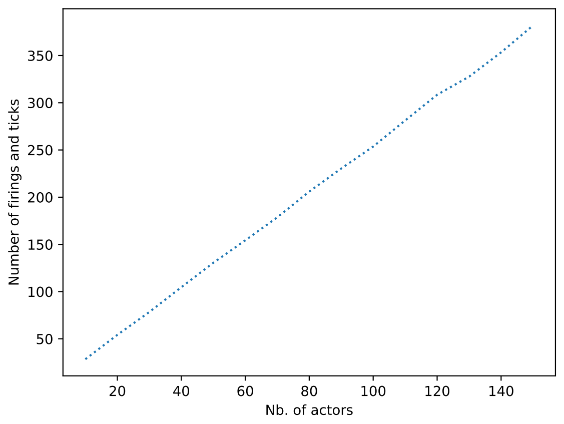


Fig. 13: Average length of the minimal consistent execution explored for the generated models, the linearity of that curve is enforced by our random generator.

Once the configured number of channels is generated, the generator outputs a first version of the model. Without an initial marking, and with phases generated randomly, it is very unlikely that this version is live. In order to produce a live model, a second version is created, this time with all channels marked with enough tokens to complete x_k firings of the consumer. The marking is thus $x_k \cdot \gamma_{ik}$. As mentioned in Remark 12, this guarantees that Algorithm 1 returns true.

7.4 Experiments on Generated Examples

Using the polygraph generation algorithm we described above, we generated 12,000 models. Models are first classified into 9 categories, based on the number of their actors, ranging from 10 to 150 with a step of 10. Each of these categories has the same number of models and contains models generated with varying parameters: the number of channels, the number of timed actors, and the number of timed actors with phases.

As mentioned above, half of the generated models are initialized to force the liveness property. As expected, most of the others were not detected as live by DIVERSITY, and we have 50.25% of live models among the 10,000 generated. For the 6030 live models, the average length of the minimal live execution explored by the algorithm increases linearly with the number of actors in the model, as shown in Fig. 13. This is explained by the guided choice of the rates in

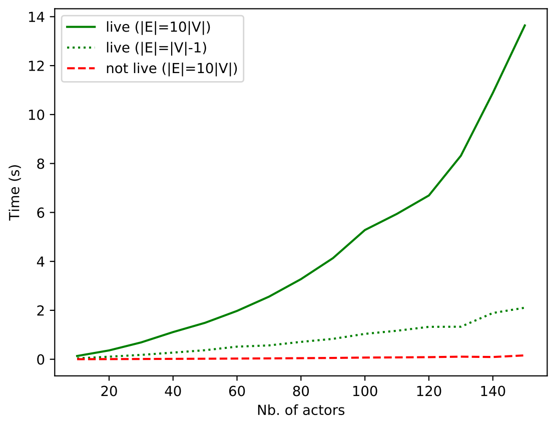


Fig. 14: Average execution time for the verification of the liveness property for the generated models.

our generator. Indeed, as mentioned above, it tries as much as possible to bound the number of firings per actor in the minimal consistent execution.

Fig. 14 shows the average time it took to check whether the liveness property holds for the considered models in each category. In order to show the impact of the number of channels in a tested model, the benchmark is divided between models that have the minimal number of channels to have a connected system graph ($|E| = |V| - 1$), and models such that $|E| = 10|V|$. As discussed in Section5, the average degree of the actors impacts the complexity of the liveness test, and with these categories we have an average degree of respectively ≈ 1 and 10 (note that our ADAS use case has an averge degree of 1.2).

For non-live models, the curve only shows that the detection occurs early in the search for a minimal live execution (almost always before the second tick). For the models that were found to be live, the curves indicate that the analysis remains very fast, with an increasing execution time with respect to the length of the minimal live execution (as explained above), and the number of channels. In the case where the average degree is close to 1 the execution time remains below 2s even for the models with more than 350 firings and ticks in the minimal consistent execution. For the case where the average degree is 10, the execution time remains below 15s for the models with the minimal consistent executions of that same length. These results confirm that the proposed technique can be efficiently applied for real-size models, since models with an average degree of 10 and more than 300 events in the minimal consistent execution are far larger than what can be expected in real-size case studies.

