Schedule synthesis for synchronous dataflow models with lower and upper timing bounds

JOEP VAN WANROOIJ, TWAN BASTEN, and MARC GEILEN, Eindhoven University of Technology, The Netherlands

Homogeneous Synchronous DataFlow Graphs (HSDFGs) have become a popular method for analysing the performance of manufacturing systems. Manufacturing tasks, modelled by actor firings in an HSDFG, are bounded by their earliest possible starting times, determined by the completion of preceding tasks. Taking into account these lower bounds, an HSDFG represents different possible task schedules for these tasks, namely any actor firing scheme that satisfies these lower bounds. However, in some cases, tasks in a manufacturing system must be completed before a deadline, which introduces an upper bound. Additionally, the relative start times between tasks may need to adhere to a lower bound. Such lower and upper bounds are not naturally supported by classical HSDFGs. In such cases, an HSDFG cannot represent all relevant aspects of the behaviour of a manufacturing system. This paper extends the HSDFG model to support the specification of lower- and upper-bound constraints on timing differences between actor-firing starts and completions. The paper then presents a new method for transforming extended HSDFGs into a (max, +) linear system in the form of its state-space matrices. This system can be used to synthesize a task schedule that adheres to the specified lower and upper bounds while achieving the earliest possible execution times, optimizing throughput or makespan. We illustrate the necessity for specifying lower and upper bounds, and the application of our techniques, with a manufacturing system case study.

CCS Concepts: • **Computer systems organization → Embedded and cyber-physical systems**; • **Theory of computation → Parallel computing models**; • **Mathematics of computing → Discrete mathematics**.

Additional Key Words and Phrases: Synchronous DataFlow, (max, +) algebra, manufacturing systems, scheduling, schedule synthesis

ACM Reference Format:
Joep van Wanrooij, Twan Basten, and Marc Geilen. 2025. Schedule synthesis for synchronous dataflow models with lower and upper timing bounds. *ACM Trans. Embedd. Comput. Syst.* 1, 1 (August 2025), 28 pages. https://doi.org/XXXXXXX.XXXXXXX

Acknowledgments

This work was supported by NXTGEN Hightech, funded by the Dutch National Growth Fund, as part of the Semicon 03 project.

1 Introduction

Manufacturing systems are growing increasingly complex due to ongoing demand for faster production and accommodating a greater variety of product types. Model-based design is used to address this complexity, where model abstractions are used to describe a system's behaviour and

Authors' Contact Information: Joep van Wanrooij, j.c.p.van.wanrooij@tue.nl; Twan Basten; Marc Geilen, Eindhoven University of Technology, Eindhoven, The Netherlands.

ACM 1558-3465/2025/8-ART
https://doi.org/XXXXXXX.XXXXXXX

its timing. The models are then used to analyze, optimize, and/or synthesize systems. Examples of such models are Homogeneous Synchronous DataFlow (HSDF) [18, 21] or Petri Nets [14, 20].

The throughput of a manufacturing system is often an important indicator used to measure its productivity, with throughput being determined by the order in which machine tasks are executed and the duration of these tasks. The arrangement of machine tasks, together with their start times, is referred to as a task schedule. The schedule can be analysed to determine properties such as throughput and makespan, while other analysis techniques can be used to find, for example, the critical path [13].

For most manufacturing systems, large parts of their behaviour are deterministic once the machine parameters are configured and when input products arrive predictably. This means that static task schedules are followed that guarantee correct behaviour and a consistent productivity. An HSDF graph (HSDFG) can represent such static and deterministic task execution well.

In an HSDFG, a task can only be scheduled once all preceding tasks are completed. In other words, the start time of a task has lower bounds from the completion times of its preceding tasks. So, an HSDFG represents a (possibly infinite) number of possible task schedules, with one being the As-Soon-As-Possible (ASAP) schedule. This is the preferred schedule for most manufacturing systems, as it yields the highest possible productivity. One approach to analyse this ASAP schedule is through the (max, +) algebraic semantics of an HSDFG [9, 12]. This semantics abstracts from the model's graph structure, focusing on timing only, and captures precisely the ASAP schedule. The (max, +) semantics enables various analysis techniques on the ASAP schedule, such as throughput, latency, and makespan analysis.

However, the ASAP schedule of an HSDFG does not always capture the intended or required manufacturing behaviour. In many manufacturing systems, certain tasks have upper-bound constraints. For instance, a product that must be placed on a substrate before glue hardens, a conveyor belt that must deliver a product within a specific time frame to prevent congestion, or an inspection that must be completed before a product moves out of sight. Furthermore, certain types of lower bounds, such as a lower bound on the start of one task (e.g., an inspection) relative to the start of another task (e.g., the transport of the object being inspected), are not supported in the classical HSDF formalism. We illustrate the need to model such lower and upper bounds with an industrial motivating example, given in Section 3.

Although HSDFGs are highly effective to model manufacturing systems, they do not support upper bounds on the timing of task executions. Also lower bounds, such as the already mentioned lower bounds on the relative start times of two tasks, are not supported in classical HSDF. The first contribution of this paper is the integration of all types of lower and upper bounds on time differences between actor firings in the HSDF formalism. We show how such bounds can be translated to classical channel dependencies using the concept of actors with negative execution times, as introduced in [10]. This formulation creates token-free cycles in an HSDFG, which, from an operational perspective, leads to deadlocks in the HSDFG. However, we show that, from an equational perspective, if all token-free cycles have non-positive overall execution times, valid schedules respecting the bounds exist.

The token-free cycles introduced by the formulation of lower and upper bounds prevent an HSDFG from being translated into the desired (max, +) semantics for ASAP schedule analysis using the method described in [9]. The second contribution of this paper is a new method for converting extended HSDF models to their (max, +) semantics. Proof is provided to demonstrate that the resulting schedule is both valid and equal to the ASAP schedule. The resulting (max, +) linear system continues to be effective for synthesising the ASAP schedule and performing various analysis techniques, just as it was with the classical HSDF model.

The extended HSDFGs introduced in this paper are qualitatively evaluated in Section 8, where we synthesize ASAP schedules for manufacturing use cases requiring lower- and upper-bound modelling. In addition, we quantitatively evaluate the runtime performance of our method for converting HSDFGs into (max, +) semantics and compare it with the conventional approach in Section 9.1. Our method is implemented as an extension to SDF^3 [23], a tool for analysing, synthesizing, and converting (H)SDFGs, which is available on GitHub [8]. Further information on the supporting tools, including CMTRACE, which is used to generate the Gantt charts presented in this paper, is available on the website [7].

The paper proceeds as follows. First, we compare this work to related work in Section 2. In Section 3, a motivating example is introduced that explains the need for lower and upper bounds. Section 4 introduces the necessary preliminaries. We then explain how to model lower and upper bounds in HSDFGs in Section 5. Section 6 explains why we require a new method for determining the matrices of the linear system. In Section 7, we prove that the proposed method yields a valid ASAP schedule for HSDFGs without inputs and outputs; a more general proof, including inputs and outputs, is provided in Appendix A. Section 8 completes the model of the motivating example and shows further applications of modelling lower and upper bounds. We evaluate the performance of our HSDFG extension in Section 9, and conclude in Section 10.

2 Related work

As discussed by Lee [17], dealing with the passing of time is a critical factor in the development of robust and reliable Cyber-Physical Systems (CPS) such as manufacturing systems. These systems interact with the physical world, which means timing cannot simply be abstracted away. That is why it is becoming increasingly important to incorporate time in our models.

Synchronous Data Flow (SDF) [18] is such a model. The model is extended with time [22] and can have external inputs and outputs [11]. A specialized form of SDF, referred to as Homogeneous SDF (HSDF), is defined by the property that all actors consume and produce exactly one token per activation. An HSDF models deterministic and synchronous behaviour, in the form of HSDF graphs (HSDFGs), making it very suitable for manufacturing systems. Manufacturing systems typically operate repetitively with a predictable flow of input and output products, where tasks must wait on the completion of other tasks. Geilen [9] analyses (H)SDFGs by converting the graphs to a (switched) (max, +) linear system. This system captures the state of an (H)SDFG for each iteration of its modelled tasks. This transformation enables efficient performance analysis for the system being modelled, such as throughput analysis.

In [10], the concept of dependencies on future events in HSDF is introduced. The work shows that actors in HSDF do not necessarily have to represent physical activities, and thus do not always have to follow a natural temporal order. The work introduces the notion of negative execution time for actors, enabling the modelling of dependencies on future events. Our paper builds on that fact, further utilizing this idea of using negative execution time to describe lower and upper bounds. The formulation of these bounds in HSDFGs creates cycles without tokens with a non-positive weight, seemingly making the HSDF model no longer operational. However, we show that these situations do not compromise the existence of a valid schedule.

The concept of representing lower and upper bounds in HSDFGs is inspired by the introduction of constraint graphs with lower and upper bounds [19]. In [19], positive and negative edge weights are used to define spacing constraints between VLSI circuit elements. A longest-path algorithm is then applied to determine the minimal (most compact) solution for these constraints. Another example, which shows the need to model lower and upper bounds in manufacturing systems, is found in [28], where constraint graphs are used to derive a schedule for a set of jobs on a production printer. The representation of lower and upper bounds in HSDFGs goes a step further, as, in contrast

to constraint graphs, an HSDFG can capture cyclic behaviour. This means that HSDFGs represent repetitive schedules. By translating an HSDFG to its (max, +) representation, these schedules can be efficiently analysed and synthesized for any given (possibly infinite) input sequence.

A transformation similar to converting (H)SDF models into (max, +) linear systems has been discussed for a special class of Petri nets known as Timed Petri nets (TPN), under the assumption that the nets are safe and decision-free [2, 6]. As with (H)SDF models, TPNs are modelled using an initial state from which the next state is computed after one iteration of the net. However, unlike the approach taken in this paper, these works do not address the formulation of lower and upper bounds. Furthermore, the linear systems derived from TPNs do not include the starting time of transitions, meaning they do not support schedule synthesis.

Schedulability analysis of real-time system specifications has been done in the context of Timed Constraint Petri Nets (TCPN) [25]. In this work, minimum and maximum timing constraints are assigned to places or transitions, similar to the lower and upper bounds in our work. Reachability simulation is used to verify if a place of interest in a TCPN can be reached. A subsequent timing analysis verifies whether this place remains reachable under the imposed timing constraints. In contrast to our work, this work focusses on analyzing the schedulability of the TCPN and does not synthesize the ASAP schedule that respects all the constraints.

Scheduling of production in manufacturing systems gives rise to the research domain of job-shop scheduling. In a classical job-shop scheduling problem, a set of jobs must be scheduled on a given set of machines. In general, jobs consist of operations that must access specific manufacturing machines in a specific order. Many variants with a wide variety of constraints and decision variables have been studied [16]. Our approach relates to schedulability analysis and ASAP-schedule synthesis. Specifically, we consider these aspects in the presence of time-bound constraints. In [3], schedulability for a lacquer manufacturing system is analysed. There are different lacquer products, each following a sequence of production steps that are subject to additional lower- and upper-bound timing constraints. The UPPAAL framework [4, 26] is used to heuristically explore feasible schedules via reachability analysis. In [5], a job-shop scheduling problem is formulated for an industrial packaging use case involving light sources. Jobs are partitioned into batches, where each batch must satisfy a batching constraint, meaning that all jobs belonging to the same batch must be processed consecutively and be done before a certain deadline. The ILOG CPLEX library [15] is used to solve the scheduling problem. Unlike our approach, both works aim to verify whether a feasible schedule exists giving machine allocations and operation ordering. If so, it is possible to model the machine allocation and operation ordering as an HSDFG, synthesize the ASAP schedule, and perform for instance a makespan analysis. This shows the generality of our approach and its ability to complement existing allocation-solving frameworks such as the UPPAAL and ILOG CPLEX frameworks. In Section 8.2, we show how to synthesize an ASAP schedule for an allocated job in the lacquer manufacturing system. In Section 8.3, we show schedule synthesis for batches of jobs with a predetermined operation orderings and machine allocations.

3 Motivating example

An example of a complex manufacturing chain is the die (chip) attaching process. As detailed in [24] (illustrated in Fig. 12.6 on page 413), the die attaching process consists of three main steps: Dispense, Die attaching, and Cure. In the Dispense step, paste is applied to a substrate. The substrate provides mechanical support, electrical connectivity, and thermal dissipation for the die. During the Die attaching step, the die is placed onto the paste. Finally, in the Cure step, the paste is heated, securing the die firmly to the substrate. This section focuses specifically on the Die attaching step, while Section 10 models the complete die attaching process. Figure 1 shows a simplified view of a die attach manufacturing machine.

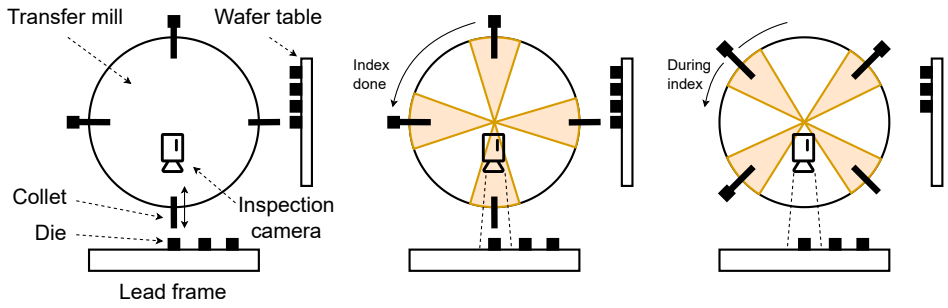

Fig. 1. Schematic of a die attach manufacturing machine. The illustration shows two inspection scenarios: the middle image depicts the collet obstructing the camera's view, while the right image shows a clear view of the substrate holder.

