On the Analysis of Synchronous Dataflow Graphs

a system-theoretic perspective

Robert de Groote

UNIVERSITY OF TWENTE.

Faculty of Electrical Engineering, Mathematics and Computer Science, Computer Architecture for Embedded Systems (CAES) group

CTIT Ph.D. thesis Series No. 15-382
Centre for Telematics and Information Technology
PO Box 217, 7500 AE Enschede, The Netherlands

This research is conducted within the Asynchronous and Dynamic Virtualisation through Performance Analysis to support Concurrency Engineering (Advance) project (Grant Agreement No. 248828) supported under the FP7-ICT-2009.3.6 program of the European Commission.

This research is conducted within the Programming Large-Scale Heterogeneous Infrastructures (Polca) project (Grant Agreement No. 610686) supported under the FP7-ICT-2013.3.4 program of the European Commission.

This thesis was typeset using LaTeX, Ti*k*Z, and Vim. This thesis was printed by Gildeprint Drukkerijen, The Netherlands.

ISBN 978-90-365-4041-4
ISSN 1381-3617; CTIT Ph.D. Thesis Series No. 15-382
DOI 10.3990/1.9789036539975404144

On the Analysis of Synchronous Dataflow Graphs

a system-theoretic perspective

Proefschrift

ter verkrijging van
de graad van doctor aan de Universiteit Twente,
op gezag van de rector magnificus,
prof. dr. H. Brinksma,
volgens besluit van het College voor Promoties
in het openbaar te verdedigen
op vrijdag 5 februari 2016 om 14.45 uur

door

Elibertus de Groote

geboren op 11 augustus 1978
te Emmen

Dit proefschrift is goedgekeurd door:

Prof. dr. ir.	G. J. M. Smit	(promotor)
Dr. ir.	J. Kuper	(assistant promotor)

ISBN 978-90-365-4041-4

Abstract

In the design of real-time systems, time forms a key requirement in a system's specification. System designers must be able to verify whether a system meets its timing demands or not, e.g., whether it responds to input within a specific time window, or whether it is able to process data at a given rate. Synchronous dataflow (SDF) graphs are models of computation that allow for *conservative analysis* of a system's temporal dynamics. By assuming worst-case temporal behaviour for the system's components, a temporal analysis translates to guarantees on the timing of the system. This potentially leads to an *over-dimensioned* system, where buffers used for communication links may be larger and clock speeds may be higher than necessary.

Different classes of SDF graphs exist. These classes vary in the richness of the properties that specify the behaviour of a graph. The richer these properties, the smaller the graph needed to express the same behaviour. As a result, the difficulty of the analysis of an SDF graph depends on its succinctness: the richer the properties of a graph, the more computationally demanding its analysis. In this thesis, we consider the following three classes, in order of increasing succinctness: *homogeneous* (HSDF), *multi-rate* (MRSDF, sometimes referred to as SDF) and *cyclo-static* (CSDF) synchronous dataflow.

Current approaches to the analysis of SDF graphs are divided into two main lines of thought. The first consists of those approaches that perform an *exact* analysis by considering the temporal dynamics of the graph at the finest possible level of granularity. As a result, the computed performance characteristics are *tight*: if the components of the system behave according to their worst-case behaviour, then the system's performance matches the performance predicted by the model. Consequently, over-dimensioning of the system is kept to a minimum. A disadvantage of the approach is its scalability: while HSDF graphs may be analysed in polynomial time, for MRSDF and CSDF graphs, analysis has an exponential time complexity.

Approaches that belong to the second line of thought are those that aim for a low computational complexity of the analysis, using *approximation*. To achieve this, they simplify the potentially complex patterns that compose the temporal behaviour of an SDF graph. This simplification is *conservative*: performance characteristics computed from this simplified behaviour are pessimistic with respect to the worst-case behaviour of the system. As a result, the system may be over-dimensioned to a larger extent when compared to an exact analysis. This is compensated by better scalability compared to exact methods, as it allows the three classes of dataflow to

be analysed in polynomial time.

With respect to accuracy and computational complexity, these two lines of thought are currently at different ends of a discontinuous spectrum. In case an exact analysis is computationally too expensive, the only alternative is a, possibly too coarse, conservative approximation. Likewise, the sole alternative to an overly pessimistic approximation is an exact analysis that may potentially be too costly.

In this thesis, we develop a mathematical characterisation of SDF graphs that provides a basis for combining the two lines of thought described above. We view SDF graphs as *linear discrete event systems*, which may be described elegantly using a mathematical structure called max-plus algebra. The central conviction of this thesis is that this system theoretic perspective unifies current approximate and exact approaches, by allowing for *incremental analysis*, in which an initial rough estimate may be improved in a stepwise fashion, until the result is accurate enough.

This approach to the analysis of synchronous dataflow graphs consists of two main building blocks, the first of which is *approximation*: we present transformations of CSDF graphs into a pair of equally-sized HSDF graphs, which give a simplification of the temporal behaviour of the CSDF graph. These HSDF graphs, which we refer to as *single-rate approximations*, may be analysed efficiently, and their performance characteristics provide bounds on those of the CSDF graph.

The second building block consists of *graph transformations*, which increase the level of detail of a specific part of the graph, by expanding it into a larger subgraph. This transformation generalises existing transformations, such as the construction of *single-rate equivalents*. Furthermore, it adds novel transformations, such as the construction of *multi-rate equivalents*: MRSDF graphs that express the same behaviour as the more succinct CSDF graphs.

As an application of the theory presented in this thesis, we present two approaches to the computation of the *throughput*, which is a primary performance characteristic, of an multi-rate synchronous dataflow *(MRSDF)* graph. Our first approach is an *exact* approach that exploits the structure of the single-rate equivalent of the graph, restricting the analysis to a subgraph. Our second approach combines the approximations and transformations into an *incremental* approach, which iteratively improves the accuracy of the analysis by *partially unfolding* critical actors.

We validate the soundness of our approach to throughput analysis by applying it to a number of benchmark sets and case studies and comparing the results with current state-of-the art approaches. This comparison confirms the efficiency of our exact approach, and the validity of our incremental approach: our exact method computes the throughput of an SDF graphs in a fraction of the time required by state-of-the-art approaches, and our incremental approach trades off accuracy with size of the analysed graph.

Samenvatting

Tijd speelt een belangrijke rol in het ontwerp van real-time systemen. Onwerpers van zulke systemen moeten na kunnen gaan of aan alle eisen met betrekking tot timing wordt voldaan. Zo moet bijvoorbeeld geverifieerd kunnen worden of het systeem binnen de vereiste tijdsspanne reageert op een specifieke invoer, of dat het systeem in staat is om een bepaalde hoeveelheid gegevens per tijdseenheid te verwerken. Synchrone dataflow (SDF) grafen zijn modellen van berekeningen, waarmee een *conservatieve analyse* van het temporele gedrag van een systeem kan worden uitgevoerd. Door, per component in het systeem, van *worst-case* timing uit te gaan, vertaalt een temporele analyse zich naar garanties over de timing van het systeem. Een potentieel gevolg hiervan is dat het systeem over-gedimensioneerd wordt: buffers krijgen een onnodig grote capaciteit en kloksnelheden zijn hoger dan noodzakelijk.

Er bestaan verschillende klassen van SDF grafen, die variëren in hoe uitgebreid de eigenschappen zijn waarmee het gedrag van een graaf kan worden gespecificeerd. Hoe uitgebreider deze eigenschappen, hoe kleiner de graaf die nodig is om het zelfde gedrag uit te drukken, en hoe meer tijd een analyse van deze graaf in beslag zal nemen. In dit proefschrift beschouwen wij de volgende drie klassen, in toenemende mate van compactheid: *homogene* (HSDF), *multi-rate* (MRSDF, soms kortweg SDF genoemd) en *cyclo-statische* (CSDF) synchrone dataflow.

Huidige aanpakken voor de analyse van SDF grafen zijn onder te verdelen in twee gedachtengangen. De eerste gedachtengang omvat benaderingen die een *exacte* analyse uitvoeren, door in de analyse de fijnste details van het temporele gedrag van de graaf mee te nemen. Hierdoor komt het berekende gedrag van het model nauwkeurig overeen met het worst-case gedrag van het systeem. Een voordeel hiervan is dat het over-dimensioneren van het systeem tot een minimum wordt beperkt. Een sterk nadeel van deze benadering is de beperkte schaalbaarheid: waar HSDF grafen in polynomiale tijd kunnen worden geanalyseerd, heeft een exacte analyse van MRSDF en CSDF grafen een exponentiële complexiteit.

Aanpakken die behoren tot de tweede gedachtengang streven naar een *lage complexiteit* van de analyse, door gebruik te maken van *approximaties*. Hiertoe vereenvoudigen ze de complexe patronen die aan het tijdsgedrag van een graaf ten grondslag liggen. Deze vereenvoudiging is *conservatief*: voorspellingen aangaande de prestaties van het systeem zijn pessimistisch met betrekking tot het werkelijke gedrag. Hierdoor kan het systeem sterker worden geoverdimensioneerd dan bij een exacte analyse. Dit nadeel wordt gecompenseerd door een betere schaalbaarheid

in vergelijking met exacte methoden: de drie genoemde klassen van SDF grafen kunnen allen in polynomiale tijd worden geanalyseerd.

Met betrekking tot nauwkeurigheid en complexiteit, bevinden de twee genoemde gedachtengangen zich aan verschillende uiteinden van een discontinu spectrum. Indien een exacte analyse computationeel te duur is, is het enige alternatief een (mogelijk te onnauwkeurige) conservatieve approximatie. Andersom is het enige mogelijke alternatief voor een pessimistische approximatie een potentieel te kostbare exacte analyse.

In dit proefschrift ontwikkelen we een wiskundige karakterisatie van SDF grafen, die een basis vormt voor het combineren van de twee genoemde gedachtengangen. Hierbij beschouwen wij SDF grafen als *lineaire discrete event systemen*, welke elegant beschreven kunnen worden in een wiskundige structuur genaamd max-plus algebra. De centrale overtuiging in dit proefschrift is dat dit systeem-theoretische perspectief de huidige approximerende en exacte aanpakken verenigt, door een *incrementele analyse* toe te staan, waarin een initieel grove benadering stapsgewijs kan worden verbeterd, totdat het verkregen resultaat nauwkeurig genoeg is.

Deze incrementele aanpak voor de analyse van synchrone dataflow grafen is opgebouwd uit twee *transformaties*. De eerste hiervan is een *approximerende*: deze transformatie zet een CSDF graaf om naar een tweetal HSDF grafen waarvan het temporele gedrag een versimpeling is van dat van de CSDF graaf. Deze HSDF grafen, genaamd *single-rate approximaties*, kunnen efficiënt worden geanalyseerd, en de verkregen prestatiekenmerken geven grenzen aan die van de CSDF graaf.

De tweede transformatie is een *graaf uitvouwing*, welke details in het gedrag van een specifiek deel van de graaf uitlicht, door dat deel te vervangen door een grotere graaf. Deze transformatie generaliseert bestaande transformaties, waaronder de constructie van zogenaamde *single-rate equivalenten*. Verder biedt het een aantal nieuwe transformaties, zoals de constructie van een *multi-rate* equivalent: dit is een MRSDF graaf met hetzelfde temporele gedrag als die van de meer beknopte CSDF graaf.

Als toepassing van de in dit proefschrift gepresenteerde theorie, geven wij twee aanpakken voor de berekening van een primair prestatiekenmerk, namelijk *doorvoersnelheid*, van een MRSDF graaf. Onze eerste aanpak is een exacte aanpak die de structuur van de single-rate equivalent van de graaf uitbuit, en hierdoor de analyse beperkt tot een kleinere graaf. Onze tweede aanpak is een incrementele aanpak, welke stapsgewijs de nauwkeurigheid van de analyse verbetert, door steeds zogenaamd kritieke actoren uit te vouwen. We valideren de degelijkheid van deze twee aanpakken op een aantal benchmark sets en case studies, en vergelijken de resultaten met de huidige state-of-the art benaderingen. Deze vergelijking bevestigt de doeltreffendheid van onze exacte aanpak en de validiteit van onze stapsgewijze aanpak. Onze incrementele aanpak laat de wisselwerking tussen nauwkeurigheid van de analyse en grootte van de geanalyseerde graaf zien, en onze exacte methode berekent de doorvoersnelheid van een SDF graph in een fractie van de tijd die state-of-the-art methodes nodig hebben.

Dankwoord

Daar is 'ie dan! De bevalling was zwaar, de draagtijd wat langer dan gebruikelijk, maar met het schrijven van dit dankwoord rond ik dan eindelijk mijn proefschrift af. Daarmee komt een einde aan een lang en soms zwaar proces, waar ik echter geen moment spijt van heb gehad. Gelukkig heb ik dit proces niet alleen doorlopen, en daarom wil ik graag een aantal mensen bedanken.

Allereerst wil ik Jan bedanken, vooral voor zijn begrip van de zoektocht waar een promovendus zich door heen beweegt en alle leermomenten die hierbij horen. Jan, bedankt voor je begeleiding in het worden van wat ik nu ben, van dag 1 tot het moment dat je met een fles whisky aanbelde om het verzenden van het concept-proefschrift te vieren. Bedankt dat je me de ruimte hebt gegeven om zelf een richting te vinden, vrij van de reeds vaag uitgestippelde lijnen van het onderzoeksproject waarin ik begon (iets met statistiek?). Door jou is dit proefschrift een logboek van mijn eigen speurtocht geworden.

Ik had natuurlijk nooit aan dit avontuur kunnen beginnen zonder het vertrouwen van Gerard. Gerard, bedankt voor de positieve manier van het leiden van de vakgroep, waar mensen centraal staan, niet processen.

Naast Jan en Gerard heb ik ontzettend veel gehad aan Philip als begeleider. Philip, bedankt dat ik altijd bij je kon binnen vallen om te sparren over iets waar ik op vast zat en voor je opbouwende kritiek. We hebben samen vele uren voor het whiteboard gestaan om te proberen soms kromme gedachtengangen te formaliseren. Maar naast de inhoudelijke begeleiding wil ik je vooral bedanken voor je vermogen om mij en anderen te motiveren. Het zinnetje "lower your standards" klinkt nog regelmatig in mijn hoofd en het lukt me nu om de lat wat lager te leggen dan voorheen (al kan hij nog flink wat verder zakken).

Tijdens mijn tijd als promovendus heb ik met heel wat mensen verschillende kamers gedeeld: Timon, Anja, Berend (ik vind het geweldig dat jullie twee uit Trondheim komen om bij mijn verdediging te zijn), Bart, Diego en Hermen: bedankt voor de goede sfeer en het delen van zowel de frustraties en beslommeringen als het enthousiasme en de persoonlijke overwinninkjes.

Marco, Jochem en Arjan, bedankt voor de werktijd die jullie kwijt waren aan de (in retrospect soms bijzonder belachelijke) overpeinzingen die ik vaak ongevraagd in jullie kantoor kwam delen. Ik heb niet bijgehouden hoeveel taart-weddenschappen er zijn afgesloten, maar ik weet wel dat ik ze vaak verloor. Door al die weddenschappen vrees ik (eigen schuld) de traditionele sketch bijna meer dan de verdediging van dit proefschrift.

De bezoeken aan conferenties waren vooral ook leuke uitstapjes, maar zo hier en daar gingen er wel eens wat dingen bijna mis. Hierbij ben ik vooral veel dank verschuldigd aan Rinse voor het vervoeren van een niet nader te noemen doch zeer belangrijk reisdocument. Ik zal het nog vaak (en terecht) moeten aanhoren, maar dat zou ik veel vervelender hebben gevonden wanneer je drie minuten later in Düsseldorf was aangekomen. En natuurlijk moet ik hier ook weer Philip bedanken, voor het ter beschikking stellen van zijn bolide. I owe you guys!

Marlous, Nicole en Thelma, bedankt voor alle hulp bij het boeken van reizen en hotels, invullen van declaratieformulieren, chocola, pepernoten, etc. Maar vooral ook voor de tijd die jullie altijd hebben om, tussen al het denk-geweld, over iets anders dan werk te praten.

Verder wil ik alle (ex-)collega-lotgenoten van CAES bedanken voor de goede sfeer en toffe pauzes, in het bijzonder Thijs, Jonathan en Koen voor de hardloopkilometers over de campus en omstreken, Albert voor het fanatisme op de squashbaan (de goede oude tijd van de 9-0 is voorbij...), Christiaan en Gerwin voor het dagelijks optrommelen van de kudde voor de lunchwandelingen en natuurlijk Jochem voor het gemak dat ik en vele andere promovendi hebben van zijn LaTeX template. Ik denk dat het goede groepsgevoel, zoals die er bij CAES is, een omgeving schept waarin mensen elkaar inspireren en waarin ideeën makkelijker tot wasdom komen. Daar wil ik iedereen voor bedanken. Ik wens alle AIO's die op het moment van schrijven met hun promotie-onderzoek bezig zijn heel veel succes met het afronden ervan. Tenslotte hoop ik dat het onderwerp van dit proefschrift, *synchrone dataflow*, binnen de vakgroep in stand wordt gehouden door Marco, Guus, Viktorio en Philip.

Gelukkig bestaat er ook nog een leven naast het promotie-onderzoek, al laat het nadenken zich vervelend goed meenemen van werk naar huis. Arjan, Derk, Henry, Joost, Joost, Jos, Lodewijk, Martijn en Martijn: het "wie wilt weg" (of wat de afkorting www ook mag betekenen) weekend in januari slaagt er elk jaar weer om het werk even totaal te vergeten. Of het nu Bremen, Berlijn, Amsterdam, Barcelona of (als ik de verhalen mag geloven) St. Martijn is, het is een feest om een paar dagen niks te moeten en alles te mogen. De timing is voor mij dit jaar misschien wat minder, maar een betere manier om de spanning voorafgaand aan de verdediging, een week later, te breken, kan ik me niet voorstellen.

Rutger, Ronald, Marloes en Liset: bedankt voor de donderdagmiddagen die ik kon besteden aan het afronden van mijn boekje, omdat Xem na school bij jullie terecht kon. Ook wil ik jullie bedanken voor de squash, films, spelletjes, etentjes en verjaardagen die vaak met hoofdbrekens om een of ander lastig raadsel eindigden. Ik hoop dat er nog vele zullen volgen!

Roy en Harold, bedankt voor de vele potjes squash, Puerto Rico, Agricola, Terra Mystica, Tikal, Catan, Machiavelli en Ganzenbord, die ik slechts bij hoge uitzondering (imba!) eens won. Ik hoop dat we nog duizenden stinknoten zullen wegwerken

en dat ik nog wat meer winstpartijen bij mag schrijven dan ik in de afgelopen twaalf jaar verzameld heb. Het is een geruststellend idee dat jullie straks links en rechts van mij zitten tijdens het drie kwartier durende vragenvuur.

Mijn schoonouders, Henk en Gerda, wil ik bedanken voor het “thuis” dat jullie zijn, en waar ik me vanaf het begin welkom mocht voelen. Het buitenleven, met in de wijde verte de schone dreven, is vooral in de zomer altijd heel erg ontspannend.

Alle (goede en minder goede) eigenschappen die me soms hebben geholpen en soms hebben tegengewerkt bij het afronden van dit proefschrift zijn gevormd tijdens mijn opgroeien in Erica. Pap en mam, bedankt voor alles wat jullie hebben gedaan om mij (en Hendrie) in staat te stellen om “door te leren”, ook al heeft dat betekend dat we nu veel minder dichtbij wonen en minder vaak even langs komen dan jullie vaak wensen. Hendrie en Kelly, ook jullie wonen niet direct om de hoek en we zien elkaar daarom eigenlijk te weinig. Toch is het altijd fijn om op bezoek te zijn en de kinderen met elkaar te zien spelen, en ik hoop dat we dat in de toekomst nog vaak zullen doen.

Last but not least ben ik enorm veel dank verschuldigd aan wat al bijna 15 jaar mijn thuis is. Lieve Marloes en Xem, ik ben in de afgelopen paar jaren niet altijd de leukste vriend of papa geweest, met een afwezig hoofd dat nog vol met werk zat. Zonder jullie had ik dit boekje nooit af kunnen ronden. Marloes, bedankt voor je enorme geduld, stabiliteit en positiviteit, die gelukkig sterker zijn dan het pessimisme dat ik soms na een frustrerende werkweek mee naar huis nam. Xem, ik sta nog steeds te kijken van de enorme hoeveelheid energie die je elke dag weer laat zien. Van jouw opgewektheid, flexibiliteit en frisse opmerkzaamheid heb ik misschien wel het meest geleerd in de afgelopen paar jaar.

Robert
Enschede, januari 2016

Contents

Introduction

Abstract – *synchronous dataflow (SDF) is a popular model of computation, used to model stream-processing applications. Analysis of the temporal dynamics of an SDF graph provides guarantees with respect to the performance of the system it models. Different classes of SDF graphs exist. As the richness of the model's properties increases, so does the potential complexity of its dynamics. Current analysis techniques are divided into* approximate *and* exact *methods. Exact methods are accurate but time consuming, whereas approximations are easier to compute, but potentially inaccurate. The main contribution of this thesis is a mathematical basis for characterising the temporal dynamics of an SDF graph. This basis is constructed from a system-theoretic perspective, and unifies existing approximate and exact approaches, by providing a set of* graph transformations. *As a result, the accuracy of an approximate analysis can be balanced with its computational costs, giving a* scalable *approach.*

A *computer* is a machine that performs *computations*. Computations map a sequence of input values to a sequence of output values. Computers perform computations by following a series of predefined steps, given in the form of programs. An important aspect of these computations is the *time* they take; some systems only function correctly, if they respond to an input within a certain time window. Examples of these systems are found in many systems that are *embedded* in larger (often mechanical) machines, such as the electronic braking systems in a car or the guiding system in rockets and satellites. Because timing is an integral part of their correct functioning, these systems are called *real-time* (embedded) systems.

Computers have advanced tremendously over the past decades; they have become both smaller and faster. The time a processor needs to perform a particular computation has decreased by several orders of magnitude, especially since the mid-1980s. Whereas in the twentieth century the increase in performance was primarily due to smaller components (a trend that follows the well-known law of Moore) allowing for higher clock frequencies, the last twenty years has seen many improvements in the *organisation* of the work carried out by a processor and its peripherals. For example, the use of caches in the retrieval of data from memory decreases the time to access data that is intensively used, and speculative execution performs work before it is known whether that work will be needed. In particular, the inclusion of multiple processors in a computer allows computations to be performed in a distributed way; partial results are computed on different processors and then combined to form the result of the full computation.

When organising a distributed computation, one must take into account the *dependency structure* of the computation: some steps in a computation require that other steps are completed first. These steps must thus be executed *sequentially*. Steps that do not depend on each other's completion, in the sense that they do not require input from other steps, can be executed in *parallel*. The available hardware resources further imposes restrictions on the organisation of work. For example, the number of processors limits the number of partial results that can be computed in parallel.

In the context of real-time embedded systems, computations must be organised in such a way that their timing satisfies a set of constraints. In order to optimise the performance of a computation, one must be able to study the interplay between its organisation, the limitations imposed by the hardware resources, and the performance. The motivation behind this thesis is that the analysis of this interplay is best done through the use of a *model of computation*, and that SDF is a suitable model to start with. An SDF model allows one to analyse the performance of a computation by incorporating non-functional properties, such as the time a computation takes, into the dependency structure of the computation. Performance analysis of real-time embedded systems using SDF is the main topic of this thesis.

1.1 Models of computation

Computations can be decomposed into smaller computations: for example, the product C of two matrices A and B can be obtained by computing the products of each of the rows of A with matrix B separately. These smaller computations may be distributed over different computers, and the result of the larger computation may be obtained by combining the partial results. A natural way to depict a computation is by means of a *directed graph* or a *network*. Each node in the graph corresponds to a step in the computation, and arcs correspond to dependencies between steps.

A *model of computation* explains how the behaviour of the whole system is the result of the behaviour of each of its components[1]. When representing a computation as a directed graph, a model of computation describes how each node must combine its inputs to produce its outputs. Several such models have been proposed over the past decades.

[1]This definition of model of computation is borrowed from the field of model-driven engineering. The term *model of computation* may also refer to the modelling of the computation steps carried out by a machine, in complexity theory, or to mathematical abstractions of computations such as *Turing machines* and *lambda calculus*.

In the year 1974, Gilles Kahn wrote an influential article on what he called "a simple language for parallel programming", showing how computations may be distributed over a network of computing devices [53]. The "how" consists of a set of rules that these distributed computations must obey in order to yield *functional determinacy*, which means that, when fed with the same input sequence, the computation yields the same output sequence. The *networked* computations introduced by Kahn are referred to as *Kahn process networks*. A key result proven by Kahn is that networks of sequential processes, which compute functions over their input data and communicate through unbounded first-in first-out channels, are functionally determinate.