8 Discussion and Related Work

Data flow and real-time. Regarding real-time and SDF graphs, most of the related work focuses on applying real-time scheduling techniques by assuming some degree of periodicity for all actors [22,31]. These approaches do not provide extensions to the formalism to express periods on a subset of actors, integrate them into the semantic of the model and allow verifying properties prior to the schedulability tests.

Some approaches propose extensions to the formalism and consider impact on the decidability of properties. Selva [30] considers different periods, with an extension to SDF adding a single throughput constraint on a channel of a consistent SDF 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. Singh et al. [32] propose an extension that allows the specification of end-to-end latencies from a source actor to a sink actor, the source actor being fired on arrival of a sporadic event. The impact on the properties is limited by several assumptions on the topology matrix.

Recently published research [10] 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 [10], 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.

Regardless of the lower expressiveness and lack of formal properties in these approaches, the results they provide are of interest to derive real-time schedulability tests and scheduling approaches for polygraphs.

Discrete events. The *reactor* model [25] and the model of computation PTIDES [35] combine a real-time semantic for sensors and actuators, and a discrete event semantic for other components like computation kernels. The reactors have an awareness of the real-time through a logical time abstraction. The resulting execution semantic has similarities with PolyGraph, since some reactors are constrained by real-time and others only react to input stimuli.

The main difference is that in the discrete event semantic, components are not forced to produce output events, while in the data flow semantic they are, according to their output rates. This means that reactors are better suited to model sparse systems in which the components execute rarely, while PolyGraph actors are more adapted to dense systems in which the components execute often with a regular behavior.

There are also small differences in the kind of real-time constraints that can be expressed. Both allow the expression of stricly periodic and globally synchronous behavior. While PolyGraph allows the expression of end-to-end latencies from sensor to actuator, individual inputs of reactors can be assigned a deadline for the production of output events in response to that particular input. Reactors can also produce sporadic events with a minimal inter-arrival time.

We thus believe that reactors and PolyGraph actors are complementary concepts. For example, in the ADAS use case detailed in Sect. 6, the relation between actor SPC and EBS would clearly benefit more from a reactor approach than the PolyGraph approach that requires periodic polling. On the other hand, the data fusion subgraph composed of LCM, RCM, PDD, RMD, DMD and APD clearly benefits more from the regular behavior of PolyGraph actors' predictability.

Synchronous languages. Synchronous programming languages [11,4] can be used to express a data flow between synchronous periodic nodes, in order to generate correct-by-construction programs. In these languages, all the nodes are synchronous, while in PolyGraph, some actors fire asynchronously when enabled.

Some derivations of synchronous languages attempt to capture asynchronous behavior. In [5], a strict interface is defined between synchronous signals of the synchronous language and asynchronous channels. This strict interface does not however provide the means to have an integrated specification of synchronous and asynchronous behavior. In [12,15], the authors use buffering to bridge the gap between synchronous nodes synchronized on compatible clocks. This approach remains globally synchronous, and is thus less adapted than PolyGraph when purely asynchronous behavior is needed. However, it would be a good output for code generation from a polygraph in which all actors are timed.

9 Future Work

Automated assistance. We mentioned automated assistance for refining incorrect polygraphs in Sect. 6. To achieve consistency, it includes finding correct rates, and to achieve liveness, correct phases and initial markings. Integrating mechanisms to achieve this in a modeling tool for polygraphs is the next step. In addition, adapting existing real-time and performance analysis techniques available for live SDF graphs and derivatives to live polygraphs would allow to benefit from their rich semantic and obtain better results.

Formalism extensions. An interesting perspective is extending PolyGraph to have a more flexible execution semantic. Our recent work [13] shows how deterministic dynamic reconfigurations can be introduced in the data flow graph.