The machine picks up dies from a diced wafer and places them onto substrates at extremely high speeds by rotating a transfer mill (task `idxMill`). This mill is equipped with four collets that pick up (`diePickup`) and place (`dieAttach`) dies. After a die is picked up, the wafer indexes (`idxWafer`) to the next die on the wafer. Simultaneously, once the die is placed on the substrate, the substrate holder indexes (`idxSub`) to position the next empty substrate for placement. One index of the mill means a counter clockwise rotation of 90 degrees. During the index, a camera inspects (`inspect`) the die placement to detect any misalignment. Table 1 presents the durations of the die attach machine's tasks in an unspecified time unit.

Table 1. Duration of the tasks of the die attach machine

Task	Duration	Task	Duration	Task	Duration
`idxMill`	50	`diePickup`	25	`dieAttach`	25
`idxWafer`	30	`idxSub`	30	`inspect`	20

Figure 1 also shows the time window during a transfer mill index when the camera can capture an image of the substrate. The middle image shows the position of the collets when the transfer mill is not moving. In this position, the camera's field of view is obstructed. Thus, no image can be taken. The collets are only out of view during an index of the transfer mill, as shown in the right image. The inspection can begin 10 time units after the start of the mill index, but not earlier, and must complete 10 time units before the index ends. This means that the start of the camera inspection is constrained by the start and end of the transfer mill index. Therefore, a logical approach to model this is by establishing two bounds. The first is a lower bound on the start of the camera inspection relative to the start of the transfer mill index. The second is an upper bound relative to the end of the camera inspection, as it must be completed before the transfer mill index finishes.

4 Preliminaries

In this section, we introduce timed synchronous dataflow graphs. We explain them from an operational perspective and describe the schedules they represent, followed by their formal definition. Additionally, we introduce (max, +) algebra, with which we can formulate a (max, +) linear system representation of an HSDFG.

4.1 Homogeneous Synchronous DataFlow Graphs

An HSDFG is a graph that contains actors that are connected with channels. The channels are directional and transfer tokens from one actor to another. Inputs and outputs are open channels, meaning tokens are transferred from and to an external environment. An actor can fire only when a token is present on all its incoming channels and inputs. Firing means that the actor activates for a duration of its assigned execution time. After this time, the actor produces a token on all of its outgoing channels and outputs. Figure 2 shows an HSDFG of the die attach manufacturing machine introduced in Section 3. The yellow circles represent the actors, with their names and durations labelled inside each circle. The arrows depict the channels of the HSDFG. The black dots represent the initial tokens. Labels x_1, x_2, x_3, and x_4 are added for reference. The HSDFG is currently in its initial state and needs the initial tokens to start its execution. For example, the actor `idxMill` is able to fire, but if token x_3 were absent, would never be able to fire. When all the actors of the graph fire once, the tokens within the graph are replaced by new ones on the same channels. We call this collection of actor firings an *iteration* of the graph.

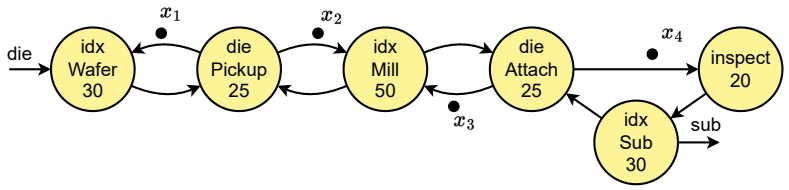

Fig. 2. HSDFG of the die attach manufacturing machine.

Figure 3 shows a Gantt chart illustrating three iterations of the graph, where the initial tokens are available from time 0. The first row shows the arrivals of new dies on the `die` input at times 0, 75, and 180. This means that from time 0, the transfer mill can make an index (`idxMill`), an inspection can be performed on an assembled substrate already within the machine (`inspect`), and the wafer can index (`idxWafer`). After the inspection, the substrate holder, which holds the inspected die, can index (`idxSub`), advancing the next empty substrate into position. When the substrate holder index finishes, the assembled substrate exits the machine, which is modelled with the `sub` output. The die can only be picked up from the wafer (`diePickup`) once both the transfer mill and wafer have completed their indexing. Similarly, placing a die onto a new substrate (`dieAttach`) requires both the transfer mill and substrate holder to finish their indexing. This cycle repeats until all dies on the `die` input have been processed.

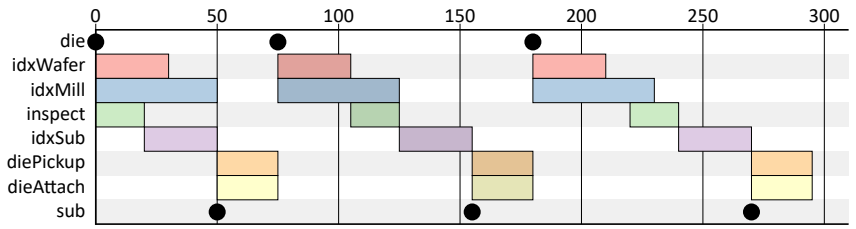

Fig. 3. A schedule for three iterations of the HSDFG of the die attach manufacturing machine of Figure 2.

Section 3 discusses the bounds of the camera inspection on the start and end of the transfer mill index. However, these bounds are not represented in the HSDFG of Figure 2. This can be seen in

the schedule shown in Figure 3. In the schedule, the lower bound on the inspection is violated, as the first inspection starts at the same time as the mill starts its rotation. Similarly, the upper bound is not respected for the second inspection, as it finished exactly when the second mill rotation ends. Additionally, the third inspection continues even after the third mill rotation has finished. Clearly, this schedule does not reflect the correct operation of the die attach manufacturing machine, even though it is a valid schedule for the given HSDFG. In Section 5, we explain how the correct bounds can be formulated with an HSDFG. The remainder of this section formally defines HSDFGs and the schedules they represent.

In HSDFGs, also known as a single-rate SDFGs, all actors consume and produce exactly one token on all their in- and outputs. The definition presented here is inspired by [10] as it includes graph inputs and outputs and allows actors to have negative execution times.

Definition 4.1 (SDFG). An HSDFG is a graph $G = (A, C, I, O, e, d_I, d_O)$ with a set A of actors and a set $C \subseteq A \times \mathbb{N} \times A$ of channels. A channel $(a_1, n, a_2) \in C$ is a channel from actor a_1 to a_2 holding n initial tokens. The execution times of the actors are defined with the mapping $e : A \to \mathbb{R}$. I is the set of inputs, with input dependencies $d_I : I \to A$. O is the set of outputs, with output dependencies $d_O : O \to A$.

We define a schedule of an HSDFG as follows.

Definition 4.2 (Schedule). A schedule of an HSDFG is a function $\sigma : A \times \mathbb{N} \to \mathbb{R}_{\geq 0}$. This function assigns a starting time to the firings of the actors in A. $\sigma(a, k)$ represents the starting time of the k-th firing of actor a. The schedule σ is considered *valid* for given input arrivals $\alpha : I \times \mathbb{N} \to \mathbb{R}_{\geq 0}$ if the following conditions hold.

- For all $(a_1, n, a_2) \in C$, the so-called *channel constraint* $\sigma(a_2, k + n) \geq \sigma(a_1, k) + e(a_1)$ holds for every $k \in \mathbb{N}$.
- For all $u \in I$ with $d_I(u) = a$, the constraint $\sigma(a, k) \geq \alpha(u, k)$ holds for every $k \in \mathbb{N}$.

Output function $\pi : O \times \mathbb{N} \to \mathbb{R}$, the production of tokens on the output $y \in O$, is given as $\pi(y, k) = \sigma(d_O(y), k) + e(d_O(y))$ for all $k \in \mathbb{N}$. A valid schedule σ for input arrivals α is called an *as-soon-as-possible* (ASAP) schedule if, for any valid schedule σ' with input arrivals α of the HSDFG, the condition $\sigma(a, k) \leq \sigma'(a, k)$ holds for all $a \in A$ and $k \in \mathbb{N}$, i.e., $\sigma \leq \sigma'$ according to the element-wise ordering. For given input arrivals, the ASAP schedule is unique.

Definition 4.2 defines infinite schedules. The definition remains consistent for finite schedules.

4.2 (max, +) algebra

(max, +) algebra [2] knows two basic operations, maximisation ($\oplus$) and addition ($\otimes$), with $x \oplus y = \max(x, y)$ and $x \otimes y = x + y$. Here, x and y are both elements of $\mathbb{T} = \mathbb{R} \cup \{-\infty\}$, referred to as the set of timestamps. For all $x \in \mathbb{T}$, $x \oplus -\infty = x$ and $x \otimes -\infty = -\infty$. The $\otimes$ operation takes precedence over the $\oplus$ operation. (max, +) algebra is a linear algebra and can thus be extended to timestamp vectors in $\mathbb{T}^n$ and matrices in $\mathbb{T}^{n \times m}$, where $n, m \in \mathbb{N}$. An entry in the matrix $\mathbf{A} \in \mathbb{T}^{n \times m}$ is referred to as $\mathbf{A}_{i,j}$, where i indicates the row and j the column. Given a second matrix $\mathbf{B} \in \mathbb{T}^{n \times m}$, the matrix operation $\mathbf{A} \oplus \mathbf{B}$ is defined by $[\mathbf{A} \oplus \mathbf{B}]_{i,j} = \mathbf{A}_{i,j} \oplus \mathbf{B}_{i,j}$ for each entry (i, j). Given a matrix $\mathbf{C} \in \mathbb{T}^{m \times p}$ with $p \in \mathbb{N}$, the operation $\mathbf{A} \otimes \mathbf{C}$ is defined by $[\mathbf{A} \otimes \mathbf{C}]_{i,j} = \bigoplus_{k=1}^{m} \mathbf{A}_{i,k} \otimes \mathbf{C}_{k,j}$. Here, $\bigoplus$ represents the iterative application of the $\oplus$ operation. Let $\mathbf{A} \in \mathbb{T}^{n \times n}$ be a square matrix and $q \in \mathbb{N}$ with $q > 0$. Then, $\mathbf{A}^q = \mathbf{A}^{q-1} \otimes \mathbf{A}$, with $\mathbf{A}^0 = \mathbf{I}$ the $n \times n$ identity matrix with 0 entries on the diagonal and $-\infty$ entries elsewhere. The zero vector in $\mathbb{T}^n$ is expressed as $\mathbf{0}$, with the size n being derivable from the context.

4.3 SDFGs as (max, +) linear systems

Modelling the behaviour of an HSDFG using a (max, +) linear system involves capturing the state (timestamp) of the tokens for each iteration of the graph. Each initial token in the graph is assigned a timestamp. These timestamps are collected in a vector. If the HSDFG has m initial tokens, the state for the k-th iteration is given by the timestamp vector $\mathbf{x}(k) \in \mathbb{T}^m$. The timestamp of token x for iteration k is denoted as $\mathbf{x}(k)[x]$. The sequence $\mathbf{x}$ of vectors describes the state evolution of the HSDFG over time. Similarly, we define the input and output event timestamp vectors for the k-th iteration as $\mathbf{u}(k) \in \mathbb{T}^{|I|}$ and $\mathbf{y}(k) \in \mathbb{T}^{|O|}$, respectively. Here, $\mathbf{u}(k)[u]$ represents the timestamp of input $u \in I$, while $\mathbf{y}(k)[y]$ represents the timestamp for the output $y \in O$. Finally, we introduce the timestamp vector $\mathbf{s}(k) \in \mathbb{T}^{|A|}$, which captures the start times of actor firings at iteration k. The start time of the k-th firing of actor $a \in A$ is denoted as $\mathbf{s}(k)[a]$. With these vectors, we can define the (max, +) linear state space and its matrices $\mathbf{A}$, $\mathbf{B}$, $\mathbf{C}$, and $\mathbf{D}$ as follows [2]. The linear system is extended with two matrices $\mathbf{H}$ and $\mathbf{J}$, used to compute the start time of the k-th firing of the actors in the system, in a manner similar to [1].