Prior to the introduction of Kahn's process networks, a similar conclusion was presented in the context of a more restricted model for parallel computations, named *computation graph*, presented by Karp and Miller in 1966 [55]. In a computation graph, arcs represent first-in first-out queues, and each node represents a *function*. Nodes read data from their input arcs, and produce data on their output arcs. This data is represented by markers placed on arcs, called *tokens* and commonly depicted by solid dots. Arcs are annotated by four numbers, which indicate the number of tokens read and produced by the corresponding functions, the number of tokens initially present on the arc, and the minimum number of tokens that must be present on the arc before tokens may be read from it.

Dataflow is a paradigm in which a computation is viewed as a network (i.e., a directed graph) of concurrently executing processes that communicate by sending data over channels. In a *dataflow graph*, nodes are called *actors*, and arcs are referred to as *channels*. Dataflow graphs are a special case of Kahn's process networks [61]. The central model of this thesis was introduced in the 1980s, and is named *synchronous dataflow* [60]. A synchronous dataflow (SDF) graph can be regarded as a restriction of Karp and Miller's computation graphs: the number of tokens *produced* (i.e., written to an output channel) and *consumed* (i.e., read from an input channel), per firing of an actor, is known a priori. Different varieties of SDF graphs exist. In this thesis, we treat the three most prominent ones, listed below in increasing order of the richness of their properties:

Homogeneous synchronous dataflow (HSDF)
: *These graphs were studied by Reiter in 1968, as a special case of the computation graphs of Karp and Miller [78]. In an HSDF graph, each actor firing involves the consumption if a single data token from each of the actor's incoming channels, and the production of a single token onto each of its outgoing channels.*

Multi-rate synchronous dataflow (MRSDF)
: *In MRSDF graphs, introduced in [60], actors produce and consume data at fixed but different rates. This is the model that was proposed in the introductory paper on SDF, where the adjective "multi-rate" was simply omitted. An MRSDF graph is a special case of the so-called* computation graphs *introduced by Karp and Miller in the late sixties.*

Cyclo-static dataflow (CSDF)

> *CSDF graphs were introduced by Bilsen in [11]. In a CSDF graph, actors have periodically varying behaviour. Each actor cycles through a fixed number of* phases, *and an actor's execution time, as well as its production and consumption rates, may differ per phase. This model was introduced about 15 years after the introduction of synchronous dataflow, as a more versatile variant of MRSDF. In the original definition of [11], actors in the graph fire in a strictly sequential fashion, which means that CSDF does not, taxonomically speaking, generalise MRSDF.*

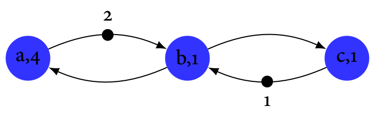
The theory and applications that we introduce in this thesis apply to CSDF graphs. In particular, in our view on CSDF, we allow for actors that have varying execution times, without limiting these actors to run in a purely sequential fashion. This gives a natural taxonomy, where CSDF is a true generalisation of MRSDF, and MRSDF generalises HSDF.

1.2 Performance Analysis

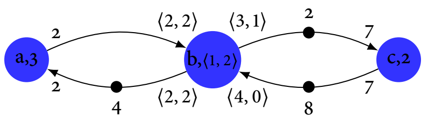
An attractive property of SDF graphs is their *analysability*. In an SDF graph, the durations of firings of each actor, as well as the production and consumption rates associated with firings, are constant integers. As a result, one may determine whether the graph allows for an infinite number of firings and whether the queues associated with the channels can be realised in bounded memory. Furthermore, the times at which actors fire follow a repetitive pattern (here we assume that graphs satisfy a property called *consistency*, which we explain in further detail in Chapter 2), called a *graph iteration*, which may be computed a priori. The potential complexity of these patterns (i.e. the length of a graph iteration), is lowest for HSDF graphs, and highest for CSDF graphs. This is illustrated in Figure 1.1, which shows the firing times of an HSDF actor, as well as those for a CSDF actor. The number of firings that compose a single pattern for CSDF actor b is 14, whereas the HSDF firing pattern consists of a single firing.

Analysis of the performance of an SDF graph involves computing the details of these firing patterns, by analysing the constraints on actor firing times, within a single graph iteration. The difficulty of analysis thus depends on the size of an iteration. Adding a single channel to a graph may result in a proportional increase in the length of an iteration of the graph: the size of a graph iteration thus grows exponentially in the size of the graph.

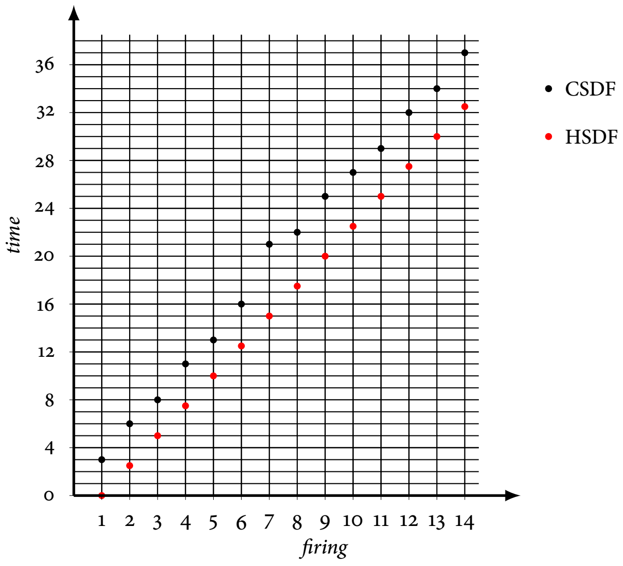
There are currently several different approaches to the analysis of SDF graphs, which may be grouped into two main *schools of thought*. The first of these two schools consists of those approaches that are *exact*, meaning that they compute the precise details of the potentially long firing patterns. Since the size of a graph iteration grows exponentially in the size of the graph, methods that fall in this class do not scale very well.



(a) HSDF graph.



(b) CSDF graph.



(c) Firing times of actor b.

Figure 1.1 – The complexity of firing patterns is determined by the kind of SDF graph. In HSDF graphs, actors may be scheduled to fire in a strictly periodic fashion, whereas for a CSDF graph, the complexity of such a pattern depends on the graph's properties.

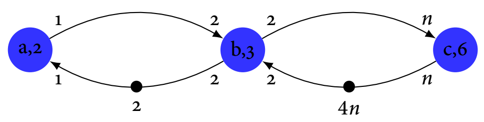


Figure 1.2 – An example of an MRSDF graph for which an exact approach does unnecessary work, and an approximation yields a large error.

The second school is concerned with *approximate* approaches: These approaches can be characterised as working by *assuming* specific firing patterns rather than deriving them. For example, they may assume that each actor fires in a strictly periodic fashion, which means that an SDF graph is essentially treated as if it were an HSDF graph. As a result, these methods scale much better, as they do not depend on the length of a firing pattern. However, the simplification of these patterns gives rise to an approximation error, which may be large.

1.3 Problem statement and approach

Exact methods provide an accuracy that may not be required, and require exponential time, whereas approximate methods require polynomial time, but provide an unknown accuracy. Existing methods fail for graphs for which an exact analysis is infeasible due to the length of a single graph iteration, and approximation yields a too pessimistic result. An example of such a graph is given in Figure 1.2: the performance bottleneck in the graph is, independent of the value of the parameter n, formed by cycle aba. For the graph, both the error of a conservative approximation, and the length of a single iteration of the graph, increase linearly in n. Choosing n very large thus renders the approximation useless in terms of accuracy, and makes an exact analysis too costly.

Graphs such as Figure 1.2 reveal two main shortcomings of the current state-of-the-art: First of all, current approximate analysis techniques do not provide any means to assess their accuracy. As a result, systems designed to satisfy real-time constraints may be severely over-dimensioned: as the throughput of the system, assessed by an approximate technique, may be underestimated, the amount of hardware resources required to let the system meet its constraints are overestimated. An assessment of the approximation error would allow a designer to balance the accuracy of a quick approximation against the time required by a thorough and exact analysis.

A second shortcoming is the scalability of exact methods. These methods make no distinction between the criticality of different parts of the graph. Typically, only a small part of the graph determines its performance bottleneck. Current exact approaches treat the entire graph as potentially critical.

Underlying these two problems is the fact that no strong connection exists between current exact and approximate methods. For graphs for which exact analysis does not scale, the only alternative is a potentially highly inaccurate approximate analysis. This thesis fills the gap between exact and approximate methods by providing means to improve approximation accuracy by increasing the size of the graph, in a scalable way. The central research question addressed by this thesis is:

> *How can we combine exact and approximate analysis of synchronous dataflow graphs into an approach that offers a trade-off between accuracy and complexity?*

We approach this question by taking a *system-theoretic* perspective. In this view, SDF graphs are mathematical structures that define how *events*, such as the start or completion of a computation, or the communication of data, are interrelated, and how this restricts the times at which these events may take place. These mathematical structures are called *discrete event systems*, and are well-studied and described using *max-plus algebra*.

This mathematical view allows us to design a basis that is *shared* between the exact and approximate approaches. As a result, both kinds of approaches can be derived from it, which we demonstrate in this thesis. The perspective furthermore allows one to reason formally over the relation between graphs; in particular, it allows us to conclude whether the "behaviour" of two graphs is equivalent, or whether one graph always shows a better performance than another graph. The basis allows us to design an *incremental* approach to the analysis of SDF graphs; starting with a rough estimate, we show how this estimate may be incrementally improved by applying *transformations* to the graph under analysis.

1.4 Contributions

The main contribution of this thesis is a theoretical one: a sound mathematical basis that characterises the temporal behaviour of HSDF, MRSDF and CSDF graphs. The core of this basis is formed by viewing SDF graphs as *discrete event systems*. We use the elegant mathematical structure called *max-plus algebra* to turn this perspective into a formal definition on the set of schedules that are valid for these graphs.

From this mathematical basis, both exact and approximate analyses naturally follow. This *unifies* the two main schools of thought in literature, which are either concerned with exact or approximate analysis. Furthermore, the combination of exact and approximate analyses gives rise to a trade-off between accuracy and runtime of the analysis. This is achieved through *graph transformations* (for example, from CSDF into MRSDF), which involve a process of algebraic rewriting.

In addition, we observed that in the literature on approximate approaches two different perspectives can be distinguished, which we refer to as the *token transfer* perspective and the *actor firing* perspective. In the first, the temporal behaviour of a graph is described in terms of the times at which tokens move over channels, whereas in the second this behaviour is described in terms of the times at which actors complete their firings. Though closely related, we show that they differ in the accuracy they provide.

The above theoretical contributions lead to the following practical results:

» A novel transformation of CSDF graphs, in which actors are *unfolded* into multiple actors, which represent subsets of the firings of the original actor. Using this transformation, CSDF graphs may be transformed into their *multi-rate equivalent*: an MRSDF that has the same temporal behaviour as the CSDF graph. This transformation is the first of its kind and allows all existing analysis methods that apply to MRSDF to be generalised to CSDF. Furthermore, the transformation may be applied to the full graph, or to a smaller *subgraph*, giving rise to *partial transformations*. Rather than transforming an entire SDF graph into an equivalent HSDF graph, one may transform only those parts of the SDF graph that are of interest into a larger HSDF graph. A remarkable property of all these transformations is that they may be applied to graphs with *parameterised* initial tokens. This invalidates the current conviction, which is that the structure of an equivalent HSDF graph depends on the number of tokens.

» A set of novel *approximate* transformation from a CSDF graph into an HSDF graph, which we refer to as a *single-rate approximation*. Single-rate approximations are either *optimistic* or *pessimistic*, which refers to the fact that their respective performance characteristics form *upper* or *lower* bounds on those of the CSDF graph. Furthermore, they may be derived from the two different viewpoints mentioned above.

» A novel *incremental* approach to throughput analysis of MRSDF graphs, which combines the unfolding transformations and the approximate transformations. The resulting algorithm computes the throughput of an MRSDF graph by *iteratively* transforming only the parts of the graph that constrain the throughput (i.e., those parts that form a bottleneck) into a larger graph. As a result, accuracy of the analysis can be balanced with the complexity of the analysis.

1.5 Outline

This thesis is organised in a bottom-up fashion, from background theory on SDF graphs and max-plus algebra to the analysis of throughput. In Chapter 2 we present the necessary terminology and concepts that are used throughout the thesis. Furthermore, the chapter gives an overview of the current approaches to the transformation, approximation and analysis of SDF graphs and related models. Chapter 3 introduces the mathematical basis underlying the tools developed in the chapters following it. It is in this chapter where we define what constitutes a valid schedule for an SDF graph, using max-plus algebra.

Chapter 4 is the first chapter that applies the mathematical basis for practical purposes. In the chapter we demonstrate how, using the max-plus algebraic characterisation of Chapter 3, an SDF actor may be *unfolded* into its firings, at a chosen granularity. The chapter furthermore describes how the three different models, HSDF, MRSDF and CSDF, may be transformed into one another.

Chapter 5 demonstrates how MRSDF and CSDF graphs may be approximated by HSDF graphs, and how the error made by such an approximation may be assessed. Chapters 4 and 5 rely on a number of identities between *integer functions* such as the *modulo*, *floor* and *ceiling* operation, which may be found in Appendix A.

Chapter 6 combines the theory of chapters 4 and 5 to form a new approach to throughput analysis of SDF graphs, which addresses the central research question of this thesis. The chapter describes an incremental approach to throughput analysis by demonstrating how the accuracy of approximate analysis may be improved in a stepwise fashion, by applying transformations to those parts of the dataflow graph that are performance-critical.

Chapter 7 applies the presented incremental analysis method to a number of case studies, and discusses the results. Among the case studies is an influential comparison that was carried out and presented almost a decade ago. Finally, Chapter 8 concludes the thesis and presents recommendations for future work. The chapter furthermore contains a more detailed list of the contributions made by this thesis.

Background and related work

Abstract – *This chapter describes the three different classes of SDF graphs that this thesis deals with, and introduces the terminology used to describe their properties. SDF graphs form a subset of the broad class of* discrete event systems*, which are elegantly described used* max-plus algebra*. This chapter gives a brief introduction to discrete event systems and max-plus algebra, and its use in characterising the temporal behaviour of SDF graphs. This chapter furthermore discusses relevant literature on the transformation, approximation and performance analysis of SDF graphs and related models, such as Petri nets and computation graphs.*

Synchronous dataflow was introduced by Lee in 1987, as a programming paradigm for the design and implementation of stream processing systems [60]. Its main purpose, as presented by Lee, was to aid in the design of DSP applications for concurrent implementation on parallel hardware, by making concurrency, which is often available in signal processing algorithms, explicit. In the data flow paradigm, algorithms are described as directed graphs, where the graph's vertices represent computations and its edges represent data dependencies [31, 63]. Computations are data-driven: a vertex may *fire* (perform its computation) as soon as sufficient input data is available on its incoming edges. This data is modelled by *tokens*. A firing of a vertex involves the consumption of tokens from its incoming edges, and the production of tokens onto its outgoing edges. Following the naming convention that is common in literature, we refer to an SDF graph's vertices as *actors* and to its edges as *channels*. In a synchronous data flow graph, the number of tokens produced and consumed per firing (production and consumption *rates*) is known *a priori*. This makes SDF graphs statically schedulable and amenable for analysis. A crucial property added to the model is *time*, which allows for *temporal analyses* of the model.

Different kinds of SDF graphs exist. In the simplest class of SDF graphs, a single firing of an actor produces and consumes a single token onto and from incident

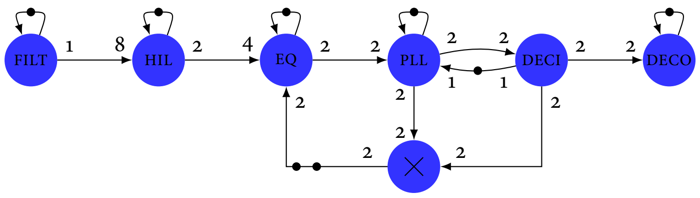


Figure 2.1 – A multi-rate SDF graph showing a voice-band data modem, taken from [60].

channels. Actors in these graphs are said to have a production and consumption rate of one. This class is called HSDF, and was introduced in [78]. We briefly touch upon analysis of HSDF graphs in Section 2.2.4.

After the introduction of the basic SDF model in [60], the model has been extended, by several authors, with several annotations and properties. By allowing the number of tokens that are produced and consumed by a single firing to differ per actor and per channel, we obtain the class of MRSDF graphs. This is the class that was introduced, using the more general name SDF, in the initial paper on synchronous dataflow, [60]. An example MRSDF graph is depicted in Figure 2.1. Analysis of MRSDF graphs involves the construction and subsequent analysis of an *equivalent* HSDF graph, called the *single-rate equivalent* of the MRSDF graph [51]. This approach is, however, penalised by the size of the latter: transforming an MRSDF graph into an equivalent HSDF graph has an exponential complexity [77].

The most general class that we consider in this thesis is CSDF, which was introduced by Bilsen in 1996, and allows the number of tokens produced and consumed, as well as an actor's execution time, to vary periodically [11]. This class of graphs generalises the scalar execution time and (production and consumption) rates, found in MRSDF, to *vectors*. The original definition of CSDF restricts the parallelism that is available to actors, by implicitly assuming that each actor has a *self-loop*, with a single token. As a result, CSDF actors are forced to run strictly sequentially, which is a restriction that does not apply to MRSDF graphs. In our definition of CSDF (see Section 2.1.5), we lift this restriction, such that MRSDF is contained in CSDF. We define the three classes listed above in terms of CSDF, which is the most succinct of the three, in Section 2.1.

Synchronous dataflow graphs form a subclass of the broad class of *discrete event systems* [17, 20, 22]. These systems interrelate the times at which events occur, using the operators *max* and + to capture respectively synchronisation and delay. The state space of a discrete event system describes how *timestamps* of events evolve over time [24]. We discuss this in more detail in Sections 2.2.1 and 2.2.

In this thesis, we regard synchronous dataflow graphs as discrete event systems. As such, a basic understanding of the algebra used to describe the latter is necessary. This algebra is referred to as *max-plus algebra*, and is described in further detail in Section 2.2.1. Building upon this algebra, Section 2.2 presents the basic terminology used to describe discrete event systems, and relates them to synchronous dataflow graphs. Spectral analysis of so-called *linear shift-invariant discrete event systems* is related to self-timed schedules of HSDF graphs, from which useful temporal properties such as throughput and latency may be derived [22, 49]. We discuss the relationship between spectral theory and these properties in Section 2.2.4.

In Section 2.3, we describe existing literature on the analysis of SDF graphs. In particular, we highlight related works on the transformation of SDF graphs into simpler graphs, and on the approximation of SDF graphs. Furthermore, we discuss several models related to SDF graphs, and dominant approaches to their analysis.

2.1 Cyclo-static SDF

In a CSDF graph, actors have cyclically varying behaviour: each actor cycles through a fixed number of *phases*. The phase that an actor is in determines its execution time and the number of tokens it produces onto and consumes from channels. The number of phases is finite: after the actor has completed its last phase, it returns to its first phase again.

Following [11], we use the term *period* to refer to the number of phases of an actor[1], and denote the period of actor v by φ_v. The phase of an actor can be derived from its *firing index* (i.e., the index in the sequence of all firings of that actor) in a straightforward way: the behaviour of the actor during its k^{th} firing (with $k = 1$ being the index of the first firing) is given by its $(k \bmod_1 \varphi_v)^{\text{th}}$ phase[2].

Each actor v has an associated *execution time vector*, which we denote by $T_v = [t_1, \dots, t_{\varphi_v}] \in \mathbb{N}^{\varphi_v}$. At the start of an execution, an actor consumes data from its incoming channels. The completion of the k^{th} execution occurs *at least* $t_{k \bmod_1 \varphi_v}$ time units after this execution has started, and involves the production of tokens onto its outgoing channels. For the sake of brevity, we write $\tau_v(k)$ to denote the execution time of the k^{th} firing of actor v.

Each channel vw (i.e., the channel from actor v to actor w) has an initial, integer number of tokens, denoted δ_{vw}. Furthermore, with each channel vw, two vectors are associated. These vectors are the channel's production rate vector, denoted $P^+_{vw} = [\rho^+_1, \dots, \rho^+_{\varphi_v}] \in \mathbb{N}^{\varphi_v}$, and consumption rate vector, denoted $P^-_{vw} = [\rho^-_1, \dots, \rho^-_{\varphi_w}] \in \mathbb{N}^{\varphi_w}$. The k^{th} firing of actor v produces $\rho^+_{k \bmod_1 \varphi_v}$ tokens onto the channel, whereas the k^{th} firing of actor w consumes $\rho^-_{k \bmod_1 \varphi_w}$ tokens from the channel. We shall write $\rho^+_{vw}(k)$ and $\rho^-_{vw}(k)$ as respective shorthand notations for $\rho^+_{k \bmod_1 \varphi_v}$ and $\rho^-_{k \bmod_1 \varphi_w}$. We often refer to the *source* v of channel vw as the channel's *producer*, and to the *target* w of vw as its *consumer*. If the number of tokens

[1]The term *period* is overloaded, as it may refer to both the phases of an actor and the *times* at which actors fire.

[2]We write $k \bmod_1 n$ as a shorthand notation for $(k-1) \bmod n + 1$, with the mod operator defined conventionally as: $a \bmod b = a - b \left\lfloor \frac{a}{b} \right\rfloor$.

on each of an actor's incoming channels is at least the channel's consumption rate, the actor may start an execution and is said to be *enabled*.

In the remainder of this thesis, we often need to refer to the total number of tokens produced or consumed by an actor in one period. We therefore denote the number of tokens produced onto channel vw in one period of v by $P^{\Sigma+}_{vw} = \sum_{i=1}^{\varphi_v} \rho^+_{vw}(i)$, and the number of tokens consumed, in one period of w, from channel vw as $P^{\Sigma-}_{vw} = \sum_{i=1}^{\varphi_w} \rho^-_{vw}(i)$. Furthermore, the *greatest common divisor* (gcd) occurs in our transformation algorithms of chapters 4 and 5. We denote the greatest common divisor, of quantities $P^{\Sigma+}_{vw}$ and $P^{\Sigma-}_{vw}$, by g_{vw}.

2.1.1 Multi-Rate SDF

In a multi-rate SDF (MRSDF) graph, actor execution times and production and consumption rates are *scalars* rather than vectors. Multi-rate SDF graphs were the first graphs introduced by Lee and Messerschmidt [60]. When referring to an MRSDF actor's execution time, or the rates associated with an MRSDF channel, we simply omit the parameter k to functions ρ^+_{vw}, ρ^-_{vw} and τ_v. Chapter 4 describes how any CSDF graph may be transformed into an equivalent MRSDF graph.

2.1.2 Homogeneous SDF

In a homogeneous SDF (HSDF) graph, production and consumption rates are all equal to one. Homogeneous SDF graphs were studied by Reiter in 1968 [78], as a restricted version of the more general computation graphs introduced two years earlier by Karp and Miller [55]. For HSDF graphs, many efficient analysis techniques are available. In Chapter 4 we give transformations from MRSDF and CSDF graphs into equivalent HSDF graphs.

2.1.3 Structural invariants

Cyclic dependencies in a SDF graph limit the frequency at which actors may fire. This maximum frequency depends on the rates and initial tokens associated with the channels that compose these cycles. For a given channel vw, actor w completes, on average, $P^{\Sigma+}_{vw}\varphi_w$ firings for every $P^{\Sigma-}_{vw}\varphi_v$ completed firings of actor v. Let the *gain* of a channel be defined as:

$$\text{gain}_{vw} = \frac{P^{\Sigma-}_{vw}\varphi_v}{P^{\Sigma+}_{vw}\varphi_w},$$

and let the gain of a *path* be the product of the gains of the channels that compose the path.

If the gain of each cycle equals one, then the graph is said to be *consistent*. The firing times of actors in a *consistent* CSDF graph follow a repetitive pattern [11]. If, furthermore, the graph is strongly connected, then the number of tokens that accumulates on a channel during execution, is bounded [40].

For a consistent CSDF graph, a minimal integer vector q exists such that, for every channel vw, the following, so-called *balance equation* holds:

$$q_v \frac{P_{vw}^{\Sigma+}}{\varphi_v} = q_w \frac{P_{vw}^{\Sigma-}}{\varphi_w}. \tag{2.1}$$

with the restriction that for each actor v, q_v is an integer multiple of φ_v [11]. Vector q is commonly referred to as the graph's *repetition vector*, and gives rise to the definition of a *graph iteration*: in a single graph iteration, actor v fires precisely q_v times. Note that for an MRSDF graph, the repetition vector entries must be relatively prime (if not, then the vector cannot be minimal as a common divisor can be divided out). For CSDF graphs the repetition vector entries are not necessarily relatively prime.