The next step in extending PolyGraph is to define composition operators for hierarchic refinement. This would allow the modeling of very large scale systems. For example, the fusion subgraph of the ADAS example containing RMD, DMD, PDD and APD could be folded into an equivalent actor. With an unambiguous and compatible interface defined for this actor, the work of refining the modeling of the fusion subgraph could be delegated by the system designer to another team.

Compatibility and code generation. As mentioned in the previous section, there are similarities between PolyGraph and language approaches to handle real-time and asynchronous components.

It should be possible for example to use PolyGraph to statically analyze the performance of such a system, and generate code in the Lingua Franca (LF) meta-language [25] to have a deterministic implementation conforming to the PolyGraph specification. This would require first to refine the equivalences with statements and concepts of LF, and decide which actors could benefit from being generated as reactors (e.g. the EBS actor in the ADAS use case).

For subgraphs of a PolyGraph in which all actors are timed, generating code in Lucy-N [15] would also allow to have an implementation conforming to the PolyGraph specification. In this case, equivalences must also be refined first.

10 Conclusion

In this work, we have introduced PolyGraph, a data flow formalism extending SDF with synchronous firing semantics for the actors. We defined its semantics, discussed its properties and illustrated them by examples. We have shown that with this extension, the existing conditions to decide of a given SDF graph's consistency and liveness are 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. We defined an algorithm for checking liveness of a given polygraph, and proved its soundness and completeness. We proposed a methodology to model a CPS with data fusion kernels, illustrated on an ADAS use case. Finally, to enable tool-assisted analysis of polygraphs, we have proposed a framework relying on DIVERSITY to verify their liveness.

We performed experiments with this tool on a few small realistic examples as well as a large set of automatically generated models. The results demonstrate that our tool is able to soundly and efficiently decide whether a given model is live, requiring only the analysis of one consistent execution and avoiding any risk of combinatorial explosion due to branching. Soundness and efficiency of the proposed verification approach directly follow from our theoretical results.

Thanks to rational communication rates and channel states, a polygraph model approximates a desired non-linear (but regular) behavior of a modeled system by a linear behavior in rational numbers, which brings the benefit to facilitate reasoning about the properties of the model. It inherits from SDF such strong properties as the link between consistent executions and repetition vectors, the existence of a minimal repetition vector for a consistent polygraph, and the fact that the existence of a live execution implies that a valid execution can never lead to a deadlock. Our experience suggests that by exhibiting both synchronous and asynchronous behavior, tightly integrated in a single formalism, PolyGraph can help engineers to model and analyze complex CPS with real-time interfaces and compute intensive kernels.

Acknowledgements. 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. The authors thank all the members of the FACE team for useful discussions, and in particular Stéphane Louise for his contribution to the initial ideas of the original conference paper [14]. We are also grateful to the anonymous reviewers for their valuable comments.

References

1. Amarasinghe, S., Karczmarek, M., Lin, J., Maze, D., Rabbah, R.M., Thies, W., et al.: Language and compiler design for streaming applications. International Journal of Parallel Programming 33(2-3), 261–278 (2005)
2. 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)
3. 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)
4. Berry, G., Gonthier, G.: The Esterel synchronous programming language: design, semantics, implementation. Science of Computer Programming 19(2), 87 – 152 (1992)
5. Berry, G., Ramesh, S., Shyamasundar, R.: Communicating reactive processes. In: Proceedings of the 20th ACM SIGPLAN-SIGACT symposium on Principles of programming languages. pp. 85–98 (1993)
6. Bilsen, G., Engels, M., Lauwereins, R., Peperstraete, J.A.: Cyclo-static data flow. In: Proc. of ICASSP. vol. 5, pp. 3255–3258 (1995)
7. 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)
8. Borosh, I., Treybig, L.B.: Bounds on positive integral solutions of linear diophantine equations ii. Canadian Mathematical Bulletin 22(3), 357–361 (1979)
9. 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)
10. Breaban, G., Stuijk, S., Goossens, K.: Efficient synchronization methods for LET-based applications on a multiprocessor system on chip. In: Design, Automation Test in Europe Conference Exhibition (DATE), 2017. pp. 1721–1726 (2017)
11. Caspi, P., Pilaud, D., Halbwachs, N., Plaice, J.A.: Lustre: A declarative language for real-time programming. In: Proceedings of the 14th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages. p. 178–188. POPL '87, Association for Computing Machinery, New York, NY, USA (1987)
12. 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)
13. Dubrulle, P., Gaston, C., Kosmatov, N., Lapitre, A.: Dynamic reconfigurations in frequency constrained data flow. In: Ahrendt, W., Tapia Tarifa, S.L. (eds.) Integrated Formal Methods. pp. 175–193. Springer International Publishing, Cham (2019)
14. Dubrulle, P., Gaston, C., Kosmatov, N., Lapitre, A., Louise, S.: A data flow model with frequency arithmetic. In: Proc. of FASE. LNCS, vol. 11424, pp. 369–385 (2019)
15. 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)
16. Gaston, C., Le Gall, P., Rapin, N., Touil, A.: Symbolic execution techniques for test purpose definition. In: Testing of Communicating Systems (TestCom). Springer (2006)
17. 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)
18. 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)