Definition 4.3 (State-space representation). Let G be an HSDFG with m initial tokens. We introduce the timestamp vectors $\mathbf{x}(k) \in \mathbb{T}^m$, $\mathbf{u}(k) \in \mathbb{T}^{|I|}$, $\mathbf{y}(k) \in \mathbb{T}^{|O|}$, and $\mathbf{s}(k) \in \mathbb{T}^{|A|}$ for the k-th iteration of G with $k \in \mathbb{N}$. The (max, +) linear state-space equations for G are:

$$\begin{aligned} \mathbf{x}(k+1) &= \mathbf{A} \otimes \mathbf{x}(k) \oplus \mathbf{B} \otimes \mathbf{u}(k) \\ \mathbf{y}(k) &= \mathbf{C} \otimes \mathbf{x}(k) \oplus \mathbf{D} \otimes \mathbf{u}(k) \\ \mathbf{s}(k) &= \mathbf{H} \otimes \mathbf{x}(k) \oplus \mathbf{J} \otimes \mathbf{u}(k) \end{aligned}$$

The state-space matrices are as follows:

- The *state* matrix $\mathbf{A} \in \mathbb{T}^{m \times m}$ represents the relation between initial tokens before and after an iteration;
- the *input* matrix $\mathbf{B} \in \mathbb{T}^{m \times |I|}$ represents the relation from inputs to initial tokens;
- the *output* matrix $\mathbf{C} \in \mathbb{T}^{|O| \times m}$ represents the relation from initial tokens to outputs;
- the *feed-forward* matrix $\mathbf{D} \in \mathbb{T}^{|O| \times |I|}$ represents the relation from inputs to outputs;
- the *actor* matrix $\mathbf{H} \in \mathbb{T}^{|A| \times m}$ represents the relation from initial tokens to the start times of the actors;
- the *actor-input* matrix $\mathbf{J} \in \mathbb{T}^{|A| \times |I|}$ represents the relation from the inputs to the start times of the actors.

The state-space equations can be combined as a single matrix-vector equation.

$$\begin{bmatrix} \mathbf{x}(k+1) \\ \mathbf{y}(k) \\ \mathbf{s}(k) \end{bmatrix} = \begin{bmatrix} \mathbf{A} & \mathbf{B} \\ \mathbf{C} & \mathbf{D} \\ \mathbf{H} & \mathbf{J} \end{bmatrix} \otimes \begin{bmatrix} \mathbf{x}(k) \\ \mathbf{u}(k) \end{bmatrix}$$

The state-space representation of an SDFG, and by extension an HSDFG, can by computed through so-called symbolic simulation [9]. This traditional algorithm for finding the state-space matrices makes use of the operational properties of the (H)SDFG. In this algorithm, the value of each initial token is expressed as a symbolic timestamp vector. When an actor can fire, it consumes tokens from its incoming channels and generates new tokens on its outgoing channels. The newly produced tokens are assigned a symbolic timestamp vector, computed as the element-wise maximum of the symbolic timestamp vectors of the tokens on the incoming channels, plus the actor's execution time. Once all actors have fired, completing one iteration of the HSDFG, the initial tokens are replaced by new tokens on the same channels. The symbolic timestamps assigned to these new tokens are then gathered to construct the matrices. In the symbolic simulation algorithm, actors

fire in a self-timed manner, meaning they fire as soon as they are able. This results in a state-space representation that corresponds to ASAP schedules [9].

Definition 4.4 (State-space schedule). The schedule σ_s for the given input vector sequence $\mathbf{u}(k)$ of the state-space representation of an HSDFG is defined as $\sigma_s(a,k) = \mathbf{s}(k)[a]$ for all $a \in A$ and $k \in \mathbb{N}$, given the initial state vector $\mathbf{x}(0) = \mathbf{0}$. The input arrivals are defined as $\alpha_s(u,k) = \mathbf{u}(k)[u]$ for all $u \in I$ and $k \in \mathbb{N}$; output arrivals are defined as $\pi_s(y,k) = \mathbf{y}(k)[y]$ for all $y \in O$ and $k \in \mathbb{N}$.

5 Lower and upper bounds in HSDFGs

We have introduced HSDFGs from an operational perspective. However, as shown in Definition 4.2, we can also approach the graph and its schedules from an equational viewpoint. The bounds between the start times of actors in an HSDFG are determined by the channels within the graph. For instance, the lower bound for the channel from actor `idxMill` to `dieAttach` in Figure 2 can be written as follows. Here, σ is a valid schedule, and the bound holds for all $k \in \mathbb{N}$.

$$\sigma(\texttt{dieAttach}, k) \geq \sigma(\texttt{idxMill}, k) + e(\texttt{idxMill})$$

So, the start time of the k-th firing of the `dieAttach` actor must be greater than or equal to the completion time of the k-th firing of the `idxMill` actor. There is no upper bound relation between these actors, allowing `dieAttach` to start at any time after the completion of `idxMill`.

Like the bound between `dieAttach` and `idxMill`, we can express the lower and upper bound for the camera inspection with inequalities. Recall that the camera inspection can begin only 10 time units after the transfer mill index starts and must be completed 10 time units before the index finishes. Given the actor execution times in Table 1, this means that the `inspect` must start in the interval $[10, 20]$, relative to the start time of the `idxMill` actor. This interval can be written as the following lower- and upper-bound constraints.

$$\sigma(\texttt{inspect}, k) \geq \sigma(\texttt{idxMill}, k) + 10$$
$$\sigma(\texttt{inspect}, k) \leq \sigma(\texttt{idxMill}, k) + 20$$

Both constraints are not aligned with the standard equational interpretation of a channel constraint in an HSDFG (Definition 4.2). To align them with this standard, we introduce the actor `unblock` with $e(\texttt{unblock}) = -e(\texttt{idxMill}) + 10 = -50 + 10 = -40$ for the lower-bound constraint.

$$\sigma(\texttt{unblock}, k) \geq \sigma(\texttt{idxMill}, k) + e(\texttt{idxMill})$$
$$\sigma(\texttt{inspect}, k) \geq \sigma(\texttt{unblock}, k) + e(\texttt{unblock})$$

For the upper-bound constraint, we rearrange the equation to derive the following:

$$\sigma(\texttt{idxMill}, k) \geq \sigma(\texttt{inspect}, k) - 20$$

We can apply the same approach as for the lower bound by introducing an actor `block` with $e(\texttt{block}) = -e(\texttt{inspect}) - 20 = -20 - 20 = -40$.

$$\sigma(\texttt{block}, k) \geq \sigma(\texttt{inspect}, k) + e(\texttt{inspect})$$
$$\sigma(\texttt{idxMill}, k) \geq \sigma(\texttt{block}, k) + e(\texttt{block})$$

We may now add channels in line with the derived constraints, along with the `unblock` and `block` actors, to the HSDFG of the die attach manufacturing machine, resulting in the updated HSDFG shown in Figure 4. The `unblock` and `block` actors are highlighted in red to indicate that they have negative execution times. The smallest solution satisfying the constraints of the updated HSDFG for the earlier given arrival of tokens on input `die`, and therefore the ASAP schedule, is shown in Figure 5. We see that this schedule satisfies the lower- and upper-bound constraints on the firings of `inspect`. Both the `block` and `unblock` actor firings are shown with dashed borders to

indicate negatively-timed activations. We see that the start time of the `inspect` actor is determined by the `unblock` actor, whose activation begins at the end time of the `idxMill` actor.

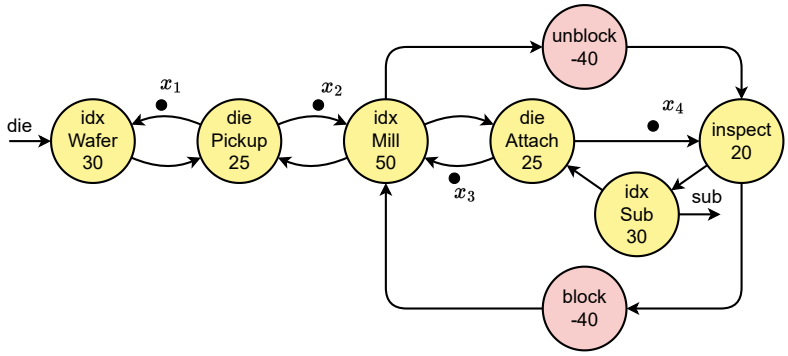

Fig. 4. HSDFG of the die attach manufacturing machine containing actors with negative execution times in red.

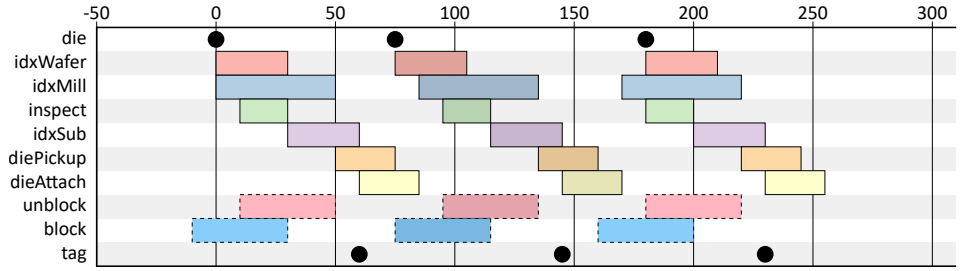

Fig. 5. The desired schedule that reflects the correct behaviour of the die attach manufacturing machine.

In Figure 4, the `idxMill` actor models the continuous motion of the transfer mill during an indexing operation. While this is one valid representation, there are multiple ways to model the blocking and unblocking behaviour associated with this motion. Another intuitive approach is to decompose the `idxMill` actor into three sub actors: `mUnblock`, `mTransfer`, and `mBlock`, and to connect the `inspect` actor between `mUnblock` and `mBlock`. Since the transfer mill makes a continuous motion, these sub-actors must execute consecutively without any delay. This imposes strict timing constraints such that $\sigma(\texttt{mTransfer}, k) = \sigma(\texttt{mUnblock}, k) + e(\texttt{mUnblock})$ and $\sigma(\texttt{mBblock}, k) = \sigma(\texttt{mTransfer}, k) + e(\texttt{mTransfer})$. Both these constraints translate to lower and upper bounds. The solution chosen in Figure 4 provides a more compact model.

In general, for a pair of actors a_1 and a_2, we want to express that the start times of actor a_2 are bounded by the interval $[r, t]$ relative to the start times of actor a_1. With a schedule σ, we can define a lower bound $r \in \mathbb{R}$ and upper bound $t \in \mathbb{R}$ between the start times of these actors as follows.

$$\sigma(a_2, k) \geq \sigma(a_1, k) + r$$
$$\sigma(a_2, k) \leq \sigma(a_1, k) + t$$

We rewrite the constraints as follows.

$$\sigma(a_2, k) \geq \sigma(a_1, k) + e(a_1) - e(a_1) + r$$
$$\sigma(a_1, k) \geq \sigma(a_2, k) + e(a_2) - e(a_2) - t$$

Representing these bounds within an HSDFG G requires a transformation as shown in Figure 6. This transformation introduces the actor a_r with $e(a_r) = -e(a_1) + r$ that represents the lower bound and actor a_t with $e(a_t) = -e(a_2) - t$ that represents the upper bound. We get the following set of constraints, which are in the standard equational form of a channel in an HSDFG.

$$\sigma(a_1, k) \geq \sigma(a_t, k) + e(a_t)$$
$$\sigma(a_t, k) \geq \sigma(a_2, k) + e(a_2)$$
$$\sigma(a_2, k) \geq \sigma(a_r, k) + e(a_r)$$
$$\sigma(a_r, k) \geq \sigma(a_1, k) + e(a_1)$$

We add the actors, with the corresponding channels to the HSDFG. If no lower bound exists, actor a_r and its incoming and outgoing channel can be omitted. Similarly, if no upper bound exists, actor a_t and its incoming and outgoing channel can be omitted. If $r = e(a_1)$ or $t = -e(a_2)$, the actors a_r or a_t can be omitted, and the channel can directly connect a_1 and a_2 in the appropriate direction.

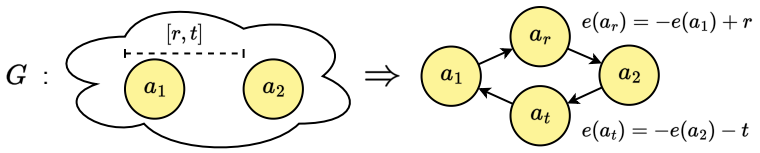

Fig. 6. Transformation on an HSDFG to represent a lower and upper bound between firings of actors a_1 and a_2.

6 Finding (max, +) linear system matrices for HSDFGs with token-free cycles

In the HSDFG shown in Figure 4, a cyclic dependency without initial tokens exists among the actors `idxMill`, `unblock`, `inspect`, and `block`. From an operational perspective, this dependency prevents any of these actors from firing, making further progress impossible. Despite this operational deadlock, the equational semantics of the HSDFG still represents the intended valid schedules. One example of such a schedule is the ASAP schedule, which represents the desired system behaviour shown in Figure 5. However, the conventional algorithm described in Section 4.3, used to express an HSDFG as a (max, +) linear system that characterizes the ASAP schedule, fails due to the presence of such cyclic dependencies.

Introducing lower and upper bounds in HSDFGs leads to the formulation of these cyclic dependencies. In cases such as the token-free cycle in Figure 4, the actors within the cycle all have at least one incoming channel without a token. Consequently, these actors can never fire according to the operational semantics, preventing the algorithm from completing the graph's iteration. Therefore, a new approach is required to find the matrices for the state-space representation. This section introduces a new method for deriving the (max, +) linear system matrices using weights of longest paths in an HSDFG that has a valid schedule. We establish that an HSDFG has a valid schedule only if the sum of the execution times of the actors within a cycle is at most zero. When this condition is met, the HSDFG represents at least one valid schedule, allowing the proposed method to be applied. We later show that the obtained (max, +) linear system representation of an extended HSDFG capturing lower and upper bounds characterizes the ASAP schedule, in line with the results for classical HSDF.