As a dual to the repetition vector, a consistent CSDF graph has a second structural invariant, which is associated with *channels* rather than actors [93]. For a consistent CSDF graph $\mathcal{G}$, a minimal integer vector s, with an entry for each channel in $\mathcal{G}$, exists, such that, for each *actor* v, the following, so-called *flow conservation equation* holds for each pair of incoming and outgoing channels, uv and vw of v:

$$s_{uv} \frac{P_{uv}^{\Sigma-}}{\varphi_v} = s_{vw} \frac{P_{vw}^{\Sigma+}}{\varphi_v}, \tag{2.2}$$

with the restriction that for each actor v, both sides of the equation are integer. Vector s is commonly referred to as a *P-semiflow* in the context of Petri Nets [93]. In the context of dataflow graphs, we refer to it as *flow normalisation vector*. The flow normalisation vector gives the ratios between the number of tokens that flows through channels in a single iteration. Furthermore, vector s may be regarded as an assignment of weights to channels. The weighted number of tokens on a channel vw is then given by $s_{vw}\delta_{vw}$. In any cycle of a consistent SDF graph, the total of the weighted number of tokens is left unchanged by firings [64, 93].

In a single iteration of a consistent CSDF graph $\mathcal{G}$, the weighted number of tokens produced onto a channel vw in $\mathcal{G}$ is given by:

$$\mathcal{N}_{\mathcal{G}} = q_v s_{vw} \frac{P_{vw}^{\Sigma+}}{\varphi_v}. \tag{2.3}$$

By definition of the repetition and flow normalisation vectors, this quantity is the same for every channel in the graph. We refer to $\mathcal{N}_{\mathcal{G}}$ as the *scalar invariant* associated with $\mathcal{G}$. We use the scalar invariant as a scaling factor in the construction of the single-rate approximations in Chapter 5.

We conclude this section with an illustration of the structural invariants, using an example CSDF graph, depicted in Figure 2.2. Each of the two actors in the graph has a period of two: $\varphi_v = \varphi_w = 2$. A repetition vector q that satisfies the balance equations is given by $q_v = 4, q_w = 6$. In a single graph iteration, actor v completes two, and actor w three periods. This means that in a single iteration, v produces

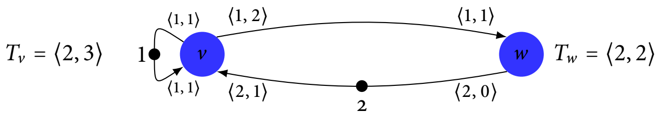


Figure 2.2 – An example of a consistent CSDF graph

six tokens onto channel vw, and four tokens onto the self-loop vv. Furthermore, actor w consumes six tokens from channel vw, and produces six tokens onto wv. The smallest integer vector s that satisfies the flow conservation equations is given by $s_{vv} = 3$ and $s_{vw} = s_{wv} = 2$. If we apply these normalisation factors to the corresponding channel, then, in a single iteration, on each channel, twelve tokens are transferred from producer to consumer. As a result, the scalar invariant of the graph is twelve.

2.1.4 Functional determinacy

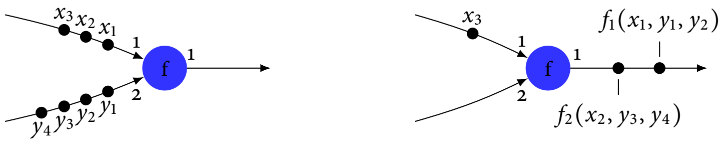
Synchronous dataflow graphs model applications that process *streams* of data. A stream represents the sequence of tokens that are transferred over a channel in an SDF graph. Since an actor in an SDF graph maps input data to output data, a *sequence of firings* of an actor maps sequences of input data to sequences of output data. Such a sequence is called a *dataflow process* [61, 76]. Dataflow processes thus generalise the notion of a *single* actor firing to *sequences* of actor firings.

In this thesis, we restrict the analysis of SDF graphs to those executions that are *functionally determinate*. That is, if the same sequence of input tokens is consumed by a dataflow process, it produces the same sequence of output tokens [53]. A sufficient condition for a dataflow process to be functionally determinate is that each actor firing is *functional*, and that firings are strictly ordered [61, 75]. This means that the data produced by a single firing is a function of the data it consumes, and that the mapping between the input and output sequences is *(prefix-)monotonic*: given a prefix of the input sequence, part of the output sequence may already be computed (see Figure 2.3).

The above is captured by the concept of a *monotonic dataflow process*. In a monotonic dataflow process, firings ensure prefix-monotonicity: an actor produces tokens in the same order as their corresponding input tokens are consumed. We describe this correspondence in more detail in Chapter 3.

2.1.5 Auto-concurrency

Actors in an SDF graph may fire as soon as sufficient input data is available. Availability of data may be sufficient for multiple firings, which may start simultaneously (note that the consumption (and production) of tokens from a channel is instantaneous). Simultaneous firings of the same actor are called *auto-concurrent*. If each firing takes the same amount of time, then firings that have started simultaneously



(a) Actor f with two enabled firings.

(b) Two firings have completed.

Figure 2.3 – Any sequence of actor firings is *functional*: the sequence of data produced by them is a function of the sequence of data they consume. This means that a sequence of actor firings is (prefix-)*monotonic*: given a prefix of the input sequence, part of the output sequence may already be computed. Monotonicity implies a *strict* ordering on an actor's firings; tokens consumed later correspond to tokens that are produced later.

also complete simultaneously. Self-loops (i.e., cycles consisting of a single channel) are commonly used to explicitly limit the degree of auto-concurrency.

To ensure functional determinacy, a firing must delay the production of its output tokens until all firings that functionally precede[3] it have produced their output tokens. No such delay is necessary for MRSDF actors: as each firing of an MRSDF actor takes the same amount of time, a firing that has started earlier than another firing, completes earlier as well. Consequently, auto-concurrency of actors cannot destroy functional determinacy. Things are different for CSDF actors, as their *execution time* varies cyclically. As a result, a firing may, even though it has started later, complete earlier than another firing that precedes it. This destroys prefix monotonicity, and consequently, functional determinacy. To illustrate this, consider again Figure 2.3, and assume that the second firing of f finishes before the first firing does. The situation after two firings of f now differs from Figure 2.3(b), in that the order of the two tokens on the outgoing channel of f is reversed. The mapping from input sequences to output sequences, of the dataflow process associated with f, is thus dependent on the timing of f, which means it is not prefix-monotonic.

To solve this problem, the original semantics of CSDF, as presented in [11, 33], implicitly assumes that each actor has a self-loop, i.e., a channel with rates set to one, and a single initial token. Such a self-loop prevents the actor to start multiple, concurrent, executions. This so-called auto-concurrency is commonly available to MRSDF and HSDF actors. The motivation for restricting auto-concurrency in CSDF is that successive phases of an actor should not overlap, as phases are assumed to imply the presence of internal state, which, in dataflow semantics, gives a sequential execution. Assuming implicit self-loops for *every* actor (including those with a single "phase") in the graph, however, unnecessarily limits the expressiveness

[3]The word "precedes" is slightly ambiguous, as it may refer both to *time* or *ordering*. We distinguish between these two meanings using the adverbs *temporally* and *functionally*.

of CSDF graphs. This breaks the taxonomy that is implied by the generalisation of production and consumption rate scalars to vectors, as offered by CSDF: any MRSDF graph that has auto-concurrent actors is, under the definition of [11], not a CSDF graph.

Some authors therefore choose to relax the original CSDF restriction of [11], by assuming an implicit self-loop only for those actors that have phases with *different* associated execution times [75, 98]. Others allow for auto-concurrent execution of firings that have different associated execution times, but take no measures to guarantee functional determinacy [91]. In this thesis, we choose to allow auto-concurrent actors to be included in a CSDF graph, regardless of their execution time vectors, and enforce functionally determinate execution by including dedicated constraints on the times at which actors may complete their firings. These constraints are described in more detail in Chapter 3.

2.1.6 Self-timed execution and throughput

In an SDF graph, execution is data-driven: an actor may fire as soon as the specified amount of data (tokens) is available on each of its incoming edges. An actor that is ready to fire is said to be *enabled*. In a *self-timed execution* of the graph, each actor fires as soon as it is enabled.

In a consistent SDF graph, a self-timed execution eventually settles in a repetitive pattern, called *periodic phase* [42]. In the periodic phase (of a self-timed execution), each actor fires at a constant *average* rate[4]. Because in general, actors have different production and consumption rates, the rates at which they fire. The ratio between the (constant) rates at which two different actors fire is given by their repetition vector entries.

The *throughput* of a consistent SDF graph $\mathcal{G}$, denoted $Th(\mathcal{G})$, is equal to the average number of iterations completed per time unit, in a self-timed execution. Formally, if we let $t_v(k)$ denote the time at which an actor v in $\mathcal{G}$ completes its k^{th} firing, then the throughput of $\mathcal{G}$ satisfies:

$$Th(\mathcal{G}) = \lim_{k\to\infty} \frac{q_v k}{t_v(k)}, \tag{2.4}$$

where q_v is the entry, in the repetition vector of $\mathcal{G}$, of actor v.

In general, the self-timed execution of an arbitrary SDF graph does not immediately enter the periodic regime [42]. The phase that precedes the periodic phase is referred to as the *transient phase* [42, 49]. The transient phase of an SDF graph may be very long; only very pessimistic bounds for the length of the transient phase exist [49]. To reduce the length of the transient phase, firings of a cleverly

[4]The word *rate* here is used in two different contexts: as a *frequency*, to refer to the number of firings that an actor completes per time unit, and as a *quantum*, to indicate the number of tokens produced or consumed by a firing.

selected set of vertices may be simulated prior to starting the self-timed execution. This technique is called *retiming* and changes the initial token distribution of the graph [73, 103].

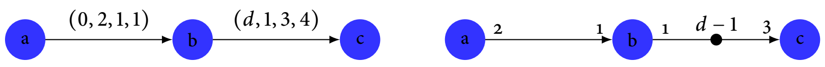
2.1.7 Related Models

Synchronous dataflow graphs are often regarded as specific instances of *computation graphs*, and are also closely related to *Petri nets*. The use of timed event graphs for the performance evaluation of real-time systems has been proposed in [21, 50]. Furthermore, various extensions have been made to the SDF model introduced in [60], in order to capture properties of specific kinds of applications. Results obtained for these related models may, either directly or indirectly often be applied to SDF graphs. We therefore discuss several related models in the following sections.

Computation Graphs

One of the earliest models for computations were introduced by Karp and Miller in 1966. This model was named *computation graph*, and represents computations as directed graphs [55]. For computation graphs, one may compute whether the computation terminates, or whether it requires bounded memory [55]. These notions are similar to those of liveness and boundedness for SDF graphs described in [40]. In a computation graph, functions are represented by vertices, and arcs indicate which functions are *composed*: the output of the arc's source is used as an input of the arc's sink. In a computation graph, with each arc four (integral) numbers are associated: an initial number of 'tokens' present on the arc, two integers that respectively give the number of tokens *produced* by the (function representing the) arc's head and *consumed* by the arc's tail, and a *threshold*, which gives the minimum number of tokens that must be on the arc in order for the tail to consume. The multi-rate SDF graphs introduced by Lee in 1986 are a special case of these computation graphs, where the threshold is equal to the number of consumed tokens. However, computation graphs can be generally represented by equivalent MRSDF graphs, as is shown in [12]. A threshold, which is greater than the number of consumed tokens, translates to a reduction of the initial number of tokens present on the MRSDF channel. This reduction is equal to the difference between the threshold and the consumed number of tokens. Figure 2.4 shows an example computation graph, and an equivalent MRSDF graph.

As a generalisation of the computation graphs of Karp and Miller, *phased computation graphs* have been proposed [96]. Phased computation graphs extend CSDF graphs with thresholds and initial states: In phased computation graphs, the behaviour of an actor cyclically varies, similar to CSDF. However, prior to the cyclic pattern, an actor goes to a predefined number of initial states. Furthermore, as in computation graphs, an actor may not fire before the number of tokens on each incoming channel is at least the channel's threshold. These thresholds are similar to those found in computation graphs, but now vary per phase. Although the initial states prevent phased computation graphs from being equivalently represented by



(a) Computation graph. (b) MRSDF graph.

Figure 2.4 – A computation graph and an equivalent MRSDF graph. Thresholds greater than the number of consumed tokens are accounted for by subtracting the difference from the initial number of tokens [12].

CSDF graph, these initial states may simply be simulated, leaving a graph with just the thresholds remaining.

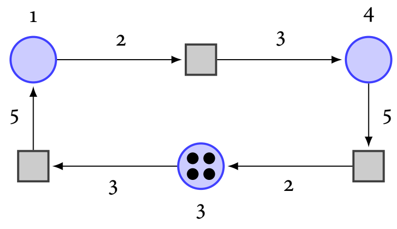
MRSDF graphs are often referred to as special cases of computation graphs, and phased computation graphs as generalisations of CSDF graphs. However, when taking a system-theoretic perspective, as is done in this thesis, the inferred differences between the classes of graphs vanish. The translation from computation graphs to MRSDF graphs, illustrated above and presented in [12], fits this view. In Chapter 4, we illustrate how such a perspective allows phased computation graphs to be represented by equivalent MRSDF graphs.

Petri Nets

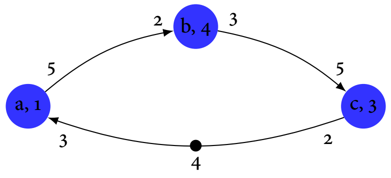
The most established model that is related to SDF is the *Petri net*. The subclass of so-called *bounded and conflict-free* Petri-nets consists of those networks that can be modeled by max-plus linear recurrence relations [49]. A Petri net is a network (i.e., a directed graph), which consist of *places* and *transitions*. Transitions are analogous to actors found in SDF, and places map to channels. Analogous to SDF graphs, transitions may *fire*, which involves moving tokens from upstream places to downstream places. Time may be included in the model by assigning *holding times* to places. A holding time denotes the minimum time a token must remain in a place, before it can be moved to a downstream place by the firing of a transition.

Weighted versions of these nets exist, where a place and transition may be connected by several arcs. This indicates that multiple tokens are produced into and consumed from the place. Figure 2.5 depicts a weighted Petri net. The gray squares are the net's transitions, the blue circles are places. The four black dots inside the center place denote initial tokens, similar to SDF graphs. Each arc that connects a place and a transition is annotated with an integer that indicates the number of tokens that are moved (from place to transition, or vice-versa) by a firing of the transition. Furthermore, each place is annotated by an integer specifying the holding time. For example, each token that is moved into the upper left place must remain in the place for at least one time unit, before it may be moved to the upper right place.

Multiple names are used in literature to refer to these weighted versions of Petri



(a) A weighted Petri net.



(b) Equivalent MRSDF graph.

Figure 2.5 – A weighted Petri net (taken from [93]), and an equivalent MRSDF graph. The holding time of each place is moved to its upstream transition.

Nets: weighted timed event graphs [5, 66], weighted marked graphs [18, 71, 83], timed event graphs with multipliers [23, 46], or weighted T-Systems [9, 34, 93]. Well-known concepts used in dataflow sometimes have different names in Petri net literature. For example, the notion of *consistency* translates to so-called *weight-balanced timed event graphs* in [26, 27] and *unitary graphs* in [65].

Any weighted Petri-net may be transformed into an equivalent MRSDF graph. An example transformation is illustrated in Figure 2.5. In the figure, the four tokens present in the center place map to four initial tokens on the channel. Whereas in the Petri net, tokens must remain in a place for a specified holding time, in the MRSDF graph, tokens remain "inside an actor" for a duration specified by actor's execution time (see Section 2.1).

Related dataflow Graphs

In the definition given by Lee in [60], the word *synchronous* in synchronous dataflow refers to the fact that the ratio between the rates at which any two actors fire is a rational number. There are several classes of dataflow graphs for which this definition does not apply, but which still may be *conservatively* abstracted by SDF graphs.

Many stream-processing applications are dynamic in the sense that both their functional and non-functional behaviour depends on the data that is processed. Different values of data invoke different modes of operation. Nevertheless, for each class of data, a fixed subset of tasks are carried out in a predefined order. This idea is exploited in *Scenario-Aware dataflow* (SADF), which distinguishes between *deterministic* processing of data in *scenarios*, and non-deterministic control, modelled by automata [36, 94]. The SADF model allows for conservative performance analysis in the presence of non-determinism [37, 87]. Methods for automatically detecting the most important variables from multimedia applications, using them to select and dynamically predict scenarios, have been proposed in literature [43].

In Affine Data-Flow (ADF) graphs, the firing times of actors are characterised by a phase and a period [16]. The model provides a conservative abstraction of SDF graphs in a way similar to the compositional temporal analysis model presented in [48]. Although this thesis is restricted to the class of CSDF graph, the transformations and approximations presented in Chapters 4 and 5 may be tailored to various extensions of the basic SDF model.

2.2 Discrete event systems

Synchronous dataflow graphs form a small part of the broad class of *discrete event systems* [17, 22, 49]. In a discrete event system, a finite number of resources (e.g., processors or machines) is shared by several users (e.g., jobs, data or manufactured parts), which all contribute to the achievement of a common goal (e.g., a parallel computation of a function applied to input data, or the assembly of a product in an automated manufacturing line).

An event is an instantaneous occurrence, such as the beginning of a computation, the production of a token or the completion of an execution. Whereas conventional systems such as mechanical or electronic ones describe how a force, current or voltage evolves over *time*, a discrete event system describes the evolution of time over events. In this view, the start or execution of a particular firing of an SDF actor, or the production of a token onto an SDF channel, may all be regarded as events.

The dynamics of discrete event systems can be described using the primitives of *synchronisation* and *delay*: synchronisation requires that several resources or users are available at the same time (e.g., the presence of all required input data), and delay captures the fact that using a resource (e.g., computing the function) takes a certain amount of time. Synchronisation may be expressed as the *maximum* (max) of the times at which the involved events occur, and a delay with respect to an event may simply be expressed by *adding* (+) a constant to the event's timestamp. These operators are provided by *max-plus algebra*. In this section, we give a brief description of max-plus algebra and show how max-plus algebra is used to describe the temporal dynamics of discrete event systems.

2.2.1 Max-plus algebra

Max-plus algebra is a mathematical structure named *semiring*, endowed with two operations: max and $+$. The algebraic properties of the operators max and $+$ resemble those of (respectively) addition and multiplication in conventional non-abstract algebra. This allows techniques that are known for (the analysis of) systems in conventional algebra to be applied to systems in max-plus algebra. To emphasise these similarities, operations $\otimes$ and $\oplus$ are commonly defined as follows, where $\otimes$ has priority over ⊕:

$$x \oplus y \overset{\text{def}}{=} \max(x, y), \tag{2.5}$$

$$x \otimes y \overset{\text{def}}{=} x + y. \tag{2.6}$$

The properties of these operators are similar to their counterparts, + and × in conventional algebra. They are associative and commuative, and ⊗ distributes over ⊕ [22, 49]. The respective max-plus counterparts of *one* and *zero*, which, in conventional algebra, are the unit elements of multiplication and addition, are 0 and $-\infty$:

$$x \oplus -\infty = \max(x, -\infty) = x,$$
$$x \otimes 0 = x + 0 = x.$$

Following convention, we denote $-\infty$ by ε. The set of the real numbers, extended with ε, is defined as $\mathbb{R}_{\max} \overset{\text{def}}{=} \mathbb{R} \cup \{\varepsilon\}$. Max-plus algebra is then formally denoted as the tuple:

$$\mathcal{R}_{\max} = (\mathbb{R}_{\max}, \oplus, \otimes, \varepsilon, 0), \tag{2.7}$$

An in-depth treatment of max-plus algebra is beyond the scope of this thesis. Nevertheless, in order to reason about SDF graphs as mathematical structures, an understanding of some basic concepts is necessary. This section provides a brief introduction to max-plus algebra, borrowing most of the notation from [49] and [22].

2.2.2 Vectors and matrices

Operations ⊕ and ⊗ can be extended to vectors and matrices in a straightforward way. We denote the set of $n \times m$ matrices with elements in $\mathbb{R}_{\max}$ by $\mathbb{R}_{\max}^{n \times m}$. A matrix $A \in \mathbb{R}_{\max}^{n \times m}$ has n rows and m columns. The element in row i and column j of A is denoted a_{ij}, or, alternatively, $[A]_{ij}$. We denote the sum $C \in \mathbb{R}_{\max}^{n \times m}$ of two matrices $A, B \in \mathbb{R}_{\max}^{n \times m}$ by $C = A \oplus B$, with C defined as:

$$c_{ij} = a_{ij} \oplus b_{ij}. \tag{2.8}$$

The product of matrices $A \in \mathbb{R}_{\max}^{n \times m}$ and $B \in \mathbb{R}_{\max}^{m \times p}$ is an $n \times p$ matrix, defined as

$$[A \otimes B]_{ij} = \bigoplus_{k=1}^{m} a_{ik} \otimes b_{kj}, \tag{2.9}$$

where $\bigoplus$ denotes summation in $\mathcal{R}_{\max}$, analogous to the summation symbol Σ in conventional algebra.

The equivalents of the elements 0 and ε in $\mathcal{R}_{\max}$ for matrices are denoted $\mathcal{E}$ and E, where $\mathcal{E}(k, m) \in \mathbb{R}_{\max}^{k \times m}$ is a matrix with all elements equal to zero (ε), and

$E(n) \in \mathbb{R}_{\max}^{n \times n}$ is a square matrix with one (i.e., e) on the main diagonal and zero (ε) elsewhere.

Finally, the k^{th} power of a square matrix $A \in \mathbb{R}_{\max}^{n \times n}$ is denoted $A^{\otimes k}$, and is recursively defined for $k \in \mathbb{N}$ by:

$$A^{\otimes 0} = E(n) \tag{2.10}$$

$$A^{\otimes k} = A \otimes A^{\otimes k-1}. \tag{2.11}$$

2.2.3 Linear dynamical max-plus systems

The dynamics of a discrete event system can be described in two related ways. First of all, one may describe the evolution of the number of times an event of a specific kind occurs, over time. Analogously, one may describe how time evolves over events. The first approach involves the usage of operators min and +: for example, the number of times an actor in an SDF graph may fire depends on the minimum of the number of times producing actors have fired, plus some delay. In the second approach, one interrelates the times at which events may occur, using operators max and +. The resulting system of recurrence relations is a linear dynamical max-plus system. The notion of time in a linear dynamical max-plus system differs from the notion of time found in conventional systems. For example, in the former kind of system, $u(i) = t$ could state that the i^{th} firing of a source actor in an SDF graph occurs at time t, whereas in a conventional system, $u(i)$ could refer to the amplitude of a signal at time i.

Recurrence relations in $\mathcal{R}_{\max}$ recursively specify the times at which events occur, and are commonly named *dater equations*. The following is an example of a set of dater equations:

$$\begin{aligned} a(k) &= a(k-1) \otimes 1 \oplus c(k-2) \otimes 2, \\ b(k) &= a(k) \otimes 2 \oplus a(k-1) \otimes 7 \oplus u(k-1) \otimes 1, \\ c(k) &= a(k-1) \otimes 2 \oplus b(k-1) \otimes 6, \\ y(k) &= c(k). \end{aligned} \tag{2.12}$$

This set of equations describes how the times at which five different kinds of events, named a, b, c, u, and y, are interrelated. The time of the k^{th} occurrence of the event of kind a is denoted $a(k)$, where the parameter k is commonly referred to as a *counter variable*. Thus, where conventional systems map *time* to a domain-specific quantity such as amplitude, force, or voltage, a max-plus system maps counters to time. Note that, although it may be tempting to refer to $x(k)$ as (the time of) the k^{th} occurrence of x, it is not implied that the $(k+1)^{\text{th}}$ occurrence of event x occurs later than its k^{th} occurrence (i.e., it is not implied that $x(k+1) \geq x(k)$).

The recurrence relations given above specify the interdependencies between the different kinds of events. The first relation, for example, states that $a(k)$ may occur no sooner than two time units after event $c(k-2)$. It furthermore states that there must be at least one time unit between two successive occurrences of a. The last equation states that events of kinds c and y occur simultaneously. Furthermore, the interdependencies of a, b and c are *cyclic*: a depends on c, c depends on b (and a), and b, in turn, depends again on a. No event depends on y, and all events, either directly or indirectly, depend on u.