19. 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)
20. Goubier, T., Sirdey, R., Louise, S., David, V.: ΣC: A programming model and language for embedded manycores. In: International Conference on Algorithms and Architectures for Parallel Processing. pp. 385–394. Springer (2011)
21. Kahn, G., MacQueen, D., Laboria, I.: Coroutines and Networks of Parallel Processes. IRIA Research Report, IRIA laboria (1976)
22. Klikpo, E.C., Munier-Kordon, A.: Preemptive scheduling of dependent periodic tasks modeled by synchronous dataflow graphs. In: Real-Time Networks and Systems RTNS. pp. 77 – 86. Brest, France (Oct 2016)
23. Kugele, S., Obergfell, P., Broy, M., Creighton, O., Traub, M., Hopfensitz, W.: On service-orientation for automotive software. In: 2017 IEEE International Conference on Software Architecture (ICSA). pp. 193–202 (April 2017)
24. Lee, E.A., Messerschmitt, D.G.: Static scheduling of SDF programs for digital signal processing. IEEE Transactions on Computers C-36(1), 24–35 (1987)
25. Lohstroh, M., Romeo, Í.Í., Goens, A., Derler, P., Castrillon, J., Lee, E.A., Sangiovanni-Vincentelli, A.: Reactors: A deterministic model for composable reactive systems. In: Chamberlain, R., Edin Grimheden, M., Taha, W. (eds.) Cyber Physical Systems. Model-Based Design. pp. 59–85. Springer International Publishing, Cham (2020)
26. Marchetti, O., Munier-Kordon, A.: A sufficient condition for the liveness of weighted event graphs. European Journal of Operational Research 197(2), 532 – 540 (2009)
27. 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)
28. Oh, H., Ha, S.: Fractional rate dataflow model for efficient code synthesis. Journal of VLSI Signal Process. Syst. Signal Image Video Technol. 37(1), 41–51 (2004)
29. de Oliveira Castro, P., Louise, S., Barthou, D.: Reducing memory requirements of stream programs by graph transformations. In: 2010 International Conference on High Performance Computing & Simulation. pp. 171–180. IEEE (2010)
30. Selva, M.: Performance monitoring of throughput constrained dataflow programs executed on shared-memory multi-core architectures. Theses, INSA de Lyon (2015)
31. Singh, A., Baruah, S.: Global EDF-based scheduling of multiple independent synchronous dataflow graphs. In: 2017 IEEE Real-Time Systems Symposium (RTSS). pp. 307–318 (Dec 2017)
32. Singh, A., Ekberg, P., Baruah, S.: Uniprocessor scheduling of real-time synchronous dataflow tasks. Real-Time Systems 55(1), 1–31 (2019)
33. The List Institute, CEA Tech: The DIVERSITY tool, `http://projects.eclipse.org/proposals/eclipse-formal-modeling-project/`
34. Theelen, B.D., et al.: Scenario-aware dataflow. Tech. Rep. ESR-2008-08, TUE (2008)
35. 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)