The state-space equations show that each value in the vectors $\mathbf{x}(k+1)$, $\mathbf{y}(k)$, and $\mathbf{s}(k)$ is determined by a (max, +) linear combination of all initial token timestamp values $\mathbf{x}(k)$ and input

timestamp values $\mathbf{u}(k)$ for iteration k. In other words, each linear combination represents a dependency between an initial token or an input and an initial token, output, or an actor firing within one iteration. The dependencies, such as between the start times of two actors, are defined by the channels in the graph. For instance, in the HSDFG of Figure 4, a dependency exists from actor `idxMill` to actor `dieAttach` due to the presence of the channels (`idxMill`, 0, `unblock`), (`unblock`, 0, `inspect`), (`inspect`, 0, `idxSub`), and (`idxSub`, 0, `dieAttach`). Essentially, this forms a path that starts from actor `idxMill`, passes through actors `unblock`, `inspect`, `idxSub`, and then ends at actor `dieAttach`.

Definition 6.1 (Path). Let G be an HSDFG with actors A and channels C. A path p is a finite sequence $\langle a_1, a_2, \ldots, a_j \rangle \in A^j$ $(j \geq 0)$ of actors such that there is a channel $(a_i, n_i, a_{i+1}) \in C$ for all $1 \leq i < j$. We use the notation $p(i)$ to refer to the actor a_i in the path p. A path is called *token-free* if for all $1 \leq i < j$, $(a_i, 0, a_{i+1}) \in C$.

Definition 6.2 (Path source and sink). Let G be an HSDFG with m initial tokens. A path p of length $j > 0$ has a source s and a sink t. The source s is the starting point of the path and can be:

- An initial token x_i, with $1 \leq i \leq m$, where x_i resides on the channel $(a, n, p(1)) \in C$.
- An actor $a \in A$, for which $p(1) = a$.
- An input $u \in I$, for which $p(1) = d_I(u)$.

The sink t is the end point of a path and can be:

- An initial token x_i, with $1 \leq i \leq m$, where x_i resides on the channel $(p(j), n, a) \in C$.
- An actor $a \in A$, for which a channel $(p(j), n, a) \in C$ exists. Here, actor a is not included in the path.
- An output $y \in O$, for which $p(j) = d_o(y)$.

A source s and a sink t are connected by an empty path, meaning a path p with length $j = 0$, if any of the following conditions hold:

- $s = u$ and $t = a$ for an input $u \in I$ and an actor $a \in A$ with $d_I(u) = a$;
- $s = x$ and $t = a_2$ for a token x that resides on channel $(a_1, n, a_2) \in C$;
- $s = x_i$ and $t = x_j$ for two tokens x_i and x_j, where $1 \leq i < j \leq m$, and both tokens reside on the same channel.

Table 2 presents several examples of paths that exist in the HSDFG shown in Figure 4.

Table 2. Examples of paths in the HSDFG of Figure 4

Source	Sink	Path
`idxMill`	`dieAttach`	⟨`idxMill`, `unblock`, `inspect`, `idxSub`⟩
`die`	x_2	⟨`idxWafer`, `diePickup`⟩
x_2	x_4	⟨`idxMill`, `dieAttach`⟩
x_4	`inspect`	⟨⟩
`idxSub`	`sub`	⟨`idxSub`⟩

For a valid schedule σ, the path $p = \langle$`idxMill`, `unblock`, `inspect`, `idxSub`$\rangle$ with $s =$ `idxMill` and $t =$ `dieAttach` has the constraint $\sigma(\texttt{dieAttach}, k) \geq \sigma(\texttt{idxMill}, k) + e(\texttt{idxMill}) + e(\texttt{unblock}) + e(\texttt{inspect}) + e(\texttt{idxSub})$. This constraint follows from rewriting the channel constraints between the source, the actors in the path, and the sink. Consequently, the earliest start time of the k-th firing of actor `dieAttach` depends on the sum of the execution times of the actors within the path. We call this sum of execution times the weight of path p.

Definition 6.3 (Path weight). The weight $w(p)$ of a path p with j actors is defined as follows.

$$w(p) = \sum_{i=1}^{j} e(p(i))$$

It follows from the definition that for the empty path ($\langle\rangle$), $w(\langle\rangle) = 0$.

The dependencies represented by the state-space equations apply only to a single iteration of the graph. As a result, we only need to consider token-free paths, as the channel constraints are between the firings of the actors within the same iteration k. Multiple such paths may exist between a given source and sink. For instance, in Figure 4, between s = `idxMill` and t = `dieAttach`, possible token-free paths include: ⟨`idxMill`⟩, ⟨`idxMill`, `unblock`, `inspect`, `idxSub`⟩, ⟨`idxMill`, `unblock`, `inspect`, `block`, `idxMill`⟩, and so on. All token-free paths p provide a constraint $\sigma(\texttt{dieAttach}, k) \geq \sigma(\texttt{idxMill}, k) + w(p)$. This means that the path with the largest weight determines the start time of the k-th firing of actor `dieAttach` relative to the start time of the k-th firing of actor `idxMill`.

Definition 6.4 (Token-free empty paths). An empty path between a source and a sink is token-free if there are no tokens between the source and sink. If a channel (a_1, n, a_2) contains n tokens labelled $x_1, \ldots, x_n$, then a token-free empty path exists between the source token x_{i-1} and the sink token x_i, where $1 < i \leq n$. Furthermore, a token-free empty path exists between token x_n and actor a_2.

For example, the empty path between source x_4 and sink `inspect` in Figure 4 is token-free.

Definition 6.5 (Set of token-free paths). The set $\mathcal{P}(s, t)$ consists of all token-free paths p with the source s and sink t.

Definition 6.6 (Weight of a longest token-free path). Given a set $\mathcal{P}(s, t)$ of paths with source s and sink t, the weight of the longest path is defined as follows.

$$\mathcal{W}(s, t) = \begin{cases} -\infty & \text{if } \mathcal{P}(s, t) = \emptyset \\ \max\limits_{p \in \mathcal{P}(s,t)} w(p) & \text{otherwise} \end{cases}$$

Due to the token-free cycle in Figure 4, there are infinitely many token-free paths between actor `idxMill` and actor `dieAttach`. Since the cycle has a weight of -10, it will not contribute to the path with the largest weight. It follows that a cycle with a positive sum of execution times in an HSDFG reachable from s and from which t can be reached leads to some pairs s, t not having a longest token-free path between them. This is because such a cycle can be traversed repeatedly, increasing the weight indefinitely with each repetition.

From an equational perspective, we can conclude the following. Consider the constraints of the channels forming the token-free cycle from `idxMill` to itself. We can rewrite the constraints as $\sigma_s(\texttt{idxMill}, k) \geq \sigma_s(\texttt{idxMill}, k) + w(p)$, where $w(p)$ represents the weight of the cycle. We get a constraint in which the start time of the actor `idxMill` is related to itself. As a result, no valid schedule can exist for a graph with such a cycle if the weight of the cycle is positive, as it would violate the channel constraints and thus not represent a valid schedule. This leads us to the following proposition.

Proposition 6.7. *An HSDFG has a valid schedule if and only if all its token-free cycles have a non-positive weight.*

An infeasible HSDFG, one that contains a positive-weight cycle, can be made feasible by adjusting the execution times of one or more actors within that cycle. Which actors to modify depends on

the specific use case. For example, consider the die attaching process HSDFG shown in Figure 4. If the execution time of `inspect` is increased to 40, it creates a positive-weight cycle between `idxMill`, `unblock`, `inspect`, and `block`. If the inspection duration cannot be reduced, for instance due to hardware limitations, then the execution time of `idxMill` must be increased. As a result, the execution times of `block` and `unblock` must also be adjusted to maintain the intended lower and upper bounds. Increasing the execution time of `idxMill` results in a reduced throughput, so it might be preferred to avoid this option. Detecting infeasibility of a system is done through the all-pairs longest-path algorithm discussed in Section 9.1.

The dependencies between the vectors in the state-space representation of Definition 4.3 can now be captured with matrices as follows. Given an HSDFG G with m initial tokens and the following labels.

- Initial tokens x_i for $0 < i \leq m$;
- actors a_i for $0 < i \leq |A|$;
- inputs u_i for $0 < i \leq |I|$;
- outputs y_i for $0 < i \leq |O|$.

We set the following values for the state-space matrices.

- Each entry $\mathbf{A}_{i,j}$ of the matrix $\mathbf{A}$ is given the weight $\mathcal{W}(x_j, x_i)$, with $0 < i \leq m$ and $0 < j \leq m$.
- Each entry $\mathbf{B}_{i,j}$ of the matrix $\mathbf{B}$ is given the weight $\mathcal{W}(u_j, x_i)$, with $0 < i \leq m$ and $0 < j \leq |I|$.
- Each entry $\mathbf{C}_{i,j}$ of the matrix $\mathbf{C}$ is given the weight $\mathcal{W}(x_j, y_i)$, with $0 < i \leq |O|$ and $0 < j \leq m$.
- Each entry $\mathbf{D}_{i,j}$ of the matrix $\mathbf{D}$ is given the weight $\mathcal{W}(u_j, y_i)$, with $0 < i \leq |O|$ and $0 < j \leq |I|$.
- Each entry $\mathbf{H}_{i,j}$ of the matrix $\mathbf{H}$ is given the weight $\mathcal{W}(x_j, a_i)$, with $0 < i \leq |A|$ and $0 < j \leq m$.
- Each entry $\mathbf{J}_{i,j}$ of the matrix $\mathbf{J}$ is given the weight $\mathcal{W}(u_j, a_i)$, with $0 < i \leq |A|$ and $0 < j \leq |I|$.

Together, these six matrices define the (max,+) linear system representation of an (extended) HSDFG (see Definition 4.3). They can be computed with an all-pairs longest path algorithm (see Section 9.1).

7 Deriving a valid ASAP state-space schedule

As stated in [9], the (max, +) linear system representation of an HSDFG generates schedules that are both valid and ASAP. With the introduction of a new method for finding the matrices of an HSDFG with token-free cycles in Section 6, we prove in this section that the resulting linear system, formulated with the newly found matrices, also produces valid and ASAP schedules.

We prove that the state-space schedule, as defined in Definition 4.4, is both valid and ASAP for an HSDFG without inputs or outputs. This simplification improves the readability and clarity of the proof, as inputs and outputs impose constraints similar to channel constraints in the HSDFG. A complete proof that includes inputs and outputs is provided in Appendix A.

Theorem 7.1. *If an HSDFG has a valid schedule, then its state-space schedule σ_s is valid.*

Theorem 7.2. *If an HSDFG has a valid schedule, its state-space schedule σ_s is the ASAP schedule.*

The state-space equations from Definition 4.3, without inputs and outputs, simplify to

$$\begin{aligned} \mathbf{x}(k+1) &= \mathbf{A} \otimes \mathbf{x}(k) \\ \mathbf{s}(k) &= \mathbf{H} \otimes \mathbf{x}(k) \end{aligned} \tag{1}$$

The first equation represents the state transition from iteration k to the next iteration $k+1$. If we look for the relation between two iterations of the graph, we can rewrite the state equation as follows.

$$\mathbf{x}(k+2) = \mathbf{A} \otimes \mathbf{x}(k+1) = \mathbf{A} \otimes \big(\mathbf{A} \otimes x(k)\big) = \mathbf{A}^2 \otimes \mathbf{x}(k)$$

This shows that the relation between the state at iteration k and iteration $k+n$ can be expressed as follows.

$$\mathbf{x}(k+n) = \mathbf{A}^n \otimes \mathbf{x}(k) \tag{2}$$

7.1 Proof of Theorem 7.1

Figure 7 visualizes an HSDFG G with m initial tokens and an arbitrary channel $c \in C$ from an actor $a_1 \in A$ to actor $a_2 \in A$ that holds n initial tokens, assumed w.l.o.g. to be labelled x_1 to x_n. The tokens labelled x_{n+1} to x_m reside on other channels in G.

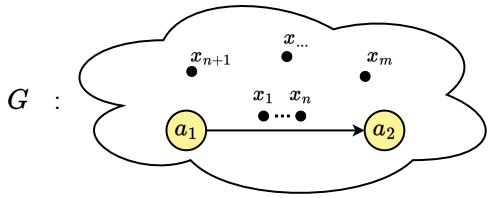

Fig. 7. SDFG G with m initial tokens and a channel (a_1, n, a_2) that holds n tokens.

We prove that the state-space schedule σ_s is valid by showing that it satisfies the constraint associated with this channel, and hence those associated to all channels. This means that the following constraint must hold for the channel c for all $k \in \mathbb{N}$.

$$\sigma_s(a_2, k+n) \geq \sigma_s(a_1, k) \otimes e(a_1)$$

In the remainder, we use notations like $\mathbf{A}_{x_i}$ to represent the row $\mathbf{A}_{i,}$ of the state matrix, which corresponds to the initial token x_i. Similarly, $\mathbf{A}_{x_i,x_j}$ refers to the element $\mathbf{A}_{i,j}$. These notations extend to other matrices. For example, $\mathbf{H}_{a_i}$ represents the row $\mathbf{H}_{i,}$ associated with actor a_i, and $\mathbf{H}_{a_i,x_j}$ represents the element $\mathbf{H}_{i,j}$ corresponding to actor a_i and token x_j.