If we collect the three variables a, b and c from this example in a vector $x = (a\,b\,c)^T$, then we may write (2.12) as the following two max-plus *linear sums* over matrices:

$$x(k) = \bigoplus_{m=0}^{2} A_m \otimes x(k-m) \oplus B \otimes u(k-1), \tag{2.13a}$$

$$y(k) = C \otimes x(k), \tag{2.13b}$$

where matrices A_m, B and C are defined as

$$A_0 = \begin{pmatrix} \varepsilon & \varepsilon & \varepsilon \\ 2 & \varepsilon & \varepsilon \\ \varepsilon & \varepsilon & \varepsilon \end{pmatrix} \qquad A_1 = \begin{pmatrix} 1 & \varepsilon & \varepsilon \\ 7 & \varepsilon & \varepsilon \\ 2 & 6 & \varepsilon \end{pmatrix} \qquad A_2 = \begin{pmatrix} \varepsilon & \varepsilon & 2 \\ \varepsilon & \varepsilon & \varepsilon \\ \varepsilon & \varepsilon & \varepsilon \end{pmatrix}$$

$$B = \begin{pmatrix} \varepsilon \\ 1 \\ \varepsilon \end{pmatrix} \qquad C = \begin{pmatrix} \varepsilon & \varepsilon & 0 \end{pmatrix}$$

The equations listed in (2.12) are examples of *higher-order* recurrence relations. An M^{th}-order recurrence relation in $\mathcal{R}_{\max}$ as an expression that has the form

$$t(k) = \bigoplus_{m=0}^{M} A_m \otimes t(k-m). \tag{2.14}$$

Higher-order recurrence relations, in which no event depends on itself, can be transformed into a first-order relation, which is a relation of the form $t(k) = A \otimes t(k-1)$ [22, 49]. First-order relations are conveniently used to express linear systems in the following *state-space representation*

$$x(k+1) = A(k) \otimes x(k) \oplus B(k) \otimes u(k), \tag{2.15a}$$

$$y(k) = C(k) \otimes x(k) \oplus D(k) \otimes u(k), \tag{2.15b}$$

where vectors x, u and y are respectively referred to as the system's *state*, *input* and *output*. Matrices A, B, C and D are commonly named the system's *dynamics*, *input*, *output* and *feed-through* matrices (in the example system (2.13) the feed-through matrix is the zero matrix). The timing of output events depends on the timing of state and input events, and state events depend on the previous state(s) and input events.

Throughout this thesis, we shall refer to a system either by its state-space representation, or through higher-order recurrence relations. The form we use depends on whichever is more natural in the given context: when referring to the system as a whole, we do so by its state-space representation. We use a higher-order recurrence relation when we need to refer to the dependencies of a specific kind of event, as we did for example in (2.12).

We conclude this section with an example. To develop an intuition for the behaviour of a discrete event system, we examine the evolution of event times that emerges from the recurrence relation. As all events depend on (the input) u, we let $u(k) = -\infty$ for $k \leq 0$, and $u(k) = 3(k-1)$ for $k \geq 1$. Furthermore, let the initial event times be given by $a(k) = b(k) = c(k) = -\infty$ for $k \leq 0$. As a result, we also have $y(k) = \varepsilon$ for $k \leq 0$. Using the recurrence relations given by (2.12), we may now compute how the timing of the events evolves. This gives:

$$\begin{aligned}
a(1) &= a(0) \otimes 1 \oplus c(-1) \otimes 2 = \varepsilon, \\
b(1) &= a(1) \otimes 2 \oplus a(0) \otimes 7 \oplus u(0) \otimes 1 = \varepsilon, \\
c(1) &= a(0) \otimes 2 \oplus b(0) \otimes 6 = \varepsilon,
\end{aligned}$$

and thus $y(1) = \varepsilon$. By iterative evaluation of the relations, we may compute the times of subsequent events. This results in the following sequence (we again let vector $x(k)$ denote $(a(k)\, b(k)\, c(k))^T$):

$$x(2) = \begin{pmatrix} \varepsilon \\ 1 \\ \varepsilon \end{pmatrix} \quad x(3) = \begin{pmatrix} \varepsilon \\ 4 \\ 7 \end{pmatrix} \quad x(4) = \begin{pmatrix} \varepsilon \\ 7 \\ 10 \end{pmatrix} \quad x(5) = \begin{pmatrix} 9 \\ 11 \\ 13 \end{pmatrix} \quad x(6) = \begin{pmatrix} 12 \\ 16 \\ 17 \end{pmatrix}$$

The first non-zero output occurs at $y(3) = 7$. The two subsequent output events (i.e., $y(4)$ and $y(5)$) occur at intervals of *three* time units, which corresponds to the time between two successive input events, $u(k)$ and $u(k+1)$. Event $y(6)$, however, occurs *four* time units after $y(5)$. This is due to the events that make up the system's state, which fail to keep up with the pace of the input events. Consequently, the time between input and output events, $u(k)$ and $y(k)$, increases steadily as k increases. This relation is depicted in Figure 2.6. In the following section, the limiting behaviour of the state on the output is discussed in more detail.

2.2.4 Spectral theory and scheduling

Classical system theory studies the input-output behaviour of dynamical systems. In a discrete event system, input and output refer to the *times* at which specific events occur, e.g., the arrival times of resources used in a manufacturing plant or the completion time of a parallel computation. Studying the input-output behaviour of a discrete event system involves computing both the *delay* between an input and its corresponding output, and the *rate* or pace at which output events may occur.

The system that served as an example in the previous section, illustrates that the maximum rate at which output events occur, is limited by the rate at which state

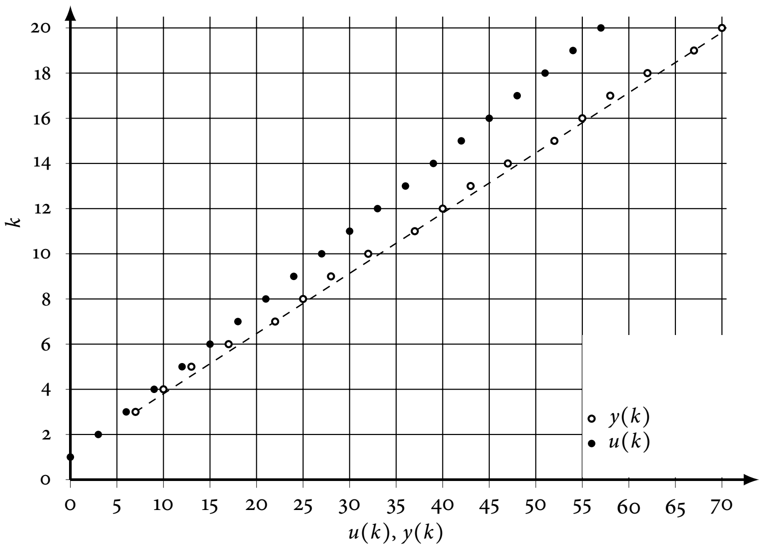


Figure 2.6 – The evolution of the times at which input and output events, $u(k)$ and $y(k)$, occur for the example max-plus system (2.12). The dashed line indicates the average pace at which output events occur in the long term.

events occur. Computing this rate involves studying the *autonomous* part of the system, which is the linear dynamical system given by

$$t(k+1) = A(k) \otimes t(k). \tag{2.16}$$

The long-term behaviour of the autonomous system limits the rate at which output events may occur [22, 49]. In particular, the *eigenvalue* of A corresponds to the shortest possible average time between subsequent events of any kind. An *eigenvector* associated with an eigenvalue gives the times at which events should occur, such that subsequent events occur at this maximum rate. Formally, an eigenvector of a square matrix A is a vector v, such that A maps v onto a scalar multiple of v. Let $A \in \mathbb{R}_{\max}^{n\times n}$ be a square matrix, $v \in \mathbb{R}_{\max}^{n}$ be an n–vector and $\lambda \in \mathbb{R}_{\max}$ be a scalar, such that

$$A \otimes v = \lambda \otimes v. \tag{2.17}$$

The scalar λ is an eigenvalue of matrix A. Note that, as is the case in conventional algebra, if v is an eigenvector, then any scalar multiple $a \otimes v$ of v is an eigenvector as well. Since an eigenvalue is associated with an eigenvector, it is natural to consider them as a *pair*. We call the pair $(v, \lambda) \in \mathbb{R}_{\max}^{n} \times \mathbb{R}_{\max}$ an *eigenmode* of matrix $A \in \mathbb{R}_{\max}^{n\times n}$, if v and λ satisfy Equation (2.17).

An eigenvector gives rise to a *strictly periodic* sequence of state vectors, in the following way. If we choose to let the initial events occur at timestamps given by an eigenvector v of A (that is, we set $t(0) = v$), then we have:

$$\begin{aligned} t(k) &= A \otimes t(k-1) = A^{\otimes 2} \otimes t(k-2) = \ldots = A^{\otimes k} t(0) \\ &= \lambda^{\otimes k} \otimes t(0), \end{aligned}$$

and straightforward induction gives $t(k+1) = \lambda \otimes t(k)$. In conventional algebra, this means that the event times follow a strictly periodic pattern, with period λ:

$$t_v(k+1) = t_v(k) + \lambda.$$

Eigenvalues and eigenvectors of max-plus systems are closely related to the notions of *throughput* and *schedules*, which are common terms in SDF parlance. A periodic sequence of events that is obtained from an eigenvector in the way described above, is analogous to the notion of a *self-timed schedule* for SDF graphs. Self-timedness refers to the fact that each firing of an actor occurs as soon as possible. The time of the initial firing of an actor is thus fully determined by the availability of sufficient tokens on its incoming channels [40, 69]. The crucial difference between this notion of a self-timed schedule and the sequence of event times obtained from an eigenvector is that, in the former, the firing times of actors to not necessarily follow a *strictly periodic* pattern [22, 41, 49], whereas in the latter they do.

To avoid confusion with existing terms in SDF literature, we use the term *eigenschedule* to refer to the strictly periodic sequence of events obtained from an eigenvector. In an eigenschedule, the *rate* at which each event occurs is equal to λ^{-1}. The reciprocal of the eigenvalue thus gives the *throughput* of the system, and this is the maximum rate that any schedule can attain.

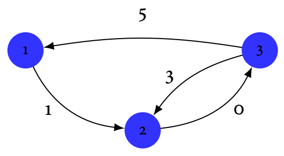
2.2.5 Graphical representations

There exists a strong relationship between square matrices in $\mathbb{R}_{\max}^{n\times n}$ on the one hand, and *weighted digraphs* on the other hand. Any square matrix may be represented by a weighted digraph, and vice versa. We denote by $\mathcal{G}(A)$ the *graph representation* of (square) matrix A. Graph $\mathcal{G}(A)$ is a weighted directed graph that has a vertex for every row (or column) in the matrix. An intuitive interpretation of the relation between A and $\mathcal{G}(A)$ is to regard the former as being the *distance matrix* of the latter: edge ij is in the graph, with *weight* $w(ij) = A_{ji}$, if and only if matrix element A_{ji} is non-zero (i.e., in the max-plus sense: not equal to ε). An example of a matrix and its graph representation is given in Figure 2.7.

Computing the k^{th} power of a square matrix in max-plus algebra has an intuitive graph-theoretical interpretation. To illustrate this, consider the matrix $B = A^{\otimes 2}$. The elements of B are given by

$$b_{ij} = \bigoplus_{k=1}^{n} a_{ik} \otimes a_{kj}.$$

$$A = \begin{pmatrix} \varepsilon & \varepsilon & 5 \\ 1 & \varepsilon & 3 \\ \varepsilon & 0 & \varepsilon \end{pmatrix}$$



$$A^2 = \begin{pmatrix} \varepsilon & 5 & \varepsilon \\ \varepsilon & 3 & 6 \\ 1 & \varepsilon & 3 \end{pmatrix}$$

Figure 2.7 – A square matrix, its graph representation, and the square of the matrix, which indicates the weight of the longest paths of two arcs.

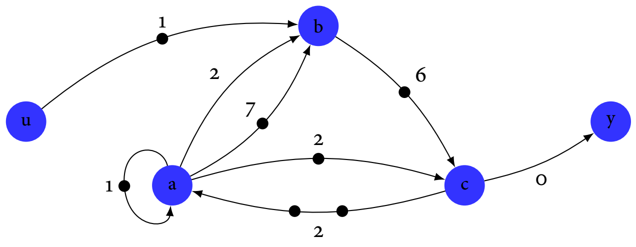


Figure 2.8 – The example max-plus system (2.12), represented as a marked, weighted, directed multigraph. Vertices a, b and c jointly denote the state of the system, vertices u and y respectively represent the system's input and output.

In words, element b_{ij} gives the longest path of length 2 from vertex j to vertex i. Max-plus algebraic matrix exponentiation is thus analogous to the computation of longest paths (of a fixed length) in the graph representation of the matrix. As a result, taking the max-plus sum of the n powers of an $n \times n$ matrix A essentially computes, for every pair of vertices i and j, the *longest path* in $\mathcal{G}(A)$ from i to j.

Graphical representations of matrices extend naturally to *higher-order recurrence relations*. We denote the graph representation of an M^{th}-order relation (see (2.14)) by $\mathcal{G}(A_0, \ldots, A_M)$. Edges in this graph have a weight and an integer *marking*. The marking of an edge is represented by a corresponding number of *tokens*, denoted by black dots. Furthermore, the graph $\mathcal{G}(A_0, \ldots, A_M)$ is a *multigraph*: multiple edges may exist between two vertices. The name for this kind of graph is a *marked weighted directed multigraph*, or simply a marked graph [25]. Such a graph is equivalent to a *timed event graph* in the context of Petri Nets, where each place has precisely one upstream and one downstream transition [24].

A graphical representation of an M^{th}-order recurrence relation is constructed as follows. For every matrix element $[A_m]_{ij}$ with $[A_m]_{ij} \neq \varepsilon$, the graph has an edge $e = ji$. This edge has a weight $w(e) = [A_m]_{ij}$ and marking $\delta_e = m$. Graph $\mathcal{G}(A_0, \ldots, A_M)$ can be regarded as the union of the graph representations $\mathcal{G}(A_i)$ of matrices A_i, where each edge in $\mathcal{G}(A_i)$ is marked with i tokens. As an example, Figure 2.8 depicts the graphical representation of relation (2.12).

Inputs to a max-plus system correspond to *sources* in its graph representation: vertices that have no incoming edges, whereas outputs correspond to *sinks*: vertices that have no outgoing edges. Vertices that are neither source nor sink represent the system's state. We assume that each state variable depends on every other state variable, either directly or indirectly. In terms of the graphical representation of the system, this means that the subgraph corresponding to the state is *strongly connected.*

The eigenvalue of an autonomous system, defined by its higher-order recurrence relation, can be computed from its graph representation. In particular, each cycle in the graph imposes an upper bound on the rate at which events may occur. Define the *cycle ratio* $\lambda(C)$ of a cycle C as:

$$\lambda(C) \overset{\text{def}}{=} \frac{\sum_{vw \in C} w_{vw}}{\sum_{vw \in C} \delta_{vw}}. \tag{2.18}$$

The eigenvalue λ is then equal to the maximum of the graph's cycle ratios:

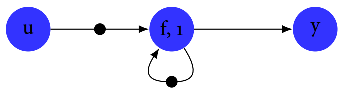
$$\lambda = \max_{C \in \mathcal{G}(A_0, A_1)} \frac{\sum_{e \in C} w_e}{\sum_{e \in C} \delta_e}. \tag{2.19}$$

The right-hand side of (2.19) is referred to as the graph's *maximum cycle ratio* [78]. Many algorithms for computing the maximum cycle ratio of a marked graph exist, see for example [39, 47, 52, 58]. An elegant and efficient algorithm for computing a graph's maximum cycle ratio, in strongly polynomial time, is given by Karp and Orlin in [54] and was later improved by Young, Tarjan and Orlin [101]. Another well-established algorithm is the adaptation of Howard's policy iteration algorithm, and is described in [19]. For an extensive comparison of the performance of these algorithms on a wide range of graphs, see [29, 30].

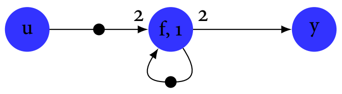
2.2.6 Shift-invariant and shift-varying systems

In conventional algebra, an important class of discrete-time systems is formed by those that are *linear* and *shift-invariant.* Linearity means that the response (i.e., the output) of system to a complex input can be described as the weighted sum of responses to simpler inputs. Shift-invariance refers to the fact that the output of the system does not depend on the particular time the input is applied: delaying the input by k discrete time units gives an identical output, but which is delayed by k time units as well.

To understand how the notion of linearity and shift-invariance applies to discrete event systems, it is important to understand the nature of inputs and outputs of these systems. Inputs and outputs of a linear dynamical max-plus system form an infinite sequence of timestamps. Discrete event systems that can be described by a max-plus *linear* sum over matrices, as given by (2.14), are *linear* [22]. Shifting the input by k corresponds to shifting the sequence by k places. A discrete event system is shift-invariant if shifting the input sequence causes the output sequence to be equally shifted.



(a) A shift-invariant system.



(b) A shift-varying system.

Figure 2.9 – Shift-invariance in discrete event systems: providing the actual input several firings later, results in a corresponding shift of the associated output.

As an example of a shift-invariant system, consider the HSDF graph depicted in Figure 2.9(a). Suppose that the sequence of inputs is the periodic sequence $(1, 2, \ldots)$. That is, the i^{th} firing of input actor u occurs at time i. The corresponding sequence of firing times of output actor y is $(\varepsilon, 2, 3, 4, \ldots)$. Shifting the input by one causes the output sequence to be shifted equally: the input sequence becomes $(\varepsilon, 1, 2, 3 \ldots)$, and the output sequence $(\varepsilon, \varepsilon, 2, 3, 4, \ldots)$.

If we enforce actor f to consume (and produce) tokens in groups of two, we obtain the MRSDF graph shown in Figure 2.9(b). The corresponding system is no longer shift-invariant: to see this, consider the same input sequence as above. Due to the fact that f requires two tokens to be present on its incoming channel, the output sequence is the sequence of even numbers $(2, 4, 6, \ldots)$. Shifting the input by one changes the output sequence in a sequence of *odd* numbers, prefixed by a single zero: $(\varepsilon, 3, 5, 7, \ldots)$. The shifts in input and output sequences do not correspond. Thus, the system corresponding to Figure 2.9(b) is not shift-invariant.

Whether a system is shift-invariant or not is derived from the system's input-output relation, as captured by the *impulse response* of the system [22, 27, 57], which can generally be expressed as:

$$y(l) = \bigoplus_m h(l, m) \otimes u(m).$$

A system is shift-invariant if, in the above representation, we have $h(l + 1, m + 1) = h(l, m)$ [68]. For example, the impulse response of the shift-invariant system corresponding to the graph depicted in Figure 2.9(a) has $h(k, m) = k - m$, for $k > m$. Systems that are not shift-invariant are called *shift-varying*. In general, shift-varying systems are more difficult to analyse than shift-invariant systems. An important subclass of shift-varying systems consists of *periodically* shift-varying systems. In such a system, the amount of shift in the output, caused by shifting the input by k firings, is periodic in k. Examples of linear periodically shift-varying systems are found in the domain of control and signal processing, such as multi-rate sampled-data control systems and multi-rate filter-bank systems [86].

For periodically shift-varying discrete systems with a single input and single output, a pair (L, M) exists such that for all l, m it holds that $h(l+L, m+M) = h(l, m)$ [67,

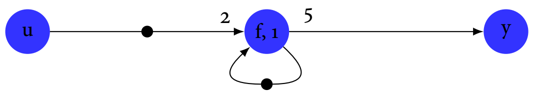


Figure 2.10 – An example of an SDF graph that is described by a (2,5)-shift-invariant linear max plus-system.

68]. As an example, consider the SDF graph depicted in Figure 2.10. In order to derive the impulse response of this system, we first describe the constraints imposed, by the three channels, on the firings of actors f and y, in max-plus algebra. Here, we write $v(k)$ to denote the time at which actor v *completes* its k^{th} firing.

$$\begin{aligned} f(k) &= u(2k-1) \oplus f(k-1) \otimes 1, \\ y(5k-m) &= f(k), \end{aligned}$$

for $0 \leq m < 5$. From these recurrent relations, we may obtain the relation between firing times of output actor y and input actor u, by eliminating the firing times of state actor f. This results in the following relation:

$$\begin{aligned} y(5k-m) &= t_f(k) \\ &= u(2k-1) \oplus t_f(k-1) \otimes 1 \\ &= u(2k-1) \oplus u(2k-3) \otimes 1 \oplus t_f(k-2) \otimes 2 \\ &= u(2k-1) \oplus u(2k-3) \otimes 1 \oplus u(2k-5) \otimes 2 \oplus t_f(k-3) \otimes 3 \\ &= \bigoplus_{i=0} u(2k-1-2i) \otimes i. \end{aligned}$$

If we write the relation between the input and output as:

$$y(l) = \bigoplus_{m} h(l,m) \otimes u(m).$$

We have $h(l+5, m+2) = h(l,m)$ for all $l, m \in \mathbb{Z}$. This means that if an input is shifted by two samples, then the output remains the same except for a shift by five samples. In the above we have restricted ourselves to systems with a single input and a single output (SISO). For generalisations of this multidimensional notion of periodicity, see for example [67].

2.3 Temporal analysis of synchronous dataflow graphs

Literature on the temporal analysis of SDF graphs can be categorised into *exact* and *approximate* methods. Exact analysis of SDF graphs exploits the fact that the self-timed schedule of a consistent graph, after an initial transient phase, follows a repetitive pattern composed of several so-called iterations. Such an iteration may be explicitly represented by transforming the graph into an equivalent HSDF graph, which has a single actor for each individual firing in the iteration [88]. For these HSDF graphs, efficient analysis techniques are available [78]. However, as the length of a single iteration, in the worst case, grows exponentially in the size of the SDF graph, approaches that rely on the construction of an equivalent HSDF graph are often considered too costly for practical use [42].

Approaches that compute a conservative approximation of the temporal behaviour of an SDF graph trade a loss in accuracy for computational efficiency. They assume that the firing times of actors follow relatively simple patterns, for example strictly periodic ones. This results in a (strongly) polynomial time complexity.

In this thesis, we present techniques to transform SDF graphs into equivalent and approximating graphs. These either transform SDF graphs into equivalent HSDF graphs, or derive from them abstractions that are easier to analyse. Transformations play a prominent role in throughput analysis of SDF graphs. The following sections discuss related work on transformations, approximations and throughput analysis.

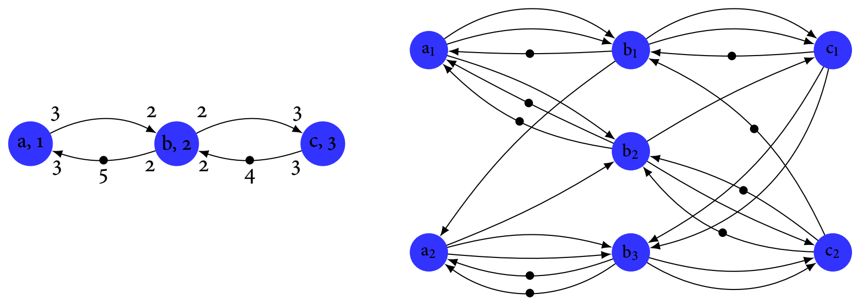
2.3.1 Synchronous dataflow graph transformations

For HSDF graphs, several efficient algorithms for computing a periodic schedule and associated throughput have been described and compared in literature. These algorithms can not be applied to SDF graphs where actors fire at different rates. A straightforward way to approach the analysis of an SDF graph is to *transform* the graph into an equivalent HSDF graph, commonly referred to as its *single-rate* equivalent. The single-rate equivalent is an HSDF graph that has as many actors as a single graph iteration has firings. This transformation thus serves to deliver a simpler kind (i.e., a homogeneous one) of SDF graph, which is *equivalent* in terms of its *temporal behaviour*. As such, analysis results obtained for the single-rate equivalent, such as its throughput, can be translated back to the SDF graph. We discuss several existing transformations in the following two sections.

Transformation of MRSDF graphs

Several transformations of an MRSDF graph into its single-rate equivalent have been described in literature. Along with the introduction of SDF in [60, 62], a transformation of an MRSDF graph into an equivalent HSDF graph was presented. Equivalence here means that any schedule that is admissible for the MRSDF graph is also admissible for the HSDF graph, and vice-versa. The transformation is based on the fact that the ordering of tokens must be maintained: if an actor v produces k tokens onto arc vw, then in the equivalent HSDF graph this particular firing of v has k outgoing arcs; one for each token. These arcs connect the producer of the corresponding token with the token's consuming actor firing.