By applying Definition 4.4, along with Equation 1 and Equation 2, we can express the left-hand side of the above channel constraint as follows:

$$\sigma_s(a_2, k+n) = \mathbf{H}_{a_2} \otimes \mathbf{x}(k+n) = \mathbf{H}_{a_2} \otimes \mathbf{A}^n \otimes \mathbf{x}(k)$$

Similarly, the right-hand side can be rewritten as

$$\sigma_s(a_1, k) \otimes e(a_1) = \mathbf{H}_{a_1} \otimes \mathbf{x}(k) \otimes e(a_1) = (e(a_1) \otimes \mathbf{H}_{a_1}) \otimes \mathbf{x}(k)$$

Thus, we need to show that

$$\mathbf{H}_{a_2} \otimes \mathbf{A}^n \otimes \mathbf{x}(k) \geq (e(a_1) \otimes \mathbf{H}_{a_1}) \otimes \mathbf{x}(k)$$

For this, it suffices to prove that

$$\mathbf{H}_{a_2} \otimes \mathbf{A}^n \geq e(a_1) \otimes \mathbf{H}_{a_1}$$

This is the same as

$$\left[\mathbf{H} \otimes \mathbf{A}^n\right]_{a_2} \geq e(a_1) \otimes \mathbf{H}_{a_1}$$

This constraint states that, for each token x_i with $1 \leq i \leq m$, the weight of a longest path from x_i to a_2 passing through n tokens is at least equal to the weight of a longest path from x_i to a_1 plus the execution time of a_1. So, the following constraint must hold for all x_i.

$$\left[\mathbf{H} \otimes \mathbf{A}^n\right]_{a_2,x_i} \geq e(a_1) \otimes \mathbf{H}_{a_1,x_i} \tag{3}$$

If there is no path from token x_i to actor a_1, then $\mathbf{H}_{a_1,x_i} = -\infty$, and the constraint trivially holds. Similarly, if there is no path from token x_i to actor a_2 that passes n tokens, then there cannot be a token-free path from x_i to a_1, since it connects a_2 via the channel c with precisely n tokens. Thus, in that case both $\left[\mathbf{H} \otimes \mathbf{A}^n\right]_{a_2,x_i} = -\infty$ and $\mathbf{H}_{a_1,x_i} = -\infty$, which means that the constraint holds. We proceed with the proof, assuming both paths exist.

Figure 8 shows the multiplication $\mathbf{H} \otimes \mathbf{A}^n$. Each entry in $\mathbf{A}$ represents the weight of a longest path from some token (corresponding to the column) to another token (corresponding to the row). Similarly, the entries in $\mathbf{H}$ represent the weights of a longest path from a token to an actor of G. Recall that the longest paths underlying these matrices are token-free. The arrows in the figure illustrate some of these longest paths, with the path weight indicated on each edge. In the figure, two possible paths are shown for the entry $\left[\mathbf{H} \otimes \mathbf{A}^n\right]_{a_2,x_i}$. The first path (the upper one in the figure) starts at token x_i and passes through tokens x_1 to x_n, eventually reaching actor a_2. It can be observed that the longest path from x_i to x_1 is the longest path from token x_i to actor a_1, with weight $\mathbf{H}_{a_1,x_i}$, passing through a_1 which adds the execution time $e(a_1)$ to the weight of the path. Since the tokens x_1 to x_n are all on the same channel c, there exists a single path between consecutive tokens x_j and x_{j+1}, with weight 0 for all j such that $1 \leq j < n$. Additionally, the entry $\mathbf{H}_{a_2,x_n}$ must also be 0, since the only path connecting token x_n to actor a_2 is empty. This makes the total weight of the path to be $\mathbf{H}_{a_1,x_i} \otimes e(a_1)$. The second path represents a longest path from x_i passing through n tokens and eventually reaching actor a_2 and hence has a length of $\left[\mathbf{H} \otimes \mathbf{A}^n\right]_{a_2,x_i}$. Note that this second path could be equal to the first path. This shows that the constraint $\left[\mathbf{H} \otimes \mathbf{A}^n\right]_{a_2,x_i} \geq e(a_1) \otimes \mathbf{H}_{a_1,x_i}$ holds, as either the longest path from x_i to a_2 is the first path, or any other path with a larger weight going n times through the remaining tokens in G.

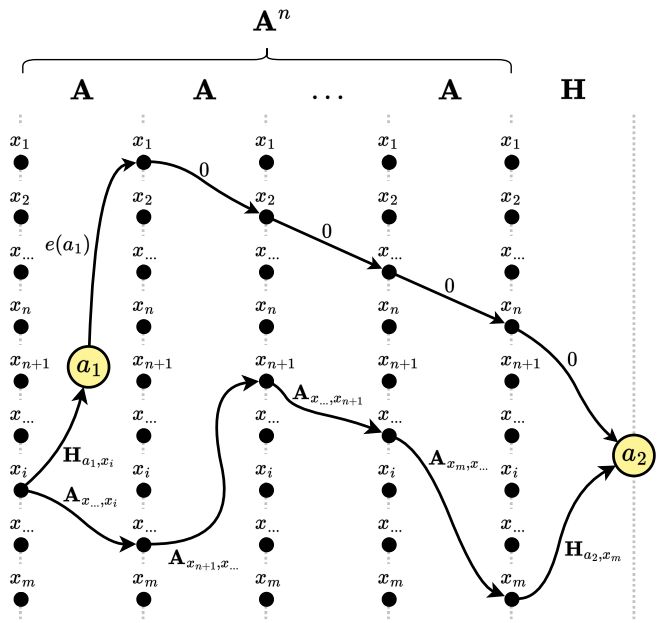

Fig. 8. Visualising the $\mathbf{H} \otimes \mathbf{A}^n$ multiplication, with the arrows representing the longest paths between multiplications.

7.2 Proof of Theorem 7.2

Theorem 7.2 states the valid state-space schedule σ_s is the ASAP schedule, which means that $\sigma_s \leq \sigma$ holds for any other valid schedule σ. Let G be an HSDFG with m initial tokens, a state-space schedule σ_s, and a valid schedule σ. The sequence of state vectors corresponding to the tokens used to compute σ_s is referred to as $\mathbf{x}^{\sigma_s}$. Since σ is not derived using state-space matrices, its corresponding state vector sequence $\mathbf{x}^{\sigma}$ naturally follows from the token timestamps in schedule σ.

The constraint $\sigma_s \leq \sigma$ implies that no valid schedule σ can exist such that, for some $a_1 \in A$ and $k \in \mathbb{N}$, the constraint $\sigma(a_1, k) < \sigma_s(a_1, k)$ holds. Suppose, towards a contradiction, that there exists a valid schedule σ where $\sigma(a_1, k) < \sigma_s(a_1, k)$ does hold. We assume w.l.o.g. that k is the smallest k for which this holds for some actor a_i. I.e., for any $l < k$ and for all $a_i \in A$, we have $\sigma(a_i, l) \geq \sigma_s(a_i, l)$.

Let x_w with $1 \leq w \leq m$ be a token in G for which the following condition holds.

$$\mathbf{H}_{a_1} \otimes \mathbf{x}^{\sigma_s}(k) = \bigoplus_{j=1}^{m} \mathbf{H}_{a_1,x_j} \otimes \mathbf{x}^{\sigma_s}(k)[x_j] = \mathbf{H}_{a_1,x_w} \otimes \mathbf{x}^{\sigma_s}(k)[x_w]$$

This means that the start time of actor a_1 in σ_s is determined by the weight $\mathbf{H}_{a_1,x_w}$ and the timestamp of token x_w at iteration k, which corresponds to the entry $\mathbf{x}^{\sigma_s}(k)[x_w]$ in the state vector of iteration k. Thus,

$$\sigma_s(a_1, k) = \mathbf{H}_{a_1,x_w} \otimes \mathbf{x}^{\sigma_s}(k)[x_w]$$

Since σ is a valid schedule, it must satisfy the constraint imposed by the longest path from token x_w to actor a_1, leading to

$$\sigma(a_1, k) \geq \mathbf{H}_{a_1,x_w} \otimes \mathbf{x}^{\sigma}(k)[x_w]$$

Furthermore, we know that for all actors a_i and all $l < k$, $\sigma(a_i, l) \geq \sigma_s(a_i, l)$. As a result, once all actors complete their execution, the token timestamps for iteration $l + 1$ are computed, ensuring that

$$\mathbf{x}^{\sigma}(l + 1) \geq \mathbf{x}^{\sigma_s}(l + 1)$$

for $l = k - 1$, we get that

$$\mathbf{x}^{\sigma}(k) \geq \mathbf{x}^{\sigma_s}(k) \tag{4}$$

Which means that

$$\mathbf{H}_{a_1,x_w} \otimes \mathbf{x}^{\sigma}(k)[x_w] \geq \mathbf{H}_{a_1,x_w} \otimes \mathbf{x}^{\sigma_s}(k)[x_w]$$

and thus

$$\sigma(a_1, k) \geq \sigma_s(a_1, k)$$

This contradicts with our assumption that $\sigma(a_1, k) < \sigma_s(a_1, k)$. Therefore, the schedule σ cannot be valid, and we conclude that σ_s is the ASAP schedule.

The proof is illustrated with the HSDFG shown in Figure 4. (We disregard the input `die` and output `sub`.) Figure 9 shows the ASAP state-space schedule σ_s and some schedule σ of three iterations of the HSDFG. Schedule σ is such that the start time of actor `dieAttach` in the iteration $k = 2$ is smaller than the same firing in schedule σ_s, i.e., $\sigma(\texttt{dieAttach}, 2) < \sigma_s(\texttt{dieAttach}, 2)$. For all k smaller than 2, the start times of all actor firings in σ are at least equal to actor start times in σ_s. This means that $\mathbf{x}^{\sigma}(2)[x_3] \geq \mathbf{x}^{\sigma_s}(2)[x_3]$. The longest path from token x_3 to the start of actor `dieAttach` has weight $\mathbf{H}_{\texttt{dieAttach},x_3}$, and thus $\mathbf{H}_{\texttt{dieAttach},x_3} \otimes \mathbf{x}^{\sigma}(2)[x_3] \geq \mathbf{H}_{\texttt{dieAttach},x_3} \otimes \mathbf{x}^{\sigma_s}(2)[x_3]$. Both these paths are drawn with the red arrows. These paths should enforce the constraint $\sigma(\texttt{dieAttach}, 2) \geq \sigma_s(\texttt{dieAttach}, 2)$. From the figure, we can see though that $\sigma(\texttt{dieAttach}, 2) < \mathbf{H}_{\texttt{dieAttach},x_3} \otimes \mathbf{x}^{\sigma}(2)[x_3]$ and that the constraint $\sigma(\texttt{dieAttach}, 2) \geq \sigma_s(\texttt{dieAttach}, 2)$ is violated by schedule σ because $220 \not\geq 230$. Thus, σ is not a valid schedule.

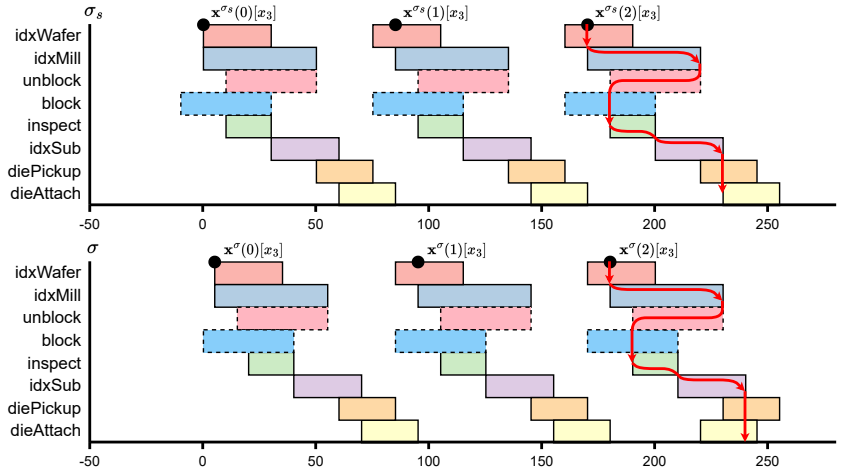

Fig. 9. The state-space schedule σ_s and a presumably valid schedule σ of the HSDFG shown in Figure 4. The sequence of red arrows represents the longest path from token x_3 to actor `dieAttach`.

8 Manufacturing use cases

In this section, we discuss three manufacturing use cases, highlighting the need for defining lower and upper bounds between machine tasks. We first complete our model of the die attach process. Then, we examine two use cases from literature. The first involves a lacquer production machine, where bounds between machine actions are defined similarly to those in the die attach process. The second use case is based on a job-shop scheduling problem, in which we analyse the processing of job batches that contain lower and upper bounds.

8.1 Die attaching process

Recall that the die attach process consists of three key steps: Dispense, Die attaching, and Cure. We already introduced the Die attaching step and its tasks in Figure 1. The Dispense step consists of three tasks: dispensing the paste (`dispPaste`), inspecting the placement of the paste droplet (`dispInspect`), and advancing the substrate holder (`dispIdxSub`). Similarly, the Cure step involves curing the paste (`cure`) by heating it until it hardens, followed by indexing the substrate holder forward (`cureIdxSub`). The complete HSDFG of the die attaching process is shown in Figure 10. The HSDFG includes two inputs and one output: the `emptySub` input represents the arrival of empty substrates, the `die` input models the arrival of dies, and the `sub` output represents the assembled substrates exiting the machine.

In the die attaching process, the substrate holder is a solid unit, which means that the actors `dispIdxSub`, `attachIdxSub`, and `cureIdxSub` must start and complete at the same time. So, for a valid schedule σ that the HSDFG represents, we model this with the following constraints.

$$\sigma(\texttt{dispIdxSub}, k) = \sigma(\texttt{attachIdxSub}, k) = \sigma(\texttt{cureIdxSub}, k)$$

These can be written as the following constraints.

$$\begin{aligned}
\sigma(\texttt{dispIdxSub}, k) &\geq \sigma(\texttt{attachIdxSub}, k) \\
\sigma(\texttt{dispIdxSub}, k) &\leq \sigma(\texttt{attachIdxSub}, k) \\
\sigma(\texttt{attachIdxSub}, k) &\geq \sigma(\texttt{cureIdxSub}, k) \\
\sigma(\texttt{attachIdxSub}, k) &\leq \sigma(\texttt{cureIdxSub}, k)
\end{aligned}$$

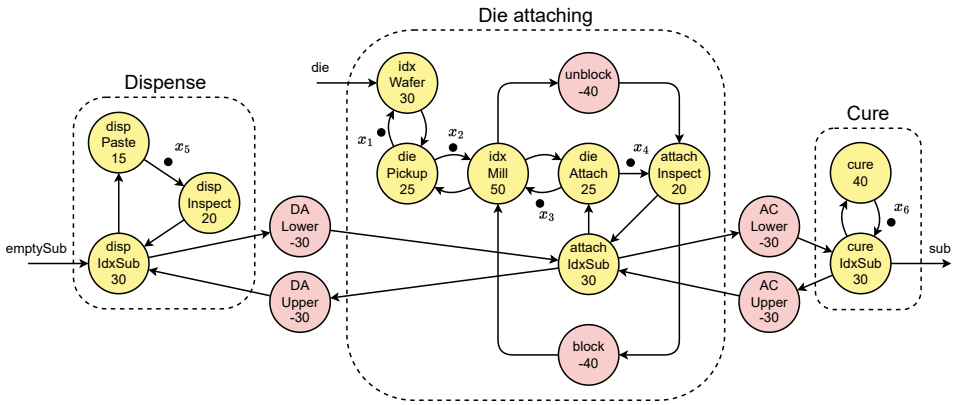

Fig. 10. HSDFG of the die attaching process, comprising a *Dispense*, *Die attaching*, and *Cure* step.

Transforming these constraints so they are represented in the HSDFG results in the actors `DALower`, `DAUpper`, `ACLower`, and `ACUpper` and their corresponding channels, already included in Figure 10. Figure 11 shows the ASAP schedule of the die attach process HSDFG of Figure 10. For clarity, negatively-timed actors have been omitted. Clearly, the bounds are respected as the start times of `dispIdxSub`, `attachIdxSub`, and `cureIdxSub` are the same for each iteration of the graph.

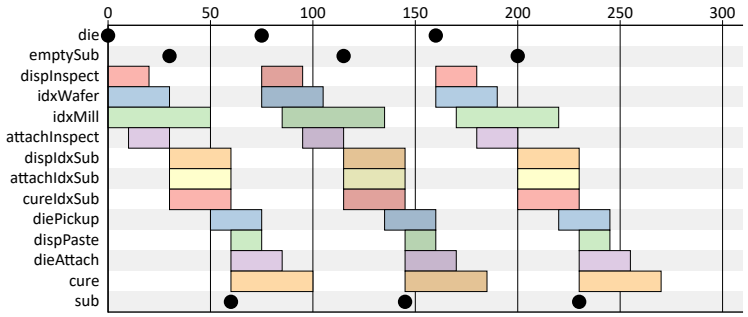

Fig. 11. The state-space schedule of the die attach process HSDFG shown in Figure 10.

8.2 Lacquer production

The lacquer production process described in [3] involves manufacturing various types of lacquers, each based on a specific recipe. A recipe specifies the necessary resources and processing times for each step. Between these steps, synchronization bars and lower and upper bound intervals are specified. Figure 12 presents the production of a lacquer vat following the recipe of Figure 1 in [3], modelled as an HSDFG after scheduling decisions have been applied. In this graph, the yellow actors represent individual steps in the lacquer production recipe. Each yellow actor has a self-loop channel with a token, not shown in the figure, ensuring a single resource claim per step. Shared resource usage is modelled using channels with initial tokens between actors. For instance, `doseSpinner1` and `doseSpinner2` share a resource, as do `lab1` and `lab2`. Gray actors represent synchronization between recipe steps. The red actors represent the timing bounds between the recipe steps and the synchronization actors. For example, Figure 1 in [3] specifies a bound of $[26, 30]$ between the start of the `dispenser` actor and the start of the `sync1` actor. This bound is

modelled using a direct channel from `dispenser` to `sync1`, along with two additional channels connected with an upper-bound actor, `dispUpper`, from `sync1` back to `dispenser`. This translation is consistent with the method shown in Figure 6.

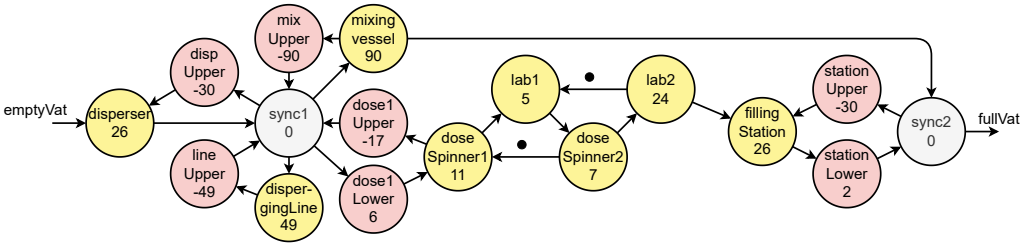

Fig. 12. An HSDFG modelling the recipe of Figure 1 of [3] for producing a specific kind of lacquer.

Figure 13 shows the schedule of producing three lacquer vats following the recipe described with the HSDFG of Figure 12. We omitted the negatively-timed actors for clarity. The machine is given three empty vats at time zero, which is modelled with the token arrivals on the input `emptyVat`. We can see the effects of the upper bounds in the second and third activations of `dispenser`, which start 30 time units before the second and third activation of `sync1`, respectively. Furthermore, the bound between `sync1` and `doseSpinner1` ensures that the activation of `doseSpinner1` is exactly six time units after the activation of `sync1`. The activation of `fillingStation` is ensured to be 30 time units before the start of `sync2`.

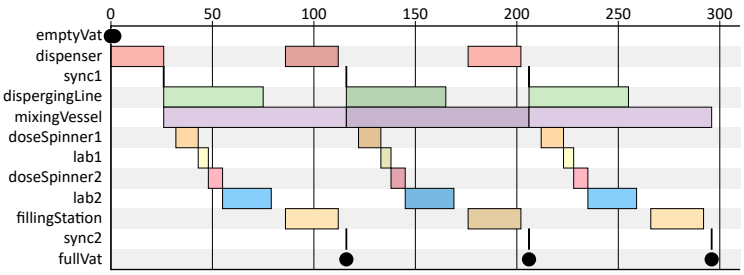

Fig. 13. A schedule of producing three lacquer vats following the recipe described with the HSDFG of Figure 12.

8.3 Batching in job-shop scheduling

In factories such as the lacquer production described in Section 8.2, orders are handled in batches. Each batch corresponds to a request for a specific quantity of a single type of lacquer. When multiple batch requests are received, they must be scheduled for production using the available machines in the factory. This scheduling challenge can be modelled as a job-shop scheduling problem with batches, as described in [5]. In the context of the lacquer use case, a job represents the production of one vat of a specific type of lacquer, consisting of a fixed sequence of operations. A batch corresponds to a specific number of vats. All jobs within the same batch follow the same operation sequence. Solving the job-shop scheduling problem following [5] results in a solution that specifies how jobs are allocated to machines. A schedule is then derived by starting the processing as soon as the required machine becomes available. However, as shown for jobs such as in lacquer production,

there are upper bounds between operations. As a result, the above-described activation may not produce correct behaviour. In the following example, we demonstrate how the job-to-machine allocation for a batch can be modelled using an HSDFG. This allows us to synthesize an ASAP schedule that respects the timing constraints of batches for a given allocation.

Consider a batch $b = \{j_1, ..., j_8\}$ consisting of eight jobs, each producing a single lacquer vat. For simplicity, assume each job consists of two operations, op_1 and op_2, with both a lower and upper bound between them: the end time of op_1 must equal the start time of op_2. There are three machines available: m_1, m_2, and m_3. Machine m_1 and m_2 both perform operation op_1, while only machine m_3 is capable of performing op_2. The processing time for op_1 is 2 time units on machine m_1 and 3 time units on m_2. Operation op_2 takes 1 time unit on m_3. Suppose the job-shop scheduling algorithm from [5] provides the following machine allocation: jobs j_1, j_3, j_5, and j_7 perform op_1 on m_1, while j_2, j_4, j_6, and j_8 use m_2. All op_2 operations are allocated to m_3.

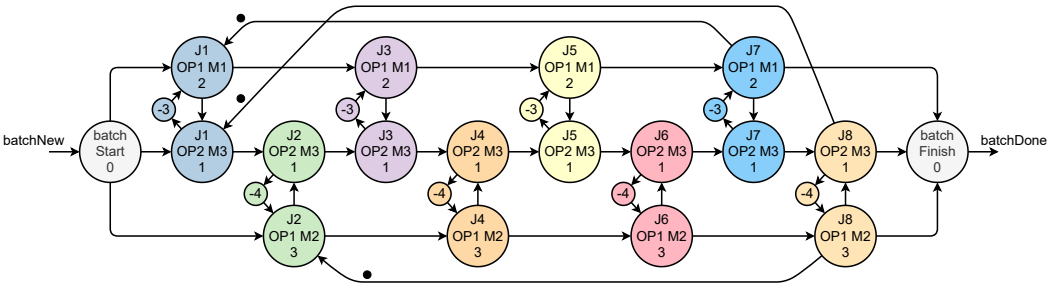

Fig. 14. Batch b with machine allocation modelled as an HSDFG.

The machine allocation can be modelled using the HSDFG shown in Figure 14. Such an HSDFG can be automatically constructed from any given machine allocation and job sequence. Each operation is modelled as an actor labelled with its job, operation, machine, and processing time. For example, "J3 OP1 M1 2" denotes job j_3, operation op_1, scheduled on machine m_1 with a processing time of 2 time units. The input `batchNew` models the arrival of a request to process the batch. Upon this request, the first three operations that are allocated to their respective machines can activate, which is modelled via the `batchStart` actor. The HSDFG is structured such that all actors corresponding to the same machine and all actors corresponding to the same job are connected with a channel. This ensures that only one stage of any given job can be processed on a machine at any given time, and that the job operations are processed consecutively. To enforce that each op_2 begins immediately after the corresponding op_1 completes, upper-bound actors are inserted between two operations. These actors are labelled only with their execution time for clarity and readability. The `batchFinish` actor marks the completion of the batch, which is also modelled with the output `batchDone`. Figure 15 shows the ASAP schedule of one batch requested at time 0. The `bounds` row shows the activations of the unlabelled actors with negative execution times. The colours used in the schedule match the actor colours in Figure 14.

The HSDFG can now be used for efficiently synthesizing schedules of multiple batches. For example, consider a scenario where customers place orders for three batches. For simplicity of the example, we assume that all orders are for batch b discussed above. Two batches are requested at time 0, while the third is requested at time 30. An ASAP schedule for processing the three batches can be synthesized from the batch HSDFGs (three times the HSDFG of Figure 14 in this case). Figure 16 shows the resulting schedule for processing these three batches, with all jobs within the same batch highlighted in the same colour.

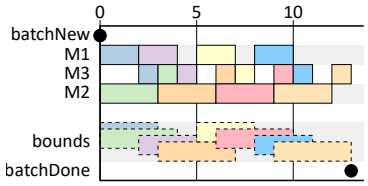

Fig. 15. The schedule of processing a single batch b synthesized from the HSDFG of Figure 14, with colours matching those used in the HSDFG.

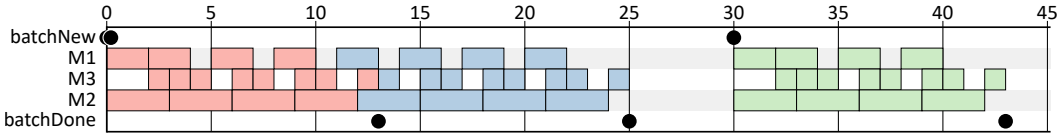

Fig. 16. An example illustrating the efficient analysis of processing three batches b at release times 0, 0, and 30.

This schedule is synthesized through three vector matrix multiplications using the max-plus linear state-space representation of the HSDFG. The schedule shows that most jobs are constrained by their upper bounds. Notably, the makespan for the first two batches is 25 time units instead of 26, due to overlap between the batches, while still respecting the timing constraints. The sketched approach can be extended to scenarios where different batches share common resources. Matrix multiplications can be performed across those different batches while preserving the timing constraints specific to each scenario.