The transformation is straightforward: each actor in the MRSDF graph is duplicated as many times as it is executed in a single iteration. These copies represent the individual firings of the actor. For each token produced by a single firing, a single channel is created. This channel connects the individual firings that produce and



(a) MRSDF graph (b) HSDF graph

Figure 2.11 – Transformation of an MRSDF graph into an equivalent HSDF *multigraph*, motivated from a *functional* perspective.

consume the token. As a result, the equivalent HSDF graph is a *multigraph* (as opposed to a *simple* graph): a pair of vertices may be connected by multiple parallel edges. The result of this transformation, applied to an example MRSDF graph, is depicted in Figure 2.11.

The transformation described above is motivated from a *functional perspective*: each actor firing signifies the processing of a number of input tokens (as specified by the consumption rate), resulting in a number of output tokens. This behaviour is maintained by the transformation into the single-rate equivalent: the total number of tokens consumed and produced by an HSDF actor equals the number of tokens consumed and produced by the corresponding MRSDF actor. However, when the graph obtained from the transformation is subsequently analysed for its *temporal properties*, the functional perspective seems a poor choice. This is reflected by the fact that many channels in the (functionally) equivalent HSDF graph are redundant, in the sense that they express an equally strong or weaker *constraint* on the firing times of other actors. In particular, if two vertices are connected by *parallel* arcs, then they may be replaced by a single arc that enforces the strongest precedence constraints enforced by the set of parallel arcs [51, 52, 88]. As a result, the multigraph obtained by the transformation outlined above is transformed into a simple graph, which simplifies the analysis. As an example, the result of applying this reduction step to the multigraph is shown in Figure 2.12(a). The reduced graph contains 16 arcs, which is a reduction of one third compared to the multigraph. Consequently, the number of tokens in the HSDF graph now no longer matches the number of tokens found in the original MRSDF graph.

The multi-rate HSDF graph contains many channels that are redundant, in the sense that the precedence constraints they impose are equal or weaker than those imposed by other channels. In [51, 52], techniques are presented to decrease the size of the graph by pruning these redundant channels, and by contracting certain actors into a single actor. As a result of these *node* and *edge degeneration* techniques, the HSDF graph is transformed into a smaller graph, which speeds up its analysis.

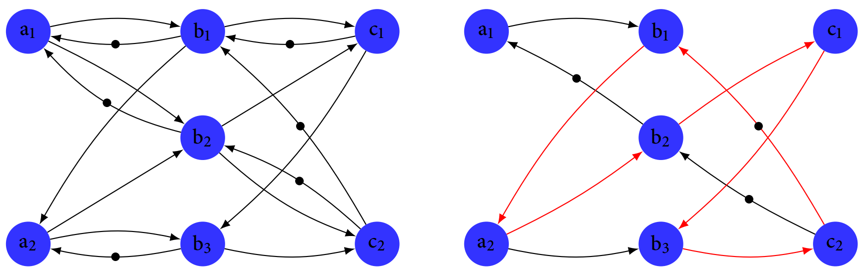
In the context of Petri nets, related transformations exist. A transformation analogous to the one presented in this thesis in Chapter 4 is given in [71], where a *weighted marked graph* (i.e., an MRSDF graph, see Section 2.1.7) is transformed into an unweighted marked graph (an HSDF graph). The transformation differs from the one given in [60] in that it is motivated from a *temporal perspective*. A notable aspect of the transformation is that it explicitly represents the ordering of firings of a single transition. As a result, the resulting unweighted marked graph is strongly connected. This differs from the transformation that we present in Chapter 4, where the single-rate equivalent of a strongly connected SDF graph may not be strongly connected. In Chapter 6, we show how strong connectedness of the single-rate equivalent is not necessary for throughput analysis.

In Chapter 4 of this thesis we present a transformation of an MRSDF graph into a *temporally* equivalent HSDF graph, that differs from the one presented in [60, 88], This transformation can be regarded as a combination of the transformation and reduction step outlined above: arcs in the equivalent HSDF graphs are treated as precedence constraints, and these constraints arise from the data dependencies enforced by channels in the MRSDF graph. A key distinction, however, is that due to choosing a max-plus characterisation of the MRSDF graph as a starting point, the transformation outlined in this thesis treats arcs in the MRSDF graph as precedence constraints. As a result, a further reduction of arcs in the equivalent HSDF graph is achieved, and no follow-up reduction step is necessary. As an example and a preview into Chapter 4, the graph obtained using our transformation is depicted in Figure 2.12(b). There are only 10 channels in the graph; 6 less than in the reduced graph of Figure 2.12(a) and 14 less than in the multigraph of Figure 2.11(b).

When analysing the throughput of an SDF graph by computing the maximum cycle ratio of its single-rate equivalent, actors in the latter that do not lie on a cycle, may be pruned from the graph as well. In Chapter 6, we show how the maximum cycle ratio of a single-rate equivalent may be computed from a single strongly connected subgraph. For some graphs, this results in an HSDF graph that is more than 15,000 times smaller than the full single-rate equivalent.

The single-rate equivalents constructed in [51, 52, 60, 88] all represent individual firings of actors by actors in the HSDF graph. The size of the single-rate equivalent is thus equal to the number of firings in a single graph iteration.

A different approach is to consider the temporal behaviour of an SDF graph from the viewpoint of the graph's *tokens*, as is presented in [35–37]. Rather than capturing the precedence constraints between actor *firings*, one may capture the interdependencies between the *tokens* that are in the graph. The time at which a token is produced onto a channel, by some firing, depends on the times at which the tokens consumed by that firing have become available. By symbolically executing a single iteration of the graph, the details of the dependencies between tokens are captured



(a) HSDF simple graph

(b) *Temporally* equivalent HSDF graph obtained in Chapter 4 of this thesis

Figure 2.12 – HSDF graph obtained from Figure 2.11(a), using the transformation presented in Chapter 4 of this thesis.

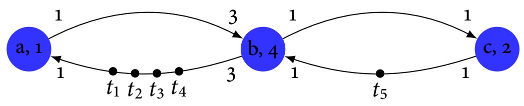


Figure 2.13 – By recording the interdependencies between individual tokens, as they flow through the graph during a single iteration, a first-order max-plus system is obtained.

as a first-order, linear shift-invariant max-plus system. The system has a single equation for each token in the graph. Thus, for graphs in which the number of tokens is smaller than the size of the graph's single-rate equivalent, this method provides an attractive and powerful alternative to the analysis of SDF graphs.

As an example, we give the max-plus system that is obtained using the token-based transformation from the MRSDF graph depicted in Figure 2.13:

$$t(k+1) = \begin{pmatrix} \varepsilon & \varepsilon & \varepsilon & \varepsilon & 0 \\ 5 & 5 & 5 & \varepsilon & 4 \\ 5 & 5 & 5 & \varepsilon & 4 \\ 5 & 5 & 5 & \varepsilon & 4 \\ 7 & 7 & 7 & \varepsilon & 6 \end{pmatrix} \otimes t(k),$$

where the first five elements of vector t correspond to the five tokens marking arc ba, in the order in which they are consumed by actor a. The last four elements in t correspond to the four tokens on arc cb.

Using a straightforward transformation, the token-dependency max-plus system can be represented by an HSDF graph. This graph is slightly larger than the max-plus system, but may be smaller than a single-rate equivalent obtained using the "traditional" transformation. In [35], it is shown that a reduction up to a factor of 279 is achieved. This particular figure is obtained for the graph shown in Figure 2.14, the single-rate equivalent of which has 10601 actors. Since the graph has only six tokens,

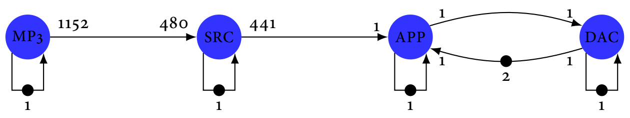


Figure 2.14 – The SDF graph representing an MP3 playback application, used in [35] as an example to demonstrate the effectiveness of capturing the dependencies between tokens, as an alternative to constructing a single-rate equivalent. As the graph contains only six tokens, the corresponding max-plus system consists of 6 equations.

the max-plus system obtained for this graph by the method of [35] is given by six max-plus equations. If we convert the max-plus system to an HSDF graph, the result has 38 actors, which is 279 times smaller than the single-rate equivalent constructed using "traditional" methods. However, this reduction factor does not provide a fair comparison with these traditional methods: since the graph of Figure 2.14 is not strongly connected, the traditional method would be applied to each of its strongly connected components, rather than to the entire graph. Applying this principle to the graph of Figure 2.14 means that three single-rate equivalents (i.e., one for each component) must be analysed: actors MP3 and SRC each form a connected component, and the subgraph induced by actors APP and DAC forms the third component. The total number of actors in the single-rate equivalent of these three components is *four*, rather than the number of 10601 given in [35]. This means that, rather than obtaining a reduction of a factor of 279, the technique presented in [35] results in a graph that is almost ten times larger. In other words, the comparison reported in [35] is unfair, and biased towards a positive outcome for the token-based transformation.

A comparison of the transformation presented in [35], and the efficiency of methods that are based on it, such as those described in [36, 87], with the "traditional approach" of construction and subsequent analysis of a single-rate equivalent, must be fair. That is, the latter must consider each of the SDF graph's strongly connected components indvidually, rather than transforming the full graph to its single-rate equivalent. We therefore repeat the comparison made in [35], but construct a single-rate equivalent for each the strongly connected components. The total size of the HSDF graph (i.e., the sum of the sizes of the different HSDF graphs) is reported in Table 2.1. The last column in the table lists the ratio between the sizes of the graphs obtained by the transformation of [35], and the transformation of each of the graph's strongly connected components. A ratio smaller than one indicates that the HSDF graph obtained by the method of [35] is in fact larger than the traditional transformation. Note that the ratios reported in [35] (listed in the second to last column) differ significantly from the actual ratios.

The transformations described above all transform an MRSDF graph into an HSDF graph that is equivalent in some sense. Other kinds of transformations are described in literature as well. The importance of dataflow graph transformations is the motivation of [82]. The paper proposes methods of modeling and transfor-

Case study	HSDF graph size reported in [35]	Total size HSDF graph components	Size using [35]	Ratio reported in [35]	Actual ratio
h.263 decoder	1190	4	10	119	0.4
h.263 encoder	201	201	11	18.3	18.3
modem	48	17	210	0.23	0.08
mp3 dec. block par.	911	3	8	114	0.38
mp3 dec. granule par.	27	3	8	3.38	0.38
mp3 playback	10601	4	38	279	0.11
sample rate conv.	612	6	31	19.7	0.19
satellite	4515	22	217	20.8	0.10

Table 2.1 – A comparison of the 'traditional' transformation of MRSDF into its single-rate equivalent, both naively and per strongly connected component, and the transformation presented in [35].

mation for more flexible forms of dataflow, such as parameterized synchronous dataflow and cyclo-static dataflow. The transformations presented in [82] is *clustering*. A clustering transformation for a dataflow graph is one that replaces a set of multiple actors in the graph with a single actor. As a result, the chosen set of actors is viewed as a single "unit". [77]. A similar kind of clustering is proposed in [80]. Here, rather than clustering actors, firings of actors are clustered. This is done by adapting their production and consumption rates. A valid clustering step is one that replaces each of the rates associated with an actor by a certain integer multiple. This gives rise to a class of SDF graph that is named *scalable synchronous dataflow* in [45, 80]. [3] presents a clustering transformation that groups certain actors together that are to be run on the same processor. In [45], it is shown how clustering of nodes may be achieved within the CSDF paradigm.

Whereas clustering reduces the size of an SDF graph, *unfolding* is a transformation that increases the size of the graph, by splitting every actor into multiple actors. The first such transformation, applied to HSDF graphs, is presented in [74]. Here, a procedure for constructing an $N-$unfolded HSDF graph is described: every actor in the graph is split into N actors, unraveling the "hidden concurrency" of iterative dataflow programs. The procedure is used to construct equivalent graphs in which each cycle consists of only a single initial token. Surprisingly little work is reported on the unfolding of MRSDF or CSDF graphs. A first attempt to the unfolding of MRSDF graphs is presented in [35]. The procedure to unfold an MRSDF graph N times (listed as Definition 5 in [35]) seems a generalisation (although no reference is made) of the procedure first described in [74]. Although the unfolding procedure is claimed to yield a graph that has the same throughput (Proposition 2 in [35]), no proof of the correctness is given. In fact, as also pointed out in [72], the unfolding presented in [35] is incorrect. According to [72], unfolding an MRSDF graph is possible only for a very specific subset of graphs. The graphs that may be unfolded are those for which each channel has an initial number of tokens that is an integer multiple of the number of tokens consumed, per iteration, of the channel's consuming actor. Furthermore, it is noted that the procedure does not necessarily apply to CSDF graphs.

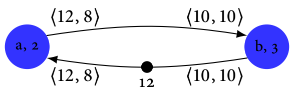
The transformations that we present in Chapter 4 may be regarded as an unfolding of CSDF graphs. They differ from the approaches taken in [35, 72, 74], in the sense that they follow from a characterisation of the temporal dynamics of a CSDF graph. Furthermore, the transformations presented in this thesis allow each actor to be unfolded separately, rather than unfolding the entire graph.

Clustering is a transformation that we use in this thesis to reduce the size of the HSDF graph that is analysed for its critical cycle. We contract actors in the single-rate equivalent into a single actor. This potentially changes the graph's throughput. In Chapter 6, we describe a method that applies these contractions only if the throughput is left unchanged. Otherwise, the graph is unfolded. Our approach is similar to the scaling step of [80], in that the clustering in the single-rate equivalent is indirectly, by changing the rates of an actor. An important difference is that the rates are not simply scaled, but rather transformed into a cyclically varying pattern of rates. The clustering and unfolding together give rise to an approach to throughput analysis where the graph's size is kept as small as possible.

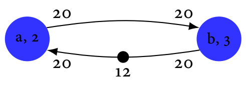
Transformation of CSDF graphs

The transformation of an MRSDF graph into an equivalent HSDF graph can be generalised towards CSDF graphs in a straightforward way, the main difference being that determining the correspondence between producing and consuming firings is slightly more complex. The construction of a single-rate equivalent of a CSDF graph is described in the introductory paper [11]. This construction differs from the token-oriented transformation of [62], which is applied to CSDF graphs in [69] and [88]. It involves computing, for each firing that occurs in a single graph iteration, the producing firings of the last tokens consumed by that firing. As a result, the in-degree of an actor in the CSDF graph equals the in-degree of each of its copies in the single-rate equivalent.

The transformation outlined in [11] is largely similar to the one we present in Chapter 4. Our concept of a predecessor function gives precisely the index of the firing that produces the last token consumed by some other firing. The predecessor function thus gives the topology of the single-rate equivalent. An important difference between the transformation of [11] and the transformation put forward in this thesis, is that [11] restricted its scope to CSDF graphs in which each actor has a self-loop. Such a self-loop enforces a sequential execution of successive phases. The transformation presented in Chapter 4 widens the scope to CSDF graphs where auto-concurrent execution of consecutive actor phases is allowed. Functional determinacy is maintained by including an extra constraint on the times at which actor firings complete. This replaces the restrictive use of self-loops in [11] by an



(a) CSDF graph.



(b) Lumped representation.

Figure 2.15 – An example CSDF graph and its *lumped* representation. The transformation is *conservative*, in the sense that the throughput of the lumped representation is never higher than the throughput of the CSDF graph. The CSDF graph has a throughput of $\frac{1}{10}$ and, since neither actor is enabled, its lumped representation is deadlocked.

equally effective measure, while ensuring that the class of CSDF graphs contains the class of MRSDF graphs.

A straightforward transformation from CSDF into MRSDF exists [45, 75, 88]. This transformation involves replacing all (execution time and rate) vectors by their sum, which simplifies a subsequent analysis. The resulting graph is often referred to as the graph's "lumped" representation. Again, this transformation takes a functional stance: multiple phases, which correspond to the sequential execution of phase-specific functions, are grouped into one monolithic function. Although the lumped representation may be regarded as functionally equivalent, its temporal dynamics are certainly not maintained. In fact, the temporal behaviour of the graph gives a conservative approximation of the CSDF graph. In the best case, the throughput of the lumped representation is the same as the throughput of the original graph. Generally, however, the throughput of the former is lower, and the transformation may introduce deadlock. This is illustrated in Figure 2.15, which depicts a live CSDF graph, and a deadlocked lumped representation.

We present a transformation of a CSDF graph into an *equivalent* MRSDF graph in Chapter 4. Equivalence here is to be understood in the temporal sense; any schedule that is admissible for the CSDF graph, can be transformed into one that is admissible for the multi-rate equivalent, and vice versa. This transformation makes existing analysis techniques for MRSDF graphs applicable to CSDF graphs. The number of actors in the equivalent MRSDF graph is equal to the total number of actor phases in the CSDF graph. Our transformation from an CSDF graph into its multi-rate equivalent is a special case of a transformation that allows an arbitrary CSDF actor to be an unfolded an arbitrary number of times. In case each CSDF actor is unfolded an integer multiple of its period, the resulting graph is an MRSDF graph.

2.3.2 Approximations

The potential complexity of schedules for MRSDF and CSDF graphs, in terms of the irregularity of the firing time patterns, is limited by the length of a single graph iteration. Because the length of a graph iteration grows exponentially in the size of the graph, an exact analysis of CSDF graphs is, generally, computationally intensive. The aim of computing a *conservative* approximation of, for example, the throughput of an SDF graph, is to compute a *guaranteed* performance metric. This is achieved by assuming simple firing patterns, such as strictly periodic schedules. These firing patterns must be *conservative*: the performance predicted from the approximation of the model must not be better than the actual performance.

Although the vast majority of literature on approximating the throughput of an SDF graphs aims at computing a *lower bound* on throughput, the problem of computing an *upper bound* is tackled in [84]. The method presented is similar to the concept of an *optimistic* single-rate approximation, presented in this thesis in Chapter 5: the upper bound on the throughput of a cycle is taken as the ratio between the weighted sum of tokens (conform the normalisation vector, see Section 2.1.3) and the sum of the execution times.

A first such approximation is presented in [23], in the context of Petri nets (named "timed event graphs with multipliers" and equivalent to MRSDF graphs). The authors use a min-plus algebraic characterisation of the number of times an MRSDF actor has fired, in terms of the firing counts of connected MRSDF actors. By taking linear bounds on what they refer to as *transition-to-transition* equations, followed by a transformation step called *linearization*, a so-called *fluid* timed event graph (i.e., an HSDF graph) is obtained. The performance (throughput) of the fluid approximation is never better than the performance of the approximated graph; deadlock may be introduced by the transformation.

The transformation of an MRSDF graph into a *single-rate approximation* outlined in Chapter 5 strongly resembles this approach. Our transformations generalise towards CSDF graphs, and, moreover, give both an optimistic (in which every admissible schedule of the CSDF graph is admissible for the optimistic approximation) and a pessimistic one (every admissible schedule of the pessimistic approximation is admissible for the CSDF graph).

Conservative approximations allow one to derive sufficient conditions for the liveness of a CSDF graph. In [8], two different sufficient conditions for the liveness of a CSDF graph are given. These conditions are derived from two different intermediate weighted directed graph representations: if the CSDF graph is deadlocked, then the weighted digraphs contain a negative cycle. Absence of a negative cycle is thus a sufficient condition for liveness of the CSDF graph. The work presented in [8] is a generalisation of a similar approach, applied to MRSDF graphs, presented in earlier in [65]. The relationship between the approaches presented in [8, 65] with the single-rate approximations presented in Chapter 5 is strong. The single-rate approximations that we present in this thesis allow one to give sufficient as well as necessary conditions for liveness of CSDF graphs. Rather than constructing a graph that considers each phase, our approach splits this up in two steps: a transformation, followed by an approximation. The sufficient condition for liveness named SC1 in [8] is analogous to the single-rate approximation of the multi-rate equivalent of the CSDF graph.

In a CSDF graph, the number of tokens that "flow" through a channel per iteration, differs per channel. This is a direct result of the fact that an actor may have different production and consumption rates associated with different channels. In the *compositional temporal analysis* (CTA) model, presented in [48], this property is explicitly captured by *transfer rates*: the CTA model uses components with transfer rates per port, in contrast with dataflow models, which consist of actors with firing rules. The use of transfer rates makes the model attractive to the modeling of multi-rate systems. As the name of the model indicates, it is *composable*. The composition of components is again a component, and the properties of the new component can be deduced from the properties of and the connections between the individual components. The authors of [48] illustrate how a CTA graph forms an abstraction of an SDF graph. In fact, the CTA abstraction of an SDF graph is similar to the transformation into a pessimistic approximation that we describe in Chapter 5. There are two important differences between the former and the latter. First of all, the CTA model assumes *periodic token arrival curves*, whereas the pessimistic single-rate approximations of Chapter 5 are based on actor firings. Second, the rate at which tokens arrive at a port in the CTA model may differ per port, whereas, since they are HSDF graphs, in the approximations of Chapter 5 all these rates are equal. Through a change of counting units (see Section 5.2.1), any CTA model may be transformed into an HSDF graph. Rather than resorting to the rather heavy tool of using an linear program solver to analyse the CTA model for consistency or throughput, one may thus apply simple combinatorial algorithms. This means that the model has the same expressive power as timed event graphs, and that standard algorithms can be used to analyse the model.

In [15], the sufficient condition for liveness of CSDF graphs is used as a fast test to generate live random CSDF graphs. The procedure is faster than the method used in [90].

2.3.3 Throughput analysis

The analysis of throughput, which is the average number of iterations completed per time unit, of an SDF graph, is a well-studied subject in case the SDF graph is homogeneous. As early as 1968, Reiter described how the throughput (or cycle time) of a restricted version of the computation graph of Karp and Miller could be determined [78]. The graphs dealt with by Reiter were, in fact, HSDF graphs. Reiter showed that the maximum rate at which each node in the graph could execute, is determined by the graph's maximum cycle ratio.

For SDF graphs where actors fire at different rates (e.g., MRSDF and CSDF), however, computing the throughput is more difficult. Two main classes of approaches exist: those that are based on the construction of a single-rate equivalent (see previous sections), and those that simulate a self-timed execution of the graph. In the influential paper [42], it is described how the throughput of an MRSDF graph may be computed by exploring the state-space of a self-timed execution of the graph for its periodic regime. This approach to throughput analysis avoids the costly transformation of MRSDF into HSDF.

A self-timed execution yields a schedule that may not be (strictly) periodic from the start, except when the initial starting times happen to coincide with the system's eigenvector (see Section 2.2.4). Assuming that the SDF graph is consistent, such a schedule will eventually settle in a periodic phase [42, 49]. A single period in this phase consists of a certain number of graph iterations. Since the length of a period (in terms of the number of graph iterations that compose it) is not known a priori, a history of actor firing times must be maintained. The approach presented in [42] cleverly keeps this history small. It exploits the fact that only those states in which some actor has just completed a full iteration need to be saved for reference. The length of the history is thus equal to the number of graph iterations that must be completed before a period is found.

Computing the throughput of a graph by simulating a self-timed execution, until a periodic phase is found, is analogous to applying the *power algorithm* to compute the eigenvalue of the corresponding max-plus system [49]. The power algorithm, applied to a matrix $A \in \mathbb{R}_{\max}^{n\times n}$, starts with an initial vector, $x(0)$, and iteratively computes a next vector by left-multiplying it (in the max-plus sense) with A:

$$x(k+1) = A \otimes x(k).$$

The sequence of vectors $x(0), x(1), \dots$ will eventually become periodic. That is, eventually, for some integers k and m, and $c \in \mathbb{R}_{\max}$, the following will hold true:

$$x(k+m) = c \otimes x(k).$$

The eigenvalue of the system is then computed as $\lambda = \frac{c}{m}$, and eigenvector is given by $\bigoplus_{j=1}^{m} \lambda^{\otimes \sigma - j} \otimes x(k+j-1)$ [49].

Unfortunately, the value of k for which the above holds may be quite large. This means that the execution of a potentially large number of iterations may need to be simulated (i.e., the transient phase), before a periodic phase is found. Only very pessimistic bounds on the length of this transient phase are known [49].

To highlight both the strengths and weaknesses of this method, we discuss two examples. To illustrate its efficiency by avoiding the construction of a single-rate equivalent, consider the graph depicted in Figure 2.16. The rates associated with actor b and the number of initial tokens in this graph have been left variable; the size of the single-rate equivalent is equal to $5 + 2n$ (assuming that n is not a multiple of 5). For this graph, the transient phase of a self-timed schedule consists of the initial n parallel firings of actor a, followed by a periodic regime in which 5 parallel firings of b are followed by n parallel firings of both a and c. Consequently, the

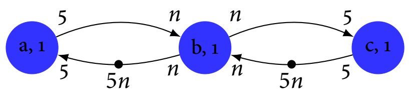


Figure 2.16 – MRSDF graph for which the self-timed execution almost immediately enters a periodic regime. As a result, a state-space exploration-based approach to compute the graph's throughput is extremely efficient.

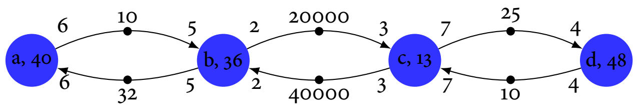


Figure 2.17 – MRSDF graph for which the self-timed execution has a long transient time.

throughput for this graph is computed in as few as 3 simulation steps, independent of the size of the equivalent HSDF graph.

The method performs poorly on graphs where a large number of firings must be simulated before the schedule settles in a periodic phase. An example of such a graph is given in Figure 2.17. For this graph, the SDF analysis tool SDF3 requires a few hours (17 hours on a 2.4GHz Intel Xeon CPU) to compute its throughput. Note that the single-rate equivalent of the graph is relatively small: it consists of 22 actors.

The work of [42] includes an experimental study, in which state-space exploration based throughput analysis is compared with several algorithms that analyse the single-rate equivalent. The study makes an unnecessary distinction between algorithms that are based on computing the *maximum cycle mean*, and those that compute the *maximum cycle ratio*: the maximum cycle mean computation is applied to a transformation of the graph's single-rate equivalent into a weighted digraph, which may be regarded as its first-order max-plus system representation. Such a transformation adds an unnecessary amount of computation time to the analysis: for one of the four benchmark sets, consisting of 100 graphs, the analysis of 44 was aborted after failing to complete within half an hour.

Algorithms that compute the maximum cycle ratio are applied directly to the single-rate equivalent. Results for this approach are not reported in the study. Instead, the authors reason that an analysis based on a maximum cycle ratio computation requires at least the time needed to transform the SDF graph into its single-rate equivalent. As the implementation of this transformation (as used in the study) took, on average, significantly more time than the analysis of the HSDF graph it produced, it formed a clear bottleneck in the analysis, leading to a conclusion that favours space-space based throughput analysis over the "classical approach".

The experimental comparison reported on in [42] involves several choices that introduce a bias in the experiment. This bias favours the state-space exploration based approach. First of all, the multigraph created by the MRSDF-to-HSDF transformation is not reduced to a simple graph prior to running the analysis. This has a dramatic effect on the algorithms that compute the graph's maximum cycle ratio, as their runtime depends on the number of arcs in the graph. Furthermore, the transformation of the MRSDF graph into an equivalent HSDF multigraph is unnecessary, as the latter is transformed into a marked weighted directed multigraph. These two steps can be merged into a single step in a straightforward way. Rethinking these choices should lead to increased performance for the HSDF-based approach, albeit that for SDF graphs for which the norm of the repetition vector is large, storing the equivalent HSDF graph in memory will become very costly.

In Chapter 6, we present an algorithm for throughput analysis of MRSDF graphs that avoids transforming the graph into an equivalent HSDF graph. It exploits the fact that individual cycles in the MRSDF graph may be analysed quite efficiently, without explicitly creating its single-rate equivalent. The computation of throughput then becomes a problem of composing the throughput of individual cycles, which is non-trivial for multi-rate graphs. We present an extensive comparison between the state-space exploration approach and our approach in Chapter 6.

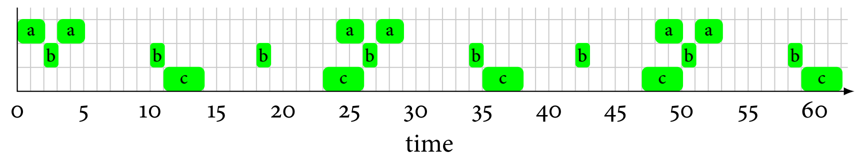
The transformation based on the symbolic execution of an SDF graph, described in [35, 36], has as a benefit that it is not susceptible to the long transient times that may affect a state-space exploration based approach. Although its efficiency has not been experimentally acknowledged, throughput analysis using said transformation is expected to be more efficient than simulation.

Approximate analysis

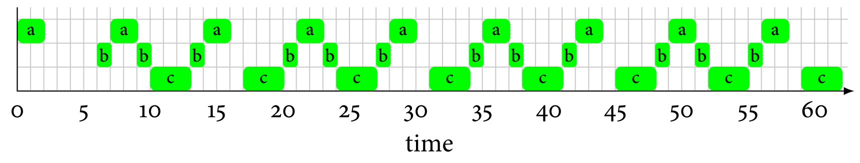
Motivated by the high complexity of an exact throughput computation of an SDF graph, methods have been proposed to approximate a graph's throughput *conservatively*. In [14] it is argued that maximum throughput is achieved by a so-called N-periodic schedule, where N is the repetition vector of the graph, the entries of which may be very large. Constructing an equivalent HSDF graph and computing its throughput is essentially the same as computing a N-period schedule. By choosing N such that contains small entries, computation time can be reduced, at the cost of acquiring a lower throughput.

As an example, Figure 2.18(a) gives a conservative (2,1,1)-periodic schedule that is admissible for the MRSDF graph of Figure 2.11(a). The throughput given by the schedule is $\frac{1}{24}$, whereas the throughput given by the $(2,3,2)$-periodic schedule (i.e., the schedule obtained from the equivalent HSDF graph) is $\frac{1}{11}$. If we construct a schedule in which we schedule actor b 3-periodically, we obtain a higher throughput, as depicted in Figure 2.18(b). As is pointed out in [14], increasing the irregularity of the schedule of an arbitrary actor does not necessarily increase the throughput of the schedule. Here, K is a vector that assigns each actor an individual period. By controlling the vector K, one can control the complexity of the computation.

There is a strong relationship between the k-periodic schedules, presented in [14],



(a) A (2,1,1)-periodic schedule, with a throughput of $\frac{1}{24}$.



(b) A (1,3,1)-periodic schedule, with a throughput of $\frac{1}{14}$.

Figure 2.18 – Two different admissible schedules for the MRSDF graph from Figure 2.11(a), with different throughputs.

on the one hand, and the transformations and approximations that we present in chapters 4 and 5, on the other hand. In fact, the computation of a K-periodic schedule essentially consists of a transformation of the CSDF graph, followed by an approximation step. The transformation involves the *unfolding* of actors, with the vector N giving the number of times each actor is unfolded. By computing a pessimistic single-rate approximation for the unfolded graph, a K-periodic schedule is obtained. This idea gives rise to an approach where the accuracy is balanced with complexity: both the approximated throughput, as given by the N-periodic schedule, and the computational effort to obtain the schedule, increase monotonically in the entries in N. However, as is pointed out in [14], for many choices of N, the throughput does not increase. The relationship between the entries in N and the resulting throughput is far from trivial.

This thesis advances the work presented in [14]. In Chapter 6, we show that enforcing the *parallel* execution of firings of actors lets the accuracy of the approximated throughput increase steadily. The approach to throughput analysis that we present in Chapter 6 tackles the complex relation between computational effort and complexity of the schedule.

Exact and approximate techniques can be *combined* to compute the throughput of a CSDF graph. In [2], an approximation technique, loosely based on the state-space exploration approach of [42], is presented. The motivation for the presented method is the exponential complexity of exact throughput analysis on the one hand, and the potential poor accuracy of approximate methods such as [100]. The approach presented in [2] involves the simulation of a self-timed execution for a fixed number of graph iterations. The authors claim that from such a partially simulated execution, one may obtain a lower bound on the graph's throughput. Although the obtained approximation does converge to the graph's throughput as the number of graph iterations considered increases, no quantitative assessment of the approximation error is given. Furthermore, as the paper lacks a formal proof, it is not clear from the paper whether the computed throughput gives a lower or upper bound on the actual throughput.

2.4 Discussion

Literature on SDF graphs has settled into two main schools: the first is concerned with *exact* analysis, and the second sacrifices accuracy for computational complexity, in methods that provide *conservative approximations*. The common ground of these two schools is that an analysis, by transforming MRSDF and CSDF graphs into an equivalent HSDF graph, is considered too costly for practical use. We believe that one of the reasons that this conviction has become predominant in literature is because the transformation that was initially proposed by Lee was motivated from a *functional* perspective, rather than from a *temporal* one. The popularity and the apparently convincing results obtained in comparably small experiments have distracted researchers from pursuing a sound mathematical basis for the characterisation of the behaviour of SDF graphs.

In this thesis, we propose to revisit a mathematical characterisation of the temporal behaviour of SDF graphs. Such a mathematical basis should serve as a common ground for both exact and approximate analysis. As a result, this thesis can be regarded as one that closes the gap that now exists between exact and approximate methods, by showing how they both derive from a solid, common, ground.

If we observe the state-of-the art approaches to analysing MRSDF and CSDF graphs, then we may conclude that the relation between SDF graphs and discrete event systems is not yet well-established. One particular example that serves to justify this conclusion is the use of the term *maximum cycle mean* in dataflow literature, where it refers to a quantity that we defined in this chapter as *maximum cycle ratio*. Although using the same term under different definitions in different fields is harmless, this ambiguity becomes troublesome when the term has different meanings in fields that are (becoming) closely linked. This is best illustrated by the experimental study conducted in [42], which performed an unnecessary and time consuming transformation prior to applying a maximum cycle-mean analysis. Regretfully, the apparent dominant performance of the presented model-checking approach has distracted researchers from pursuing a mathematical characterisation of the dynamics of SDF graphs.

The terms and definitions given in this chapter are partially derived from the field of synchronous dataflow, and partially from the field of discrete event systems. This mixture may be confusing especially to readers that are primarily familiar with the former: well-established mathematical concepts such as eigenvalues and eigenvectors are typically not used in SDF parlance. Although SDF terms such as throughput, periodic schedules, self-timed execution, etc., are all intuitive concepts with a well-understood interpretation, a mathematical characterisation of these concepts provides a perspective that unifies concepts known from the two said fields.

The main principle underlying this thesis is that analysis of SDF graphs should be based on a solid, mathematical foundation. This is a principle that is successfully applied in many fields that study the dynamics of a system. In areas such as mechanical and electrical engineering, telecommunications and physics, linear time-invariant systems play an important role. Although the systems studied in these areas are typically non-linear and / or time-varying, linearisation techniques often allow them to be analysed, at the expense of reduced accuracy. The characterisation of SDF graphs as discrete event systems allows one to apply similar techniques to cope with the complexity of analysing large systems.

3 A mathematical characterisation of Synchronous Dataflow Graphs

Abstract – *There are different ways to define the semantics of a synchronous dataflow graph. One may define the graph as a function that evolves a state over time, specify how the number of tokens on a given channel depends on the number of times actors have fired, etc. In this chapter, we present a mathematical characterisation of synchronous dataflow graphs that allows one to make formal statements by reasoning over their admissible schedules. The characterisation serves as a basis for the transformations and approximations that we discuss in subsequent chapters.*

If we take an operational perspective on an SDF graph, and consider it as a process in which actors fire and tokens flow through channels, then the "behaviour" of an SDF graph is largely determined by only two simple rules. These rules state that (1) an actor may fire as soon as the required data is available on each of its incoming channels, and (2) that such a firing takes an amount of time. Here, the term *required data* relates to the *number of tokens*, which must be at least as high as the consumption rate associated with the firing.

The two rules given above define the order in which firings may occur. In Section 3.1, we specify this ordering formally. We introduce so-called *predecessor* functions, which specify the dependencies that govern the consumption and production of tokens, in terms of the properties of the channels in an SDF graph. Predecessor functions serve as the basis for the approximations and transformations described in the following two chapters.

Large parts of this chapter have been published in [RdG:3].

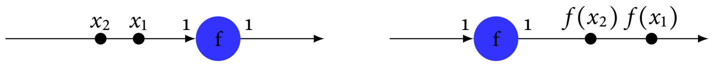


Figure 3.1 – This thesis views SDF graphs as monotonic dataflow process networks. Any sequence of actor firings is *functional*: the sequence of data produced by them is a function of the sequence of data they consume. This means that the ordering of the stream of input tokens is maintained in the stream of corresponding output tokens, regardless of the time of the computation.

The ordering of firings gives rise to a *system-theoretic* characterisation of SDF graphs. In this view, an SDF graph is regarded as a system that transforms input sequences of timestamps into output sequences of timestamps. In Section 3.2, we show that for consistent SDF graphs, this system is *linear* and *periodically shift-varying*, when expressed in max-plus algebra. This perspective allows us to relate different SDF graphs to each other and identify common properties, through their input-output behaviour. In particular, this gives rise to the notions of *equivalence* and *abstraction*, which we introduce in Section 3.3. In subsequent chapters we use these notions when proving certain relations between graphs and their transformations.

3.1 Dataflow processes and firing order

In an SDF graph, firings of actors follow the *dataflow principle*[32, 60]: An actor may fire if sufficient data is available on its incoming channels, as specified by the consumption rates. This data is made available by firings of upstream actors. A sequence of actor firings is referred to as a *dataflow process* [32, 61]. In this thesis we restrict our attention to SDF graphs in which the dataflow process associated with each actor is *functionally determinate*. That is, a sequence of actor firings always maps the same sequence of input tokens to the same sequence of output tokens, regardless of the duration of a firing (see Figure 3.1). These SDF graphs are called *monotonic dataflow process networks* [61].

A sufficient condition for a dataflow process to be functionally determinate is that each actor firing is *functional*, and that firings are strictly ordered [61]. This means that the data produced by a single firing is a function of the data it consumes, and that the mapping between the input and output sequences is *prefix-monotonic*: given a prefix of the input sequence, part of the output sequence may already be computed. Furthermore, tokens are produced onto and consumed from channels in a first-in-first-out fashion.

Formulated differently, we assume that firing k of actor v consumes and produces its input and output tokens before its next firing, $k + 1$, does. The effect of this assumption is that the index of an actor's firings expresses the order in which these firings occur. Second, this assumption guarantees that the dataflow processes we consider are prefix-monotonic. We refer to a specific firing of an actor, by the index in the dataflow process associated with the actor, written in subscript: we denote firing k of actor v by v_k. The above gives rise to a *strict partial ordering* on actor firings, defined as follows:

Definition 3.1 (Functional ordering of firings). *Let $\mathcal{G}$ be an SDF graph. The* functional ordering of firings *of actors in $\mathcal{G}$ is a relation, denoted $\prec$, between the firings of actors in $\mathcal{G}$, and defined as follows. The functional ordering of firings of the* same *actor, v, follows from their index in the dataflow process associated with v:*

$$k < m \Rightarrow v_k \prec v_m.$$

The functional ordering of firings of different *actors, v and w, is defined for actors that are connected by a channel vw:*

$$v_k \prec w_m,$$

if the m^{th} firing of w can not consume its tokens before the k^{th} firing of v has completed.

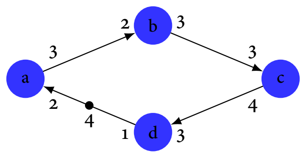
This definition is best illustrated with an example, see Figure 3.2. In the MRSDF graph of Figure 3.2(a), the first firing of actor b requires that (at least) two tokens have been produced onto channel ab. This condition is satisfied as soon as actor a has completed its first firing. We thus have $a_1 \prec b_1$. Examples of other relations that hold are $a_1 \prec b_2$ and $b_1 \prec c_1$. Figure 3.2(b) gives an overview of all the relations, in the form of a directed acyclic graph: it contains an edge vw if and only if $v \prec w$.

Clearly, the relation $\prec$ is *transitive*. For example, in Figure 3.2 we have $a_2 \prec c_2$ because $a_2 \prec b_2$ and $b_2 \prec c_2$. Firings for which $\prec$ is undefined are said to be unrelated. Unrelated firings may occur in any order, such as for example a_2 and b_1 in Figure 3.2. A second property of $\prec$ is that it is *irreflexive*. That is, for *no* firing v_i we have $v_i \prec v_i$. Furthermore, $\prec$ is *anti-symmetric*: if $v_i \prec w_j$, then not $w_j \prec v_i$. Hence, the functional ordering defined by Definition 3.1 is indeed a *strict partial ordering*.

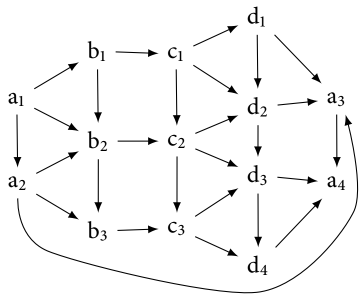
The functional ordering between the producing and consuming actors of a channel follows from the channel's production and consumption rates, as well as its initial tokens. We make the relation between $\prec$ and these properties specific, starting with the introduction of the following *token balance* function $\Delta_{vw}(i,j)$, which gives the number of tokens on a channel vw, in terms of the number of completed producing firings, i, and consuming firings, j. For the sake of completeness, we let the definition allow for a *negative* numbers of firings.

$$\Delta_{vw}(i,j) = \delta_{vw} - \sum_{l=i+1}^{0} \rho^{+}_{vw}(l) + \sum_{l=1}^{i} \rho^{+}_{vw}(l) + \sum_{l=j+1}^{0} \rho^{-}_{vw}(l) - \sum_{l=1}^{j} \rho^{-}_{vw}(l). \qquad (3.1)$$

A consuming actor may start its next firing if all of its incoming channels contain sufficiently many tokens. Formulated differently, each producing actor must have



(a) MRSDF graph.



(b) Functional ordering of firings, represented as a directed acyclic graph.

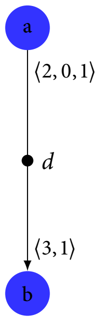
Figure 3.2 – The functional ordering of firings of actors follows from properties of the channels. The rates and initial tokens determine which producing firing precedes a particular consuming firing.

completed at least as many firings such that the consuming firing leaves a non-negative number of tokens, on each of its incoming channels. We formalise this dependency between the number of completed consuming firings on the one hand, and the required number of producing firings on the other hand, into the concept of a (firing) *predecessor function*. This function generalises the above formulation from channels to *paths*. It is defined as follows:

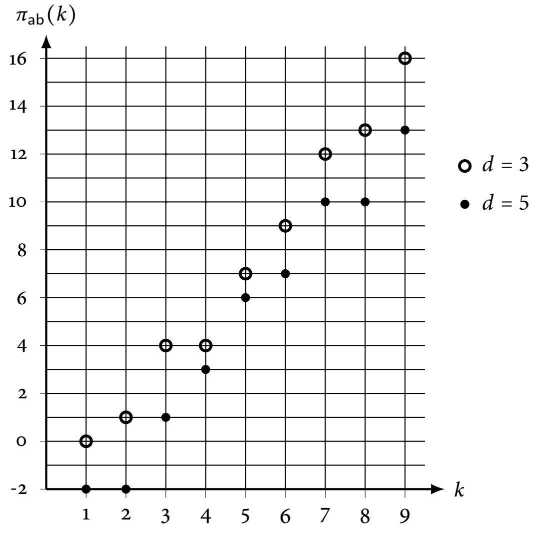
Definition 3.2 (Firing predecessor function). *The* firing predecessor function *associated with a path $P = v_1 \ldots v_n$ gives the number of* producing *firings of v_1 that must precede the k^{th}* consuming *firing of v_n, such that a* minimum, *non-negative number of tokens is left on each of the path's channels. Formally,*

$$\begin{aligned} \pi_{vw}(k) &= \min \{ m \in \mathbb{Z} | \Delta_{vw}(m,k) \geq 0 \}, & \\ \pi_{v_1 \ldots v_n}(k) &= (\pi_{v_1 v_2} \circ \pi_{v_2 \ldots v_n})(k), & \textit{if } n > 2. \end{aligned} \tag{3.2}$$

As an illustration of the (firing) predecessor function, consider its graphical representation, shown in Figure 3.3. If there are three initial tokens on the channel of Figure 3.3(a), then no producing firings of actor a need completion to start the first firing of actor b. We thus have (for $d = 3$) $\pi_{ab}(1) = 0$. As another example, consider the third firing of actor b. If $d = 5$, then the first two firings of b can start, but the third requires the production of two more tokens by a. These two tokens are produced by the first firing of a, thus $\pi_{ab}(3) = 1$. Note that $\pi_{ab}(k)$ may be negative. A negative value indicates that "undoing" one or more producing firing still leaves sufficiently many tokens for the k^{th} consuming firing. For example, for $d = 5$ the figure gives $\pi_{ab}(1) = \pi_{ab}(2) = -2$. This means that the last two firings



(a) CSDF channel.



(b) The channel's predecessor function, for different values of the initial tokens on the channel, given by d.

Figure 3.3 – A CSDF channel and a graphical representation of its predecessor function.

(these correspond to the second and third phase) of a can be undone, which reduces the number of tokens on ab, by one, from five to four. These four tokens are still sufficient to start the first two firings of a.

An obvious property of $\pi_{vw}(k)$ is that it is non-decreasing in k. This is a result of the fact that firing an actor never results in the removal of tokens from its outgoing channels. The firing predecessor function is a so-called *resource-class-wise affine* function (see Definition A.1): the increase in $\pi_{vw}(k)$ follows a repetitive staircase pattern, the shape of which is non-trivially determined by the rates and initial tokens associated with the channel. We use this repetitive nature to simplify the predecessor function, by rewriting the free variable m into an integer number of periods, and a non-negative remainder. That is, we substitute $m'\varphi_v + i$ for the free variable m in the set-builder notation of Definition 3.2. This allows the predecessor function to be written as the minimum over a number of *ceiling* terms, in the following way:

$$\begin{aligned}
\pi_{vw}(k) &\overset{(3.1)}{=} \min\left\{m \in \mathbb{Z} \middle| \left\lfloor \frac{m}{\varphi_v} \right\rfloor P^{\Sigma+}_{vw} + \Delta_{vw}(m \bmod \varphi_v, k) \geq 0\right\} \\
&\overset{(m \to m'\varphi_v + i)}{=} \min_{i<\varphi_v} \left\{\min\left\{m' \in \mathbb{Z} \middle| m' P^{\Sigma+}_{vw} + \Delta_{vw}(i,k) \geq 0\right\} \varphi_v + i\right\} \qquad (3.3) \\
&\overset{(A.1b)}{=} \min_{i<\varphi_v} \left\{\left\lceil \frac{-\Delta_{vw}(i,k)}{P^{\Sigma+}_{vw}} \right\rceil \varphi_v + i\right\}.
\end{aligned}$$

Each firing of an actor has, per incoming channel, a uniquely defined predecessor. The inverse relation, however, is not well-defined, as the predecessor function is not injective. That is, if $\pi_{vw}(k) = \pi_{vw}(m)$, then this does not imply that $k = m$. In order to allow some reasoning over the inverse relation between the domain and co-domain of π, we define a *pseudo-inverse* of the firing predecessor function, which we name *firing successor function* and define as follows.

Definition 3.3 (Firing successor function). *Let $p = v_1 \ldots v_n$ be a path in an SDF graph. The* firing successor function, *associated with p and denoted σ_p, maps a firing of v_1 to the first firing of v_n that depends on it:*

$$\sigma_p(m) = \min\left\{k \in \mathbb{Z} \middle| \pi_p(k) = m\right\}.$$

We conclude this section with a property that is heavily relied on in forthcoming chapters. As mentioned earlier, the predecessor function follows an increasing and repetitive staircase pattern. If the SDF graph is consistent, then the predecessor function is *residue-class-wise affine*, modulo the repetition vector entry of the consuming actor. This is proven in the proposition that follows. We return to this property in the following section.

Proposition 3.1 (π and σ are residue-class-wise affine mappings). *Let $\mathcal{G}$ be a consistent SDF graph. The predecessor and successor function, associated with any path in $\mathcal{G}$, are both residue-class-wise affine, modulo the repetition vector entry of, respectively, the last and first actor in the path. That is, for any path $P = v_1 v_2 \ldots v_n$, the following holds:*

$$\begin{aligned}
\pi_{v_1 \ldots v_n}(k + m q_{v_n}) &= \pi_{v_1 \ldots v_n}(k) + m q_{v_1}, \\
\sigma_{v_1 \ldots v_n}(k + m q_{v_1}) &= \sigma_{v_1 \ldots v_n}(k) + m q_{v_n}.
\end{aligned}$$

Proof. We prove the property for the predecessor function, π. Let vw be a channel in $\mathcal{G}$, and let $q_v = r_v \varphi_v$ and $q_w = r_w \varphi_w$. We have:

$$\begin{aligned}
\Delta_{vw}(i, k + m q_v) &\overset{(3.1)}{=} \Delta_{vw}(i,k) - m r_w P^{\Sigma-}_{vw} \\
&\overset{(2.1)}{=} \Delta_{vw}(i,k) - m r_v P^{\Sigma+}_{vw},
\end{aligned}$$

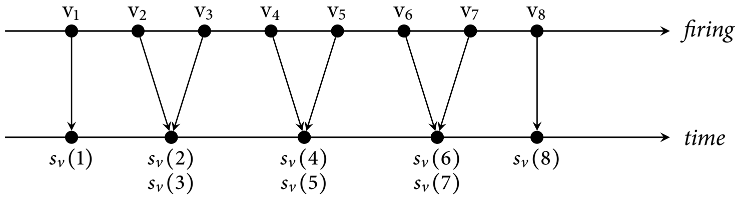


Figure 3.4 – The temporal ordering of the firing times of actors preserves their functional ordering. Whereas firings are *strictly* ordered, the times at which they occur are not; multiple firings may complete simultaneously.

and, consequently, (3.3) gives $\pi_{vw}(k + mq_w) = \pi_{vw}(k) + mq_v$. The proposition readily follows by induction over the channels in P. The proof for the successor functions follows in a similar fashion. $\square$

As a direct consequence of the above proposition, it follows that the *composition* of the (firing) successor and predecessor function is residue-class-wise affine modulo the repetition vector entry of the *last* actor in the path. If $p = v_1 \dots v_n$ is a path in a consistent SDF graph, then the following holds:

$$\sigma_p(\pi_p(k + q_{v_n})) = \sigma_p(\pi_p(k) + q_{v_1}) = \sigma_p(\pi_p(k)) + q_{v_n}. \tag{3.4}$$

3.2 Temporal dynamics

A firing of an actor involves the consumption of tokens from its incoming channels, and the production of tokens onto its outgoing channels. When reasoning over the functional determinacy of sequences of actor firings, only the order in which firings occur is relevant: firings are treated as indivisible, no notion of time is involved.