9 Evaluation

In this section, we discuss the algorithmic complexity of the all-pairs longest-path algorithm used to find the (max, +) linear state-space matrices for HSDFGs that include negatively-timed actors and token-free cycles. For traditional HSDFGs, we experimentally compare the performance of our method to symbolic simulation. We conclude with a discussion on the practical implications of applying the methods discussed in industrial contexts.

9.1 Algorithm complexity

In [22], an all-pairs shortest-path algorithm is presented for computing the minimum-weight paths between all pairs of vertices in a directed graph. The algorithm operates on a weighted adjacency matrix, where each entry represents the weight of an edge from one vertex to another. If no edge exists between two vertices, the corresponding matrix entry is set to infinity. The algorithm supports graphs with negative edge weights and positive-weight cycles. It iteratively updates the path lengths by looping three times over all vertices, applying a (min, +) operation at the core of the loop.

The algorithm can be adapted to compute the longest paths between all sources and all sinks in an HSDFG. In this case, the input is an adjacency matrix where each entry value is the execution time of the source actor of a token-free channel. If no token-free channel exists between two actors, the entry is set to $-\infty$. The algorithm again loops three times over all actors A of the HSDFG, but instead uses a (max, +) operation. The resulting matrix captures the token-free longest paths between all pairs of actors in the HSDFG. From this matrix, which contains all actor-to-actor paths, the state-space matrices of the corresponding linear system can be derived. If the matrix contains

a positive value for a path from an actor to itself, it means that the HSDFGs contains a positive cycle without tokens, making the resulting linear system infeasible. Since the algorithm involves a triple nested loop over all actors, its time complexity is $O(|A|^3)$, where A is the set of actors in the HSDFG. Notably, the presence of negatively-timed actors or token-free cycles does not impact the runtime performance of the algorithm.

The symbolic simulation algorithm for traditional (H)SDFGs [9] described in Section 4.3 has a similar cubic complexity. In this algorithm, for each actor in the HSDFG, the element-wise maximum of the symbolic timestamp vectors from all incoming channels is computed. The length of each symbolic timestamp vector corresponds to the number of initial tokens in the graph. This means that the overall time complexity of the algorithm is $O(|A| \times |C| \times m)$, where A is the set of actors, C the set of channels, and m the number of initial tokens in the HSDFG.

Figure 17 compares the runtime performance of both the all-pairs longest-path algorithm and the symbolic simulation algorithm on a shared benchmark. The benchmark excludes HSDFGs with token-free cycles, as the symbolic simulation algorithm is not applicable in such cases. A separate benchmark for graphs with token-free cycles is unnecessary, since, as previously discussed, these cycles do not affect the runtime of the longest path algorithm. The figure shows results for graphs with up to 240 actors, which is significantly larger than those typically found in practical applications. Each data point is the average run time of 30 HSDFGs with the given number of actors and average actor degrees (deg). The token percentage (tok) indicates the proportion of channels that contain initial tokens. Both algorithms were executed on a standard laptop equipped with an Intel(R) Core(TM) i7-10610U CPU and 32 GB of RAM.

The results align with the computational complexities of the two algorithms, which show similar practical runtimes. The all-pairs longest-path algorithm shows a moderate increase in runtime as the degree and number of tokens grow, mainly due to the overhead of constructing the final linear state-space matrices from the actor-to-actor path matrix. For small graphs, the all-pairs longest-path algorithm performs faster. However, as the graph size increases while the token count and average degree remain low, symbolic simulation proves more efficient. It is only when both the degree and token count of large graphs become high that the all-pairs longest-path algorithm becomes the preferred option. Again, when token-free cycles are present, such as in the use cases discussed in Section 8, only the all-pairs longest-path algorithm can be used.

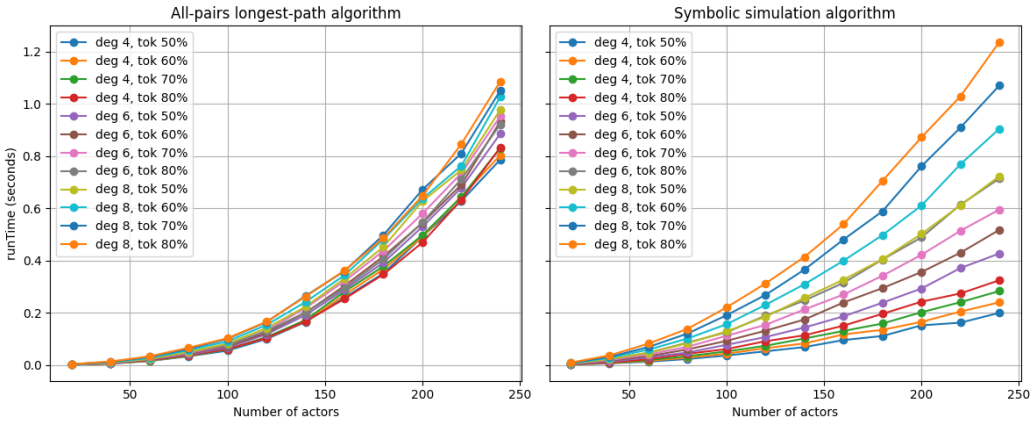

Fig. 17. Runtime comparison of the all-pairs longest-path algorithm and the symbolic simulation algorithm across varying numbers of actors, average actor degrees, and token percentages.

9.2 Practical implications

A common approach for modelling manufacturing systems in practice is through the use of a Domain Specific Language (DSL), such as the Logistics Specification and Analysis Tool (LSAT) [27]. Such a tool integrates the domain knowledge of a manufacturing company, enabling engineers to model resources, system behaviour, and timing characteristics they are already familiar with. LSAT focuses primarily on performance analysis. The models created within the DSL are translated into a mathematical model for analysis. In the case of LSAT, this model is a (max, +) linear system.

A similar approach can be followed for the extended HSDFGs introduced in this paper. A DSL could be developed for a specific manufacturing domain, for instance, as an extension to the LSAT DSL. These DSL-based models would then be translated into HSDFGs, which may serve as the underlying formal mathematical model for performance analysis and synthesis of ASAP schedules.

In a (job-shop) scheduling context, as discussed in Section 8.3, an HSDFG as shown in Figure 14 can be derived from a machine allocation and job-operation sequences per machine. ASAP batch processing schedules can then be constructed using (max, +) matrix-vector multiplication. This provides another example of using an HSDFG as a formal back-end for ASAP-schedule synthesis.

10 Conclusion

We introduced a novel transformation on HSDFGs to represent lower and upper bounds on timing differences between actor firings. This transformation results in token-free cycles within the graph with actors with negative execution times. We introduced a new method to convert such an HSDFG into a (max, +) linear system, enabling the synthesis of a valid ASAP schedule. This schedule ensures the highest possible productivity of a modelled system while adhering to the defined lower and upper bounds.

This work lays the foundation for further research. It is interesting to extend this work on incorporating bounds in HSDFGs to SDFGs. Additionally, we are interested in capturing the dynamic behaviour of manufacturing systems using synchronous dataflow scenarios while maintaining these bounds. Another topic is enhancing the model with negative tokens to define lower and upper bounds across graph iterations.

References

[1] Hadi Alizadeh Ara, Amir Behrouzian, Marc Geilen, Martijn Hendriks, Dip Goswami, and Twan Basten. 2016. Analysis and visualization of execution traces of dataflow applications. In *Integrating Dataflow, Embedded Computing, and Architecture* (Vienna, Austria) *(Report ESR-2017-01, Eindhoven University of Technology)*. 19–20.

[2] François Baccelli, Guy Cohen, Geert Jan Olsder, and Jean-Pierre Quadrat. 1994. Synchronization and linearity - an algebra for discrete event systems. *The Journal of the Operational Research Society* 45 (01 1994). https://doi.org/10.2307/2583959

[3] Gerd Behrmann, Ed Brinksma, Martijn Hendriks, and Angelika Mader. 2005. Scheduling lacquer production by reachability analysis – a case study. *IFAC Proceedings Volumes* 38, 1 (Jan. 2005), 50–55. https://doi.org/10.3182/20050703-6-CZ-1902.01433

[4] Johan Bengtsson, Kim Larsen, Fredrik Larsson, Paul Pettersson, and Wang Yi. 1996. UPPAAL — a tool suite for automatic verification of real-time systems. In *Hybrid Systems III*, Rajeev Alur, Thomas A. Henzinger, and Eduardo D. Sontag (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 232–243. https://doi.org/10.1007/BFb0020949

[5] Márton Drótos, Gábor Erdős, and Tamás Kis. 2009. Computing lower and upper bounds for a large-scale industrial job shop scheduling problem. *European Journal of Operational Research* 197, 1 (Aug. 2009), 296–306. https://doi.org/10.1016/j.ejor.2008.06.004

[6] Didier Dubois and Kathryn E. Stecke. 1983. Using Petri nets to represent production processes. In *The 22nd IEEE Conference on Decision and Control* (San Antonio, TX, USA). IEEE, 1062–1067. https://doi.org/10.1109/CDC.1983.269683

[7] Electronic Systems Group, Department of Electrical Engineering, Eindhoven University of Technology. 2025. Model-Based Design Lab. https://computationalmodeling.info

[8] Electronic Systems Group, Department of Electrical Engineering, Eindhoven University of Technology. 2025. SDF3. https://github.com/TUE-EE-ES/SDF3/

[9] Marc Geilen. 2011. Synchronous dataflow scenarios. *ACM Transactions on Embedded Computing Systems* 10, 2, Article 16 (Jan. 2011), 31 pages. https://doi.org/10.1145/1880050.1880052

[10] Marc Geilen. 2018. If we could go back in time... on the use of 'unnatural' time and ordering in dataflow models. In *Principles of modeling: essays dedicated to Edward A. Lee on the occasion of his 60th birthday*, Marten Lohstroh, Patricia Derler, and Marjan Sirjani (Eds.). Springer International Publishing, Cham, 267–286. https://doi.org/10.1007/978-3-319-95246-8_16

[11] Marc Geilen, Mladen Skelin, J. Reinier van Kampenhout, Hadi Alizadeh Ara, Twan Basten, Sander Stuijk, and Kees G.W. Goossens. 2020. Scenarios in dataflow modeling and analysis. In *System-Scenario-based Design Principles and Applications*. Springer International Publishing, Cham, 145–180. https://doi.org/10.1007/978-3-030-20343-6_8

[12] Amir H. Ghamarian, Marc Geilen, Sander Stuijk, Twan Basten, Bart Theelen, Mohammad R. Mousavi, Arno Moonen, and Marco J.G. Bekooij. 2006. Throughput analysis of synchronous data flow graphs. In *Sixth International Conference on Application of Concurrency to System Design* (Turku, Finland). IEEE, 25–36. https://doi.org/10.1109/ACSD.2006.33

[13] Martijn Hendriks, Jacques Verriet, Twan Basten, Bart Theelen, Marco Brassé, and Lou Somers. 2017. Analyzing execution traces: critical-path analysis and distance analysis. *International Journal on Software Tools for Technology Transfer* 19, 4 (Aug. 2017), 487–510. https://doi.org/10.1007/s10009-016-0436-z

[14] Larry E. Holloway, Bruce H. Krogh, and Alessandro Giua. 1997. A survey of Petri net methods for controlled discrete event systems. *Discrete Event Dynamic Systems* 7, 2 (April 1997), 151–190. https://doi.org/10.1023/A:1008271916548

[15] IBM. 2024. ILOG CPLEX optimization studio. https://www.ibm.com/products/ilog-cplex-optimization-studio

[16] Anant Singh Jain and Sheik Meeran. 1999. Deterministic job-shop scheduling: past, present and future. *European Journal of Operational Research* 113, 2 (March 1999), 390–434. https://doi.org/10.1016/S0377-2217(98)00113-1

[17] Edward A. Lee. 2009. Computing needs time. *Commun. ACM* 52, 5 (May 2009), 70–79. https://doi.org/10.1145/1506409.1506426

[18] Edward A. Lee and David G. Messerschmitt. 1987. Static scheduling of synchronous data flow programs for digital signal processing. *IEEE Trans. Comput.* C-36, 1 (1987), 24–35. https://doi.org/10.1109/TC.1987.5009446

[19] Yuh-Zen Liao and C.K. Wong. 1983. An algorithm to compact a VLSI symbolic layout with mixed constraints. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 2, 2 (1983), 62–69. https://doi.org/10.1109/TCAD.1983.1270022

[20] Tadao Murata. 1989. Petri nets: properties, analysis and applications. *Proc. IEEE* 77, 4 (1989), 541–580. https://doi.org/10.1109/5.24143

[21] Guillaume Roumage, Selma Azaiez, and Stéphane Louise. 2022. A survey of main dataflow MoCCs for CPS design and verification. In *2022 IEEE 15th International Symposium on Embedded Multicore/Many-core Systems-on-Chip*. IEEE, 1–9. https://doi.org/10.1109/MCSoC57363.2022.00010

[22] Sundararajan Sriram and Shuvra Bhattacharyya. 2009. *Embedded multiprocessors: scheduling and synchronization, Second Edition.* CRC Press, Inc., USA. 1–361 pages. https://doi.org/10.1201/9781420048025