We now move from the functional ordering of firings to a *temporal* one, which interrelates the times at which firings may take place. The mapping from firings to time should be order-preserving. That is, if one firing precedes another in the functional ordering (see Definition 3.1), then the former must not occur later in time than the latter. Production and consumption of tokens, as well as their transfer over a channel, is instantaneous. Consequently, whereas the functional ordering is *strict* (for any two related firings v_k and w_m we have either $v_k \prec w_m$ or $w_m \prec v_k$), the times at which firings occur is not; subsequent firings may start and complete simultaneously (see Figure 3.4).

We can associate several events with a firing. There is the *start* of the firing, which involves with the consumption of tokens, and its *completion*, which coincides with the production of tokens. Furthermore, we may consider the production of each of the individual tokens as a separate event as well. In this section, we derive, from the functional ordering described in the previous section, a temporal ordering of different events associated with a firing. This gives rise to two mathematical characterisations of an SDF graph. Both are linear dynamical systems, formulated in max-plus algebra. These systems are named, in order of appearance, the *actor firing* and *token transfer* perspective. The former follows from the functional ordering defined in the previous section. In the latter, we derive, from the functional ordering of actor firings, a temporal ordering on the times at which *individual tokens* are transferred over a channel.

3.2.1 The actor firing perspective

The previous section treated firings as indivisible and instantaneous events. Firings, however, take time. Whereas the functional ordering of firings is *strict*, their temporal ordering is not: As the production and consumption of tokens is instantaneous, firings may complete or start at the same time.

Actors may start a firing once sufficiently many tokens are available for consumption. A firing starts with the consumption of tokens, and completes with the production of tokens. Tokens may only be produced as soon as all (functionally) preceding firings have completed. As a result, the execution time associated with an actor firing only gives a *minimum duration* of the firing. If a firing $k+1$ of an actor starts simultaneously with firing k, but has a shorter execution time, then it must postpone the production of its tokens until firing k has completed.

It is more straightforward to express the above constraint in terms of *completion* times of firings, rather than *start* times. If we denote by $t_v(k)$ the time at which actor v *completes* its k^{th} firing, then we have:

$$v_k \prec v_m \Rightarrow t_v(k) \leq t_v(m).$$

The firing completion times of an actor thus preserves their functional ordering. In a similar fashion, the constraints on firing (completion) times of actors connected by a channel follow from Definition 3.1. Again, the temporal ordering preserves the functional ordering, and takes into account the execution time of the firing. Token travel along channels is instantaneous; tokens become available for consumption as soon as they are produced. This gives the following constraint, which is imposed by every channel vw in the SDF graph (recall that $\tau_w(m)$ denotes the execution time of the m^{th} firing of actor w):

$$\forall k, m : v_k \prec w_m \Rightarrow t_v(k) + \tau_w(m) \leq t_w(m).$$

Together, these two constraints give rise to the following definition..

Definition 3.4 (Admissible schedule). *Let $\mathcal{G}$ be an SDF graph, and t a function that maps firings, of each actor in $\mathcal{G}$, to time, where $t_v(k)$ denotes the time at which actor v completes its k^{th} firing. Function t is called an* admissible schedule *if the following relation holds for all actors v in $\mathcal{G}$:*

$$\forall k, m \in \mathbb{Z} : k \geq m \Rightarrow t_v(k) \geq t_v(m), \tag{3.5}$$

and the following relation holds for all channels vw in $\mathcal{G}$:

$$\forall k, m \in \mathbb{Z} : \pi_{vw}(k) \geq m \Rightarrow t_w(k) \geq t_v(m) \otimes \tau_w(k). \tag{3.6}$$

The set of constraints imposed by the channels of a graph, on the firing completion times of its actors, can be expressed as linear recurrence relations in max-plus algebra. The *actor firing perspective* gives a mathematical characterisation of the interdependencies between the *firing times* of actors, formulated as a set of recurrence relations in max-plus algebra.

Definition 3.5 (Actor firing perspective). *Let $\mathcal{G}$ be an SDF graph with m source actors $u_1, \dots u_m$ and p sink actors $y_1, \dots, y_p$. Furthermore, let $\mathcal{G}_{in}$ and $\mathcal{G}_{out}$ be subgraphs of $\mathcal{G}$ that are induced by channels that are incident to, respectively, source actors and sink actors, and $\mathcal{H} = \mathcal{G} - \mathcal{G}_{in} - \mathcal{G}_{out}$. The* actor firing perspective *of $\mathcal{G}$ is the max-plus system defined by the following set of higher-order recurrence relations:*

$$\begin{aligned} t_w(k) &= \bigoplus_{vw \in \mathcal{H}} t_v\left(\pi_{vw}(k)\right) \otimes \tau_w(k) \oplus t_w(k-1) \oplus \bigoplus_{u_i w \in \mathcal{G}_{in}} u_i\left(\pi_{u_i w}(k)\right) \otimes \tau_w(k), \\ y_j(k) &= \bigoplus_{v y_j \in \mathcal{G}_{out}} t_v\left(\pi_{v y_j}(k)\right), \end{aligned}$$

where $u_i(k)$ and $y_j(k)$ denote the times at which the source, respectively sink actors, complete their k^{th} firing, and $t_v(k)$ denotes the times at which actor $v \in \mathcal{S}$ completes its k^{th} firing. Furthermore, $t_v(k) = \varepsilon$ for all $v \in \mathcal{H}$ and $k \leq 0$, and $u_i(k) \geq u_i(k-1)$ for all k.

In the mathematical characterisation given by Definition 3.5, we regard an SDF graph as a *discrete event system*. The events in this system are the completion times of actor firings. The actor firing perspective is thus the dynamical max-plus system describing the evolution of event times of the discrete event system. Note that in this perspective, the temporal dynamics are slightly different from the usual notion of self-timed execution, which reflects the dataflow principle: an actor may fire as soon as sufficiently many tokens are available. If we were to apply this notion to graphs that have source actors, then these source actors, since they have no incoming channels, would fire infinitely often, in zero time. In a system-theoretic perspective, however, the firing times of source actors serve as an external input, from which the firing times of other actors follow. In this view, a true data-driven execution, in the sense that actors may fire as soon as they are enabled, is obtained by setting the input u to a sequence of *zeroes*, i.e., in the max-plus sense, so $u(i) = \varepsilon$, for all $i \in \mathbb{Z}$.

Note that the above definition applies to graphs that have source and sink actors. In the view of Definition 3.5, we regard graphs *without* sources and sinks as an *autonomous* system. Chapter 6 is dedicated to the performance analysis of autonomous systems.

Of particular interest is the relation between the input and output of the system given by Definition 3.5. For this, we must consider the dependencies between firing (completion) times of sink actors on the one hand, and source actors, on the other hand. These dependencies follow from the *transitive closure* of the constraints given by Definition 3.4. This means that constraints imposed by channels generalise naturally to constraints imposed by paths. The following function gives this generalisation.

Definition 3.6 (Predecessor latency). *Let $\mathcal{G}$ be an SDF graph, and $p = v_1 \dots v_n$ a path in $\mathcal{G}$, with $n \geq 2$. The* predecessor latency *associated with p is a function that maps the k^{th} firing of actor v_n to the minimum amount of time between the completions of and that firing, and firing $\pi_p(k)$ of actor v_1, imposed by the execution times of actors on p.*

$$\begin{aligned} T_{vw}(k) &= \tau_w(k), \\ T_{v_1,\dots,v_n}(k) &= \tau_{v_n}(k) \otimes T_{v_1\dots v_{n-1}}\left(\pi_{v_{n-1}v_n}(k)\right), \quad \textit{if } n > 2. \end{aligned}$$

For consistent SDF graphs, the predecessor latency function is a periodic function. The total execution time along a path from a firing k of actor v to its predecessor, varies periodically in k, with the period given by the repetition vector entry of v. This property follows from the periodicity of the execution time of an actor, and the fact that the firing predecessor function is *resource-class-wise affine*, as is stated by the following proposition.

Proposition 3.2 (Periodicity of predecessor latency). *Let $\mathcal{G}$ be a consistent SDF graph, with repetition vector q. The predecessor latency associated with any path $p = v_1 \dots v_n$ in $\mathcal{G}$, is a periodic function, with period q_{v_n}. That is:*

$$T_p(k + q_{v_n}) = T_p(k).$$

Proof. Observe that the execution time of an actor is periodic in the number of phases of that actor. In particular, this gives: $\tau_w(k+mq_w) = \tau_w(k)$. The proposition now follows directly from Proposition 3.1. □

The predecessor latency function is derived from the transitive closure of Equation 3.6. As such, it gives rise to a *necessary* condition for firing completion times, but not a sufficient one. A condition that is both necessary and sufficient is obtained by taking into account Equation 3.5 as well. If two consecutive firings of the same actor share the same predecessor, (i.e., $\pi_{vw}(k) = \pi_{vw}(k-1) = m$), then the minimum time between the k^{th} firing of w and the m^{th} firing of v is given by the *maximum* of the predecessor latencies of firings k and $k-1$ of actor w. The following proposition formalises this reasoning in the relationship between the input and output of the actor firing perspective.

Proposition 3.3 (Input-output relationship). *Let $\mathcal{G}$ be an SDF graph, with n sources $u_1, \dots u_n$, and m sinks $y_1, \dots, y_m$. Furthermore, let $\mathcal{P}_{ij}$ denote the set of paths in $\mathcal{G}$, from u_i to y_j. The j^{th} output of the actor perspective of $\mathcal{G}$ is given by:*

$$y_j(k) = \bigoplus_{i=1}^{n} \bigoplus_{p \in \mathcal{P}_{ij}} \left(u_i \left(\pi_p(k) \right) \otimes \bigoplus_{l=k}^{(\sigma_p \circ \pi_p)(k)} T_p(l) \right).$$

Proof. The proposition follows from the transitive closure of the constraints that hold for an admissible schedule, as expressed through the predecessor and predecessor latency functions. □

From the relationship between the input and output of the actor perspective, we derive two important properties. The first property of the system given by Definition 3.5 is *linearity*. Linearity means that the output of a system satisfies the superposition principle: if $y_1(t)$ and $y_2(t)$ are the system's responses to inputs $x_1(t)$ and $x_2(t)$, then its response to $ax_1(t) + bx_2(t)$ is $ay_1(t) + by_2(t)$. In the max-plus sense, linearity means a *propagation of delay*: delaying a particular firing of input u by $\max\{\alpha, \beta\}$ time units causes the corresponding firing of output y to be delayed by $\max\{\alpha, \beta\}$ time units as well.

A second property of the actor firing perspective relates to the output of the system as a response to a *shift* in the input sequence. For consistent SDF graphs, specific shifts in the input sequence cause an identical output, but one that is shifted as well. The two properties are formalised in the following proposition.

Proposition 3.4 (SDF and linear periodically shift-varying systems). *The actor firing perspective of a consistent SDF graph is a linear and periodically shift-varying system.*

Proof. To prove linearity, let y_1 and y_2 be outputs of the system, due to inputs u_1 and u_2. That is, for $i \in \{1, 2\}$ we have:

$$y_i(k) = \bigoplus_{p \in \mathcal{P}} u_i \left(\pi_p(k) \right) \otimes \bigoplus_{i=k}^{(\sigma_p \circ \pi_p)(k)} T_p(i).$$

The response to the system to a linear combination of the inputs, i.e., $a \otimes u_1 \oplus b \otimes u_2$, with $a, b \in \mathbb{R}_{\max}$, is then given by:

$$\begin{aligned}
&\bigoplus_{p \in \mathcal{P}} \left(a \otimes u_1 \left(\pi_p(k) \right) \oplus b \otimes u_2 \left(\pi_p(k) \right) \right) \otimes \bigoplus_{i=k}^{(\sigma_p \circ \pi_p)(k)} T_p(i) \\
&= \left(a \otimes \bigoplus_{p \in \mathcal{P}} u_1 \left(\pi_p(k) \right) \otimes \bigoplus_{i=k}^{(\sigma_p \circ \pi_p)(k)} T_p(i) \right) \oplus \left(b \otimes \bigoplus_{p \in \mathcal{P}} u_2 \left(\pi_p(k) \right) \otimes \bigoplus_{i=k}^{(\sigma_p \circ \pi_p)(k)} T_p(i) \right) \\
&= a \otimes y_1(k) \oplus b \otimes y_2(k)
\end{aligned}$$

The response of the system to a linear combination of inputs is thus equal to the linear combination of responses to each of the individual inputs. In other words, the system is linear.

To show that the system is periodically shift-varying, consider any path p from an input (source) actor, u_i to an output (sink) actor, y_j. By Proposition 3.1, the predecessor function associated with p satisfies $\pi_p(k + mq_{y_j}) = \pi_p(k) + mq_{u_i}$. As before, let $\mathcal{P}$ denote the set of all paths from source actor u to sink actor y. By Proposition 3.3, the input-output relationship is given by:

$$\begin{aligned} y(k+q_y) &= \bigoplus_{p\in\mathcal{P}} u(\pi_p(k+q_y)) \otimes \bigoplus_{i=k+q_y}^{(\sigma\circ\pi)(k+q_y)} T_p(i) \\ &= \bigoplus_{p\in\mathcal{P}} u(\pi_p(k)+q_u) \otimes \bigoplus_{i=k}^{(\sigma\circ\pi)(k)} T_p(i). \end{aligned}$$

In words, shifting the input sequence by q_{u_i} causes a shift in the output sequence of q_{y_j}. The system between source actor u_i and sink actor y_j thus corresponds to a (q_{u_i}, q_{y_j})-shift-invariant system, i.e. a periodically shift-varying system. □

Note that by Proposition 3.4, any HSDF graph corresponds to a *shift-invariant* system: in an HSDF graph, each actor has a repetition vector entry of one. Shifting the input by one thus causes the output to be shifted by one. Any shift in the input sequence thus causes the output sequence to be equally shifted - the system is thus shift-invariant. Transforming an SDF graph into its single-rate equivalent is thus analogous to transforming a periodically shift-varying system into a shift-invariant one. We describe this perspective on transforming a CSDF graph into an equivalent HSDF graph, in more detail, in Section 4.3.

3.2.2 The token transfer perspective

There is a natural counterpart to the actor firing perspective described in the previous section. The completion of an actor firing involves the production of tokens onto the actor's outgoing channels. This production of tokens is instantaneous; the completion of the firing coincides with the production of each of the corresponding tokens. Instead of describing the evolution of firing completion times, we may thus analogously describe the evolution of token production times. The relation between the completion of an actor firing and the tokens it produces are given by:

$$\psi_{vw}(k) = \min\{m \in \mathbb{Z} | \Delta_{vw}(m, 0) - \delta_{vw} \geq k\}. \tag{3.7}$$

Function $\psi_{vw}(k)$ gives the number of firings that must be completed by the producing actor v, such that (at least) k tokens are produced onto vw.

As a natural dual to the actor firing perspective described in the previous section, we introduce the concept of a *predecessor* function. This function gives the number of *tokens* that need to be produced on an upstream channel, such that a given number of tokens can be produced onto a downstream channel. Its definition follows the same principle that underlies Definition 3.2: the number of tokens that remains on each channel, after the necessary firings have completed, must be non-negative. We use a black dot to distinguish this transfer predecessor function from the firing predecessor function defined earlier.

Definition 3.7 (Transfer predecessor function). *The* transfer predecessor function *gives the number of tokens that must be produced onto the first channel of a path, such that k tokens may be produced onto the path's last channel. The transfer predecessor function associated with path $P = v_1 \dots v_n$ is denoted $\pi^\bullet_P$ and defined for $n \geq 3$ by:*

$$\begin{aligned} \pi^\bullet_{uvw}(k) &= -\Delta_{uv}\left(0, \psi_{vw}(k)\right), \\ \pi^\bullet_{v_1\dots v_n}(k) &= \left(\pi^\bullet_{v_1v_2v_3} \circ \pi^\bullet_{v_2\dots v_n}\right)(k), \qquad \textit{if } n > 3. \end{aligned} \tag{3.8}$$

Figure 3.5 shows the transfer predecessor function associated with the path of two CSDF channels ab and bc. As an example, consider the eighth token produced onto bc, by the sixth firing of b. Up to and including this firing, eight tokens are consumed from ab. In order to ensure that this leaves a non-negative number of tokens on ab, in case $d = 2$, six tokens must be produced onto the channel by actor a. For $d = 2$, we thus have $\pi^\bullet_{abc}(8) = 6$. The figure furthermore shows how tokens that are produced by the same firing of b all map to the same value: for $d = 0$, we have $\pi^\bullet_{abc}(1) = \pi^\bullet_{abc}(2) = 1$ and $\pi^\bullet_{abc}(5) = \pi^\bullet_{abc}(6) = 5$. Also, for $d = 2$, the transfer of the fifth, sixth and seventh token over bc each requires the transfer of three tokens over channel ab.

The graph of Figure 3.5 shows a number of similarities with the graph of the firing predecessor function, shown earlier in Figure 3.3. Analogous to its counterpart, the transfer predecessor function is a resource-class-wise affine function. Whereas the former is resource-class-wise affine modulo the repetition vector entry of the consuming actor, the latter is resource-class-wise affine modulo the *number of tokens*, consumed in a single graph iteration, from the last channel in the path:

Proposition 3.5 ($\pi^\bullet$ is a residue-class-wise affine mapping). *Let $\mathcal{G}$ be a consistent SDF graph, and $P = v_1v_2 \dots v_n$ a path in $\mathcal{G}$, with $n \geq 3$. The transfer predecessor function associated with P is residue-class-wise affine modulo the number of tokens consumed, during a single graph iteration, from the last channel in P. Formally,*

$$\pi^\bullet_{v_1\dots v_n}\left(k + m\frac{P^{\Sigma-}_{v_{n-1}v}q_{v_n}}{\varphi_{v_n}}\right) = \pi^\bullet_{v_1\dots v_n}(k) + m\frac{P^{\Sigma+}_{v_1v_2}q_{v_1}}{\varphi_{v_1}}. \tag{3.9}$$

The two predecessor functions, given by Definitions 3.2 and 3.7, are closely related. This relationship can be stated formally in terms of (3.7). For a path that consists of at least two channels, there are two equivalent ways to relate the number of firings that must be completed, by the first actor of the path, to the number of tokens

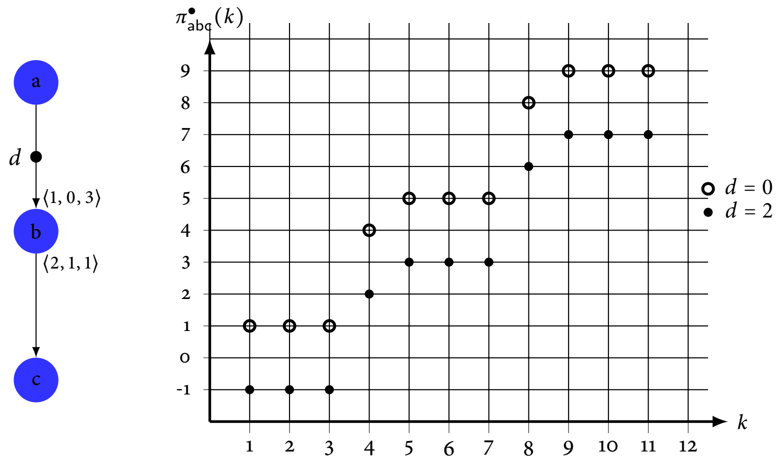


Figure 3.5 – The transfer predecessor function associated with path abc, for different values of the number of initial tokens on ab, as specified by d.

that can be produced, onto the last channel. These two ways are expressed by the following equality:

$$\psi_{v_1 v_2}\left(\pi^\bullet_{v_1 \dots v_n}(k)\right) = \pi_{v_1 \dots v_{n-1}}\left(\psi_{v_{n-1} v_n}(k)\right), \tag{3.10}$$

for all paths $P = v_1 \dots v_n$ with $n \geq 3$.

In a procedure similar to the one followed for the firing predecessor function in Section 3.1, we reformulate the transfer predecessor function $\pi^\bullet_{uvw}$ as a minimum over as many terms as actor v has phases. Substituting the integer variable m that appears in (3.8) with an integer number of periods, m', and a phase i, gives:

$$\begin{aligned}
\pi^\bullet_{uvw}(k) &= -\Delta_{uv}\left(0, \min\left\{m \in \mathbb{Z} \,\middle|\, \sum_{l=1}^{m} \rho^+_{vw}(l) \geq k\right\}\right) \\
&= -\Delta_{uv}\left(0, \min_{i<\varphi_v}\left\{\min\left\{m' \in \mathbb{Z} \,\middle|\, m' P^{\Sigma+}_{vw} + \sum_{l=1}^{i} \rho^+_{vw}(l) \geq k\right\}\varphi_v + i\right\}\right) \\
&= -\Delta_{uv}\left(0, \min_{i<\varphi_v}\left\{\left\lceil \frac{k - \sum_{l=1}^{i} \rho^+_{vw}(l)}{P^{\Sigma+}_{vw}} \right\rceil \varphi_v + i\right\}\right).
\end{aligned}$$

As a next and final step, we use the fact that $\Delta(i, j)$ decreases monotonically in j. This allows us to eliminate Δ from the above, rewriting it into:

$$\pi^\bullet_{uvw}(k) = \min_{i<\varphi_v}\left\{\left\lceil \frac{k - \sum_{l=1}^{i} \rho^+_{vw}(l)}{P^{\Sigma+}_{vw}} \right\rceil P^{\Sigma-}_{uv} + \sum_{l=1}^{i} \rho^-_{uv}(l) - \delta_{uv}\right\}. \tag{3.11}$$

In the token transfer perspective, we characterise the interdependencies between the times at which tokens are *transferred over* channels as a linear autonomous max-plus system. The inputs of this system control the times at which tokens may be transferred over the graph's channels. We assume that the sequence of inputs is non-decreasing. That is, the transfer of the k^{th} token does not occur earlier than the transfer of token $k-1$. Outputs of the system serve to "observe" the times of specific actor firings. The system is defined as follows:

Definition 3.8 (Token transfer perspective). *Let $\mathcal{G}$ be an SDF graph with m source actors $u_1, \dots, u_m$ and p sink actors $y_1, \dots, y_p$. Furthermore, let $\mathcal{G}_{in}$ be the subgraph of $\mathcal{G}$ induced by channels that are incident to a source actor, and $\mathcal{H} = \mathcal{G} - \mathcal{G}_{in}$. The* token transfer perspective *of $\mathcal{G}$ is the max-plus system that is defined by the following set of higher-order recurrence relations.*

$$\begin{aligned} t^{\bullet}_{vw}(k) &= \bigoplus_{uv \in \mathcal{G}} t^{\bullet}_{uv}\left(\pi^{\bullet}_{uvw}(k)\right) \otimes \tau_v(\psi_{vw}(k)) \oplus t^{\bullet}_{vw}(k-1), \\ t^{\bullet}_{u_i v}(k) &= u_i(\psi_{u_i v}(k)), \\ y_j(k) &= \bigoplus_{v y_j \in \mathcal{G}} t^{\bullet}_{v y_j}\left(-\Delta_{v y_j}(0,k)\right), \end{aligned}$$

where $t^{\bullet}_{vw}(k)$ denotes the time at which the k^{th} token is transferred over channel vw, and $y_j(k)$ (resp. $u_i(k)$) the time of the k^{th} firing of actor y_j (resp. u_i). Furthermore, $t^{\bullet}_{vw}(k) = \varepsilon$ for all $vw \in \mathcal{H}$ and $k \leq 0$, and $u_i(k) \geq u_i(k-1)$ for all k.