[23] Sander Stuijk, Marc Geilen, and Twan Basten. 2006. SDF^3: SDF for free. In *Sixth International Conference on Application of Concurrency to System Design* (Turku, Finland). IEEE, 276–278. https://doi.org/10.1109/ACSD.2006.23

[24] Shinji Takeda and Takashi Masuko. 2009. Die attach adhesives and films. In *Materials for Advanced Packaging*, Daniel Lu and C.P. Wong (Eds.). Springer US, Boston, MA, 407–436. https://doi.org/10.1007/978-0-387-78219-5_12

[25] Jeffrey J.P. Tsai, S. Jennhwa Yang, and Yao-Hsiung Chang. 1995. Timing constraint Petri nets and their application to schedulability analysis of real-time system specifications. *IEEE Transactions on Software Engineering* 21, 1 (1995), 32–49. https://doi.org/10.1109/32.341845

[26] Uppsala Universitet and Aalborg Univerity. 2025. UPPAAL. https://uppaal.org/

[27] Bram van der Sanden, Yuri Blankenstein, Ramon Schiffelers, and Jeroen Voeten. 2021. LSAT: specification and analysis of product logistics in flexible manufacturing systems. In *IEEE 17th International Conference on Automation Science and Engineering* (Lyon, France). IEEE, 1–8. https://doi.org/10.1109/CASE49439.2021.9551412

[28] Umar Waqas, Marc Geilen, Jack Kandelaars, Lou Somers, Twan Basten, Sander Stuijk, Patrick Vestjens, and Henk Corporaal. 2015. A re-entrant flowshop heuristic for online scheduling of the paper path in a large scale printer. In *2015 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 573–578. https://doi.org/10.7873/DATE.2015.0519

A Proof of valid and ASAP state-space schedule

In this appendix, we extend the proofs for Theorems 7.1 and 7.2 to show that a schedule synthesized via a (max, +) linear system, constructed using the proposed all-pairs longest-path method for determining the system matrices, results in a valid ASAP schedule. We recap the theorems from Section 7, extending them to incorporate input arrivals α_s. We show that, for given input arrivals α_s, these two theorems hold. It is not necessary to include outputs in the proofs, as they do not affect actor starting times, as defined in Definition 4.2.

Theorem 7.1

If an HSDFG has a valid schedule for given input arrivals α_s, then its state-space schedule σ_s is valid.

Theorem 7.2

If an HSDFG has a valid schedule for given input arrivals α_s, its state-space schedule σ_s is the ASAP schedule for these input arrivals.

A.1 Proof validity state-space schedule

The state-space schedule σ_s with input arrivals α_s is a valid schedule if it adheres to the constraints given in Definition 4.2. We label them for reference purposes.

Channel validity: For all $(a_1, n, a_2) \in C$, $\sigma_s(a_2, k+n) \geq \sigma_s(a_1, k) + e(a_1)$ for every $k \in \mathbb{N}$

Input validity: For all $u \in I$ with $d_I(u) = a$, $\sigma_s(a, k) \geq \alpha_s(u, k)$ for every $k \in \mathbb{N}$

Proof. Channel validity: Consider an HSDFG G with m tokens and state-space schedule σ_s. We consider a channel $(a_1, n, a_2) \in C$ of G connecting actors $a_1 \in A$ and $a_2 \in A$. The n tokens on this channel are assumed w.l.o.g. to be labelled x_1 to x_n. The tokens labelled x_{n+1} to x_m reside on other channels in G. For channel (a_1, n, a_2), the following constraint must hold.

$$\sigma_s(a_2, k+n) \geq \sigma_s(a_1, k) \otimes e(a_1)$$

For brevity, in the following derivations, (max,+) multiplications $\otimes$ may be left implicit.

Definition 4.3 introduces the following state-space equation

$$\mathbf{x}(k+1) = \mathbf{A}\mathbf{x}(k) \oplus \mathbf{B}\mathbf{u}(k)$$

Rewriting this for a second iteration results in

$$\begin{aligned}\mathbf{x}(k+2) &= \mathbf{A}\mathbf{x}(k+1) \oplus \mathbf{B}\mathbf{u}(k+1)\\ &= \mathbf{A}(\mathbf{A}\mathbf{x}(k) \oplus \mathbf{B}\mathbf{u}(k)) \oplus \mathbf{B}\mathbf{u}(k+1) = \mathbf{A}^2\mathbf{x}(k) \oplus \mathbf{A}\mathbf{B}\mathbf{u}(k) \oplus \mathbf{B}\mathbf{u}(k+1)\end{aligned}$$

Similarly, in general, for the state relation between iteration $k+n$ and k, we have that

$$\mathbf{x}(k+n) = \mathbf{A}^n\mathbf{x}(k) \oplus \bigoplus_{v=1}^{n} \mathbf{A}^{n-v}\mathbf{B}\mathbf{u}(k+v-1)$$

The first term characterizes the dependency on the starting state of the n iterations and the second term the dependency on the inputs during the iterations. Using this equation, we can write the left

side of the channel constraint as

$$\begin{aligned}\sigma_s(a_2, k+n) &= \mathbf{H}_{a_2}\mathbf{x}(k+n) \oplus \mathbf{J}_{a_2}\mathbf{u}(k+n)\\ &= \mathbf{H}_{a_2}\mathbf{A}^n\mathbf{x}(k) \oplus \bigoplus_{v=1}^{n} \mathbf{H}_{a_2}\mathbf{A}^{n-v}\mathbf{B}\mathbf{u}(k+v-1) \oplus \mathbf{J}_{a_2}\mathbf{u}(k+n)\\ &= \left[\mathbf{H}\mathbf{A}^n\right]_{a_2}\mathbf{x}(k) \oplus \bigoplus_{v=1}^{n} \left[\mathbf{H}\mathbf{A}^{n-v}\mathbf{B}\right]_{a_2}\mathbf{u}(k+v-1) \oplus \mathbf{J}_{a_2}\mathbf{u}(k+n)\end{aligned}$$

The right side of the channel constraint can be rewritten as

$$\sigma_s(a_1, k) \otimes e(a_1) = e(a_1) \otimes \mathbf{H}_{a_1}\mathbf{x}(k) \oplus e(a_1) \otimes \mathbf{J}_{a_1}\mathbf{u}(k)$$

So, for all initial tokens x_i with $1 \le i \le m$ and inputs u_j with $1 \le j \le |I|$, we can rewrite the channel constraint to be

$$\begin{aligned}\left[\mathbf{H}\mathbf{A}^n\right]_{a_2,x_i}\mathbf{x}(k)[x_i] \oplus \bigoplus_{v=1}^{n} \left[\mathbf{H}\mathbf{A}^{n-v}\mathbf{B}\right]_{a_2,u_j}\mathbf{u}(k+v-1)[u_j] \oplus \mathbf{J}_{a_2,u_j}\mathbf{u}(k+n)[u_j] \ge \\ e(a_1) \otimes \mathbf{H}_{a_1,x_i}\mathbf{x}(k)[x_i] \oplus e(a_1) \otimes \mathbf{J}_{a_1,u_j}\mathbf{u}(k)[u_j]\end{aligned}$$

Clearly, as $\bigoplus$ represents the maximum over all v for $1 \le v \le n$, the following holds

$$\bigoplus_{v=1}^{n} \left[\mathbf{H}\mathbf{A}^{n-v}\mathbf{B}\right]_{a_2,u_j}\mathbf{u}(k+v-1)[u_j] \ge \left[\mathbf{H}\mathbf{A}^{n-1}\mathbf{B}\right]_{a_2,u_j}\mathbf{u}(k)[u_j]$$

So, it suffices to prove that

$$\begin{aligned}\left[\mathbf{H}\mathbf{A}^n\right]_{a_2,x_i} &\ge e(a_1) \otimes \mathbf{H}_{a_1,x_i}\\ \left[\mathbf{H}\mathbf{A}^{n-1}\mathbf{B}\right]_{a_2,u_j} &\ge e(a_1) \otimes \mathbf{J}_{a_1,u_j}\end{aligned} \tag{5}$$

In Section 7, we have already proven that $\left[\mathbf{H}\mathbf{A}^n\right]_{a_2,x_i} \ge e(a_1) \otimes \mathbf{H}_{a_1,x_i}$ (Equation 3). To also prove the second constraint, the entry $\left[\mathbf{H}\mathbf{A}^{n-1}\mathbf{B}\right]_{a_2,u_j}$ can be rewritten as

$$\left[\mathbf{H}\mathbf{A}^{n-1}\mathbf{B}\right]_{a_2,u_j} = \bigoplus_{p=1}^{m}\bigoplus_{q=1}^{m} \mathbf{H}_{a_2,x_q}\mathbf{A}^{n-1}_{x_q,x_p}\mathbf{B}_{x_p,u_j}$$

Thus, the following constraint shows the existence of a path that starts from u_j, passes through x_1, continues through x_n, and finally reaches a_2.

$$\bigoplus_{p=1}^{m}\bigoplus_{q=1}^{m} \mathbf{H}_{a_2,x_q}\mathbf{A}^{n-1}_{x_q,x_p}\mathbf{B}_{x_p,u_j} \ge \mathbf{H}_{a_2,x_n}\mathbf{A}^{n-1}_{x_n,x_1}\mathbf{B}_{x_1,u_j}$$

Since token x_n is the last token on the channel from actor a_1 to a_2, we have $\mathbf{H}_{a_2,x_n} = 0$. Additionally, the longest paths between tokens from token x_1 to token x_n all have weight 0, as these tokens reside on the same channel. Thus, $\mathbf{A}^{n-1}_{x_n,x_1} = 0$. Furthermore, $\mathbf{B}_{x_1,u_j}$ represents the longest path from the input u_j to token x_1. Since x_1 is the first token on the channel from actor a_1 to a_2, we know that $\mathbf{B}_{x_1,u_j} = e(a_1) \otimes \mathbf{J}_{a_1,u_j}$. Thus, we can show that the second constraint of Equation 5 holds as follows.

$$\begin{aligned}\bigoplus_{p=1}^{m}\bigoplus_{q=1}^{m} \mathbf{H}_{a_2,x_q}\mathbf{A}^{n-1}_{x_q,x_p}\mathbf{B}_{x_p,u_j} &\ge e(a_1) \otimes \mathbf{J}_{a_1,u_j}\\ \left[\mathbf{H}\mathbf{A}^{n-1}\mathbf{B}\right]_{a_2,u_j} &\ge e(a_1) \otimes \mathbf{J}_{a_1,u_j}\end{aligned}$$

Input validity: Consider an HSDFG G with state-space schedule σ_s and input arrivals α_s. It has an actor $a_1 \in A$ and an input $u_1 \in I$ with $d_I(u_1) = a_1$. That means that the following constraint must hold for all $k \in \mathbb{N}$.

$$\sigma_s(a_1, k) \geq \alpha_s(u_1, k)$$

This can be rewritten as

$$\mathbf{H}_{a_1} \otimes \mathbf{x}(k) \oplus \mathbf{J}_{a_1} \otimes \mathbf{u}(k) \geq \mathbf{u}(k)[u_1] \tag{6}$$

From

$$\mathbf{J}_{a_1} \otimes \mathbf{u}(k) = \bigoplus_{i=1}^{|I|} \mathbf{J}_{a_1,u_i} \otimes \mathbf{u}(k)[u_i]$$

it follows that

$$\bigoplus_{i=1}^{|I|} \mathbf{J}_{a_1,u_i} \otimes \mathbf{u}(k)[u_i] \geq \mathbf{J}_{a_1,u_1} \otimes \mathbf{u}(k)[u_1]$$

Thus, the left-hand side of Equation 6 can be rewritten as follows

$$\mathbf{H}_{a_1} \otimes \mathbf{x}(k) \oplus \bigoplus_{i=1}^{|I|} \mathbf{J}_{a_1,u_i} \otimes \mathbf{u}(k)[u_i] \geq \mathbf{J}_{a_1,u_1} \otimes \mathbf{u}(k)[u_1]$$

Since the input u_1 is connected to the actor a_1, we have $\mathbf{J}_{a_1,u_1} = 0$, as the set of paths $\mathcal{P}(u_1, a_1)$ includes the empty path (which has weight zero) and potentially paths that contain a token-free cycle with a weight of at most zero. Therefore,

$$\mathbf{J}_{a_1,u_1} \otimes \mathbf{u}(k)[u_1] \geq \mathbf{u}(k)[u_1]$$

proving Equation 6. □

A.2 Proof that the state-space schedule is the ASAP schedule

In Section 7.2, we showed that the state-space schedule σ_s is the ASAP schedule for HSDFGs without inputs, since $\sigma_s \leq \sigma$ holds for any other valid schedule σ. Let G be an HSDFG with a set of inputs I. If both σ_s and σ have the same input arrivals α_s, then for all $u \in I$ with $d_I(u) = a$, the constraints $\sigma_s(a, k) \geq \alpha_s(u, k)$ and $\sigma(a, k) \geq \alpha_s(u, k)$ must hold for every $k \in \mathbb{N}$, as specified in Definition 4.2. Therefore, it follows that $\sigma(a, k) \geq \sigma_s(a, k) \geq \alpha_s(u, k)$, confirming that the proof in Section 7.2 remains valid even when inputs are considered, as both schedules must satisfy the input timing constraints.

Received 30 March 2025; revised 16 June 2025; accepted 11 July 2025