The token transfer perspective is a linear dynamical max-plus system. Furthermore, if the graph it describes is consistent, then the system is periodically shift-varying. This readily follows from the strong relationship with the actor firing perspective:

Proposition 3.6 (Linearity and shift-variance of the transfer perspective). *The token transfer perspective of a consistent SDF graph is a linear and periodically shift-varying system.*

Proof. The times at which tokens are transferred over channels coincide with the times at which corresponding producing firings complete. Since, by Proposition 3.4, the actor firing perspective of a consistent SDF graph is both linear and periodically shift-varying, it follows that the token transfer perspective is linear and periodically shift-varying as well. □

Analogous to the actor firing perspective, the token transfer perspective of an SDF graph is a max-plus system, with multiple inputs and outputs. Whereas the state of the system corresponds to times at which tokens are transferred, the inputs and outputs of the system map to *firing times* of (source and sink) actors. In this sense, the inputs and outputs of the two perspectives are equivalent. In Section 3.3, we prove their equivalence in a stronger sense.

3.2.3 Comparing the two perspectives

The actor firing and token transfer perspectives describe different but strongly related kinds of events in an SDF graph: the times at which actors complete their firings, and the times at which tokens that are produced by these firings are transferred over channels. For an HSDF graph, these two kinds of events are indistinguishable, as each actor firing corresponds with the transfer of a single token over each of the actor's outgoing channels. As a result, for HSDF graphs, the two perspectives are identical. In MRSDF graphs, multiple tokens are transferred over a channel simultaneously. The constraints on the transfer times of these tokens are identical, and thus the token transfer perspective adds some redundancy over the actor firing perspective. In CSDF graphs, some firings may produce no tokens at all. For these graphs, the difference between the two perspectives, in terms of redundancy, depends on the rate vectors of the graph's channels.

A clear difference between the two perspectives is the size of the systems in the two perspectives. The actor firing perspective has one recurrence relation for each *actor*, whereas the token transfer perspective has one relation for each *channel*. The token transfer perspective is thus at least as large, in terms of system size, as the actor firing perspective, and typically larger.

Whether one perspective should be preferred over the other depends on the context in which they are used. In the firing perspective, multiple token transfers are grouped into a single, indivisible, event: the firing that produces them. In the token perspective, individual token transfers are separate events. This fits the view that is taken in several approaches where the focus lies on *token arrival curves* [48, 95]. In particular, it allows one to *approximate* the temporal dynamics of the graph by assuming conservative linear bounds on these curves. Such an approximation may be applied to the actor firing perspective as well, but by constructing linear bounds on curves that describe the completion times of consecutive firings. We cover the differences in the approximations obtained from each of the perspectives in more detail in Chapter 5.

3.3 Equivalent systems

Classical system theory studies the relationship between the *input* of a system, and its *output*. The system is treated as a black box: when observing two systems that are fed with the same input, it may be impossible to tell them apart. For example, a system that models an electronic circuit can be replaced by an equivalent system by combining multiple resistors into a single, resultant, resistance (see Figure 3.6); when feeding the circuit with the same voltage, the observed current is the same for both systems. This indicates that the two configurations of resistors are *equivalent*: the series-parallel configuration in Figure 3.6(a) has the same resistance as the single resistor in Figure 3.6(b).

For discrete event systems formulated in max-plus algebra, inputs and outputs correspond to *times* at which events occur. By studying the relationship between

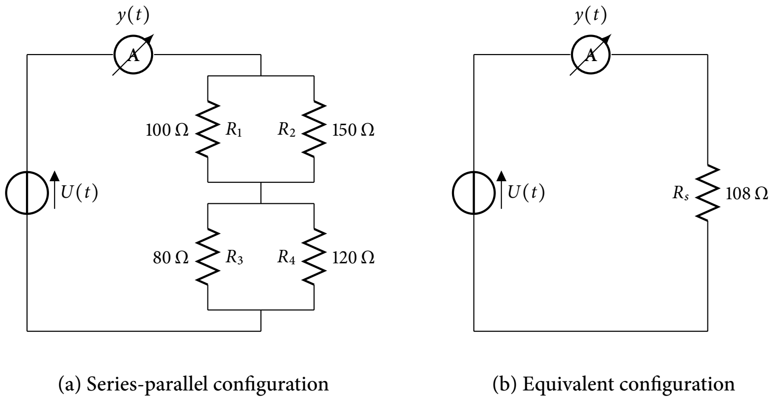


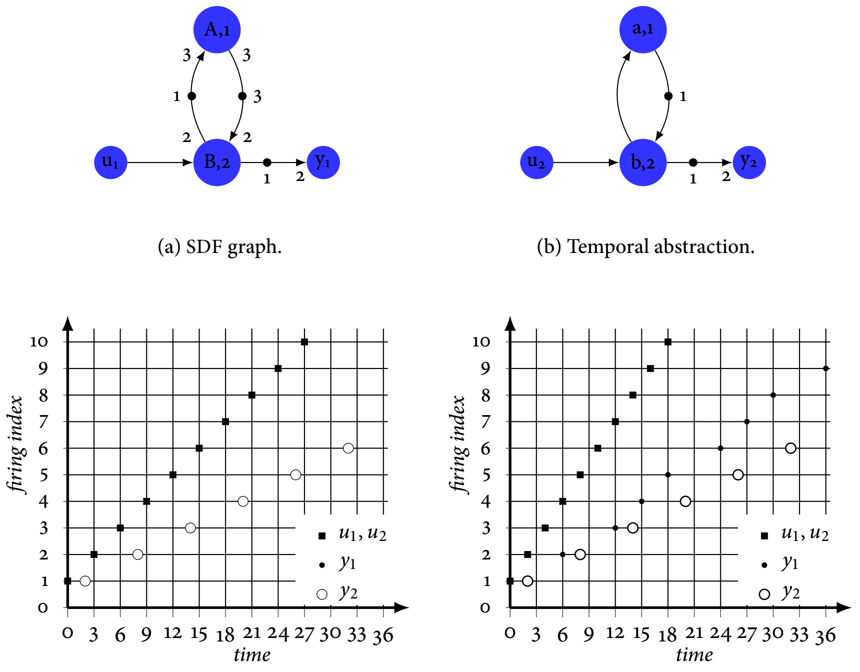
Figure 3.6 – An example of two equivalent systems: if we consider the voltage as the system's input, and the current as its output, then both systems produce the same output when fed with the same input.

such a system's inputs and outputs, we gain insight in the performance of the system. That is, the input-output behaviour gives the time the system needs to provide a response to some input, and the rate at which it can process input events. For example, a source actor in an SDF graph could represent the periodic arrival of frames in a digital movie, and a sink actor the refresh of the screen on which the movie is to be displayed. When analysing the output in terms of the input, the SDF graph that models the (temporal dynamics of the) processing of the movie frame, is treated as a black box. If we have another movie-processing graph that, when fed with the same frame arrival times, always gives the same screen refresh times, we may conclude that the two systems have an equivalent performance. In the following sections, we give a formal definition for this temporal notion of equivalence of SDF graphs. Similar to how the equivalence of the two electronic circuits implied an equivalent resistance, we describe how equivalence of max-plus systems relates to the schedules of the SDF graphs they describe.

We relate two systems to each other by comparing their outputs, while feeding them with the same sequence of inputs. Two systems are equivalent if they have the same output when fed with the same input. Before we give a formal definition, we introduce a weaker notion of equivalence, which we name *temporal abstraction*.

Definition 3.9 (Temporal abstraction). *Let S_1 and S_2 be max-plus linear systems, with respective inputs u_1 and u_2, and respective outputs y_1 and y_2. System S_1 is a temporal abstraction of S_2, if output events of S_1 never occur sooner than corresponding output events of S_2. That is, $S_1 \sqsubseteq_T S_2$ if the following holds:*

$$\forall k \in \mathbb{Z} : u_1(k) = u_2(k) \Rightarrow y_1(k) \oplus y_2(k) = y_1(k),$$



(a) SDF graph.

(b) Temporal abstraction.

(c) Response, $T = 3$.

(d) Response, $T = 2$.

Figure 3.7 – The graph shown in (b) temporally abstracts the graph depicted in (a). The source actors fire simultaneously and strictly periodically, with a period of T. For $T = 3$, both systems have identical output sequences: the sink actors fire simultaneously.

In terms of performance, a system that is a temporal abstraction of another system, *conservatively* models that other system: the performance of the former is never better than the performance of the latter. A temporal abstraction relates both the *latency* (that is, the difference between the k^{th} firing times of input and output actors) and the *rates* at which output actors fire of two graphs. If a system is temporally abstracted by another system, then this reveals some information about the maximum possible rates at which events in the systems can occur, in the following way. Suppose $\mathcal{G} \sqsupseteq_T \mathcal{H}$. This means in particular that even the output produced by $\mathcal{G}$ at its fastest possible rate, never precedes (in time) the corresponding output of $\mathcal{H}$. In other words, $\mathcal{H}$ can always "keep up" with the rate of $\mathcal{G}$, and thus the maximum rate at which $\mathcal{H}$ can produce data must be at least as high as the maximum rate $\mathcal{G}$.

As an example of a temporal abstraction, consider Figure 3.7. The figure shows an SDF graph (Figure 3.7(a)) that is temporally abstracted by another SDF graph (Figure 3.7(b)). Output (i.e., sink actor) $y_2(k)$ never occurs earlier than output

$y_1(k)$. When fed with strictly periodic inputs, $u_1(k) = u_2(k) = T(k-1)$, both systems give an identical output for $T = 3$. This is shown in Figure 3.7(c). Increasing the rate at which input events occur only affects the output of the leftmost system (see Figure 3.7(d)).

If the rate at which input events occur is low enough, then this input rate determines the rate at which output events occur. That is, firings of output actors in both the temporally abstracted graph and the abstraction occur at the same rate. The latency imposed by the abstraction is at least as high as the latency imposed by the abstracted graph. This finds a useful application in the *approximation* of performance characteristics of graphs. If these characteristics are difficult to assess for some graph $\mathcal{G}$, but easy to obtain for graph $\mathcal{H}$ with $\mathcal{H} \supseteq_T \mathcal{G}$, then we may, for example, obtain a bound on the throughput of $\mathcal{G}$ from the throughput of $\mathcal{H}$. We further elaborate on this in Chapter 5.

We conclude this section with a formal definition and some examples of *temporal equivalence*. Two systems are temporally equivalent, if, when fed with the same input, their outputs are identical. In terms of the above definition of temporal abstraction, this means that two graphs are temporally equivalent if they temporally abstract each other.

Definition 3.10 (Temporal equivalence). *Two max-plus systems $\mathcal{S}_1$ and $\mathcal{S}_2$ are* temporally equivalent, *denoted $\mathcal{S}_1 \equiv_T \mathcal{S}_2$, if $\mathcal{S}_1$ temporally abstracts $\mathcal{S}_2$, and vice versa. That is, $\mathcal{S}_1 \equiv_T \mathcal{S}_2$ if $\mathcal{S}_1 \subseteq_T \mathcal{S}_2$ and $\mathcal{S}_2 \subseteq_T \mathcal{S}_1$.*

Examples of temporally equivalent systems are the actor firing and token transfer perspective defined earlier. Their equivalence follows from the strong relationship between the completion time of a firing, and the time at which the produced tokens are transferred over a channel.

Proposition 3.7 (Temporal equivalance of perspectives). *Let $\mathcal{G}$ be an SDF graph. The actor firing perspective and token transfer perspective of $\mathcal{G}$ are temporally equivalent.*

Proof. Tokens are produced and transferred over a channel instantaneously at the completion of a firing. Consequently, a token's transfer over a channel coincides with the completion of its producing firing. A token is mapped to its producing firing by (3.7). This means that we token transfer times and actor firings are related through:

$$t^{\bullet}_{vw}(k) = t_v(\psi_{vw}(k)).$$

Now, let $y^{\bullet}_j(k)$ denote the kth firing time of sink actor y_j, as determined by the token transfer perspective of $\mathcal{G}$. Similarly, let $y_j(k)$ denote the kth firing time of y_j in the actor firing perspective of $\mathcal{G}$. The equality of $y^{\bullet}_j(k)$ and $y_j(k)$ follows from the following rewriting steps:

$$\begin{aligned} y_j^\bullet(k) &= \bigoplus t_{vy_j}^\bullet\left(-\Delta_{vy_j}(0,k)\right) \\ &= \bigoplus t_v\left(\psi_{vy_j}\left(-\Delta_{vy_j}(0,k)\right)\right) \\ &= \bigoplus t_v\left(\pi_{vy_j}(k)\right) \\ &= y(k). \end{aligned}$$

Thus, the two perspectives are equal. □

As an example, Figure 3.8 shows two graphs that are temporally equivalent. To see this, observe that, in the MRSDF graph, every odd firing of the source actor, u_2, enables a next firing of s_1, and every even firing of u_2 enables a next firing of s_2. A similar relationship exists between output y_2 and s_1 and s_2: odd firings of y_2 are enabled by the next firing of s_1, and even firings are enabled by the next firing of s_2. As a result, a next firing of the output actor, y_2, requires a next firing of the source actor, u_2. The fastest possible rate at which the sink actors of both graphs can fire is the same for both graphs. In the HSDF graph, actor s may fire once every single time unit. Consequently, actor y_1 may fire at most once per time unit as well. The maximum firing rate of actors in the MRSDF graph is bounded by the loop $s_1 s_2 s_1$: actor s_1 and s_1 may both fire once every *two* time units. As a result, actor y_2 may fire at most once per time unit. In other words, if the firing times of the two source actors are given by $u_1(k) = u_2(k) = u(k)$, then the firing times of both sink actors are given by:

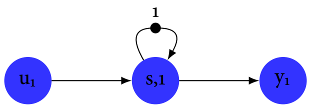
$$y_1(k) = y_2(k) = u(k) \otimes 1 \oplus 1^{\otimes k}.$$

and thus the systems are temporally equivalent.

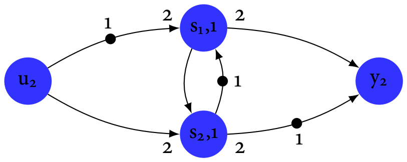
The system-theoretic view on SDF graphs gives rise to a notion of throughput that differs from the classical definition given in [40, 60]. Classically, throughput of an SDF graph is defined as the number of *graph iterations* (see Section 2.1.6) completed per time unit, in a self-timed execution of the graph. From a system-theoretic perspective, (maximum) throughput of a system relates to the input-output behaviour of the system: it quantifies the response (output) of a system to an infinitely high load. In this view, the classical definition of throughput falls short. In fact, we may construct two systems that are temporally equivalent but, by the classical definition, have different throughputs. This is illustrated in Figure 3.8: the throughput of the MRSDF graph is a half iteration per time unit, whereas the throughput of the HSDF graph is one iteration per time unit.

3.4 Discussion

The characterisation of the temporal dynamics of an SDF graph, as presented in this chapter, is based on two principles. The first is that the SDF graphs are *functionally determinate*: given the same sequence of input values, a sequence of actor



(a) HSDF graph.



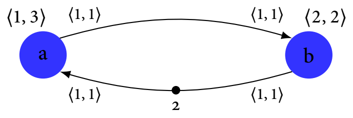
(b) Temporally equivalent MRSDF graph.

Figure 3.8 – Temporal equivalence does not imply equal throughput. The actor firing perspectives of the HSDF and MRSDF graphs are temporally equivalent. In the classical definition of throughput, the HSDF graph has a throughput of one, and the MRSDF graph has a throughput of one half.

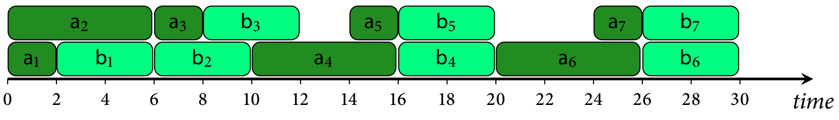
firings produces the same sequence of output values. In other words, tokens in the graph may not "overtake" each other. The second principle is the *dataflow principle*, which states that a computation may commence as soon as its input data is available. The actor firing and token transfer perspectives presented in this chapter are mathematical embodiments of these two principles, formulated in max-plus algebra.

Our mathematical characterisation differs from the semantics given by other authors. For example, the operational semantics formulated in [6, 40, 42, 60, 91] embed only the second principle, and do not enforce functional determinacy. For HSDF and MRSDF graphs, the second principle suffices to ensure functional determinacy. This follows from the fact that HSDF and MRSDF actors have non-varying execution times. As a result, firings that have started earlier than consecutive firings, complete earlier as well. For CSDF graphs where actors are permitted to fire auto-concurrently, these operational semantics lead to a temporal dynamics that differs from the dynamics given in this chapter. This is best illustrated by an example, which is shown in Figure 3.9. The graph shown in this figure has no self-loops. As a result, if sufficient tokens are present on an actor's input channels, an actor may start multiple firings simultaneously. Under the operational semantics described above, a self-timed execution allows the fifth firing of a to complete and produce its output tokens, before the fourth firing of a has completed. In other words, the dataflow process associated with a is not functionally determinate, as tokens have "overtaken" each other. The firing times given by the actor firing perspective satisfy the constraint imposed by functional determinacy: the completion of the fifth firing of a is postponed until the moment the fourth firing of a has completed. Functional determinism leads to a lower actor firing rate: the schedule of Figure 3.9(b) gives a throughput of $\frac{1}{5}$, whereas the schedule of Figure 3.9(c) gives a higher throughput, of $2/7$.

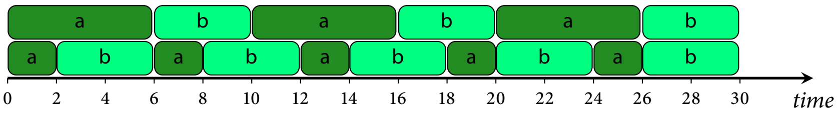
In the original definition of CSDF this problem is solved by letting each actor have



(a) A simple CSDF graph.



(b) Self-timed schedule, obeying functional ordering.



(c) Self-timed schedule when ignoring functional ordering.

Figure 3.9 – A CSDF graph in which actors are free to fire multiple firings simultaneously. When restricting executions to monotonic dataflow processes, maximum throughput is limited to $\frac{1}{5}$ (the last block of four firings forms a single period), as illustrated by the self-timed schedule shown in (b). Figure (c) shows the resulting self-timed schedule when we ignore the functional ordering. In this case, the graph's maximum throughput is $\frac{4}{15}$ (the schedule shows a single period), which is higher.

an implicit self-loop with a single-token. This restriction forbids auto-concurrency. Furthermore, due to this restriction, CSDF, in the definition of [11], does not strictly generalise MRSDF. This means that not every MRSDF graph is also a CSDF graph, which breaks the typically assumed taxonomy of SDF graphs.

A self-loop enforces a stronger constraint than functional determinacy. Weaker constraints may be enforced by carefully adding cycles to the graph. However, the weakest constraint that still ensures functional determinacy is difficult, if not impossible, to enforce solely by manipulating the graph. The actor firing and token transfer perspective defined in this chapter both allow auto-concurrent execution of firings of a CSDF actor. Furthermore, they enforce functional determinacy in a straightforward way, by including a constraint on the *completion time* of actor firings (or a similar constraint on the transfer times of tokens).

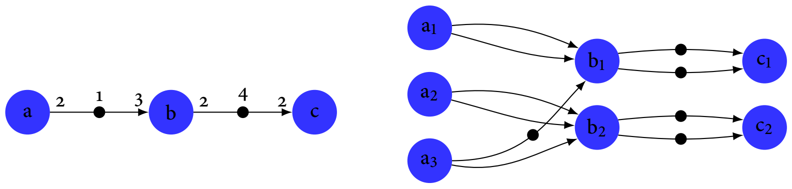
The system-theoretic perspective that we introduce in this chapter allows two SDF graphs to be compared in terms of their performance characteristics. The notions of *abstraction* and *equivalence* are useful as they allow for the computation of performance characteristics of a *complex* system, through the analysis of a *simpler* system. In the chapters that follow, equivalence and abstraction are the guiding principles in, respectively, the *transformation* and *approximation* of SDF graphs.

4 Synchronous dataflow graph transformations

Abstract – *For homogeneous synchronous dataflow graphs, efficient analysis techniques have been available for a few decades. Multi-rate and cyclo-static dataflow graphs may be analysed by transforming them into an equivalent homogeneous synchronous dataflow (HSDF) graph. This transformation, however, generally results in a much larger graph; in the worst case the size of the equivalent graph grows exponentially in the size of the original graph. In this chapter, we revisit the construction of an equivalent homogeneous synchronous dataflow (HSDF) graph, and present a transformation algorithm that produces smaller graphs. We furthermore present an algorithm to transform a cyclo-static dataflow (CSDF) graph into an equivalent multi-rate synchronous dataflow (MRSDF) graph.*

The different classes of SDF graphs introduced in Chapter 2, e.g. cyclo-static, multi-rate and homogeneous SDF, vary in the richness of their properties. Where for HSDF graphs these properties are restricted to execution times and intitial tokens, in a CSDF graph actors have execution time *vectors* and channels have, along with initial tokens, production and consumption rate *vectors*. The three classes are equally expressive, as both MRSDF and CSDF graphs may be transformed into an *equivalent* HSDF graph [11, 60, 62, 88]. Here, "equivalence" lacks a formal definition. In literature, intuitive notions of equivalence are used rather than formal ones. These notions stem from either a functional or a temporal perspective. To illustrate these different interpretations of the word equivalence, consider the CSDF graphs depicted in Figure 4.1 and Figure 4.2.

Large parts of this chapter have been published in [RdG:3].



(a) MRSDF graph. (b) Single-rate equivalent.

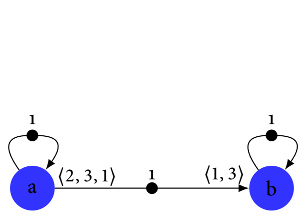
Figure 4.1 – The *de facto* transformation from MRSDF to HSDF, as originally proposed in [60, 62].

Both figures show a transformation of an SDF graph into an HSDF graph. In the first figure (Figure 4.1), the input to the transformation is an MRSDF channel. For every three firings of actor a, actor b may fire twice . A single iteration of the MRSDF channel thus consists of three firings of a and two of b. The HSDF graph constructed by the transformation has an actor for every firing that occurs in a single iteration. Furthermore, the graph has a channel for every token that is produced in a single iteration; since every firing of actor a produces two tokens, the three actors a_1, a_2 and a_3 each have two outgoing channels. Likewise, the two actor corresponding to firings of actor b each have three incoming channels. The resulting HSDF graph is called the *single-rate equivalent* of the MRSDF graph. Note that, as some tokens may be both produced by the same firing and consumed by the same firing, the single-rate equivalent is a *multi-graph*.

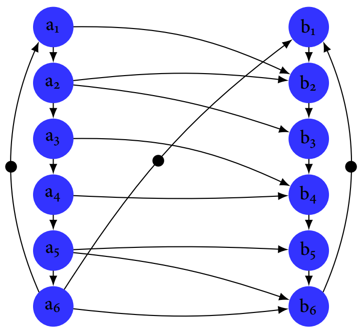
The above transformation can be understood from a *functional* perspective; there is a correspondence between the *data* that is processed by the firings of the actors of the two graphs. Suppose an actor firing in the MRSDF graph consumes values $x_1, \ldots x_n$, from some incoming channel, and produces $y_1, \ldots y_m$ onto some outgoing channel. In the single-rate equivalent, the HSDF actor corresponding to that firing consumes and produces the same values, be it that a different set of channels it involved. As a result of the transformation, the former and latter have the same number of *initial tokens*.

The principle behind the transformation from MRSDF into its single-rate equivalent, as depicted in Figure 4.1, differs from the principle that underlies the transformation illustrated in Figure 4.2(b). The latter figure shows the transformation of a CSDF graph into an HSDF graph, using the algorithm outlined in [11]. In the CSDF graph, the number of tokens produced and consumed by each actor varies cyclically; a single iteration of the graph consists of six firings of each actor.

The difference between this transformation and the previous one is illustrated by the fact that the number of tokens, produced and consumed by firings of the CSDF actors in Figure 4.2, does not correspond with the number of tokens consumed and



(a) CSDF graph.



(b) Equivalent HSDF graph.

Figure 4.2 – A CSDF graph and an equivalent HSDF graphs, derived by the transformation described in [11]. Here, equivalence can be understood from a *functional* perspective.

produced by the HSDF actors in the transformation. For example, the first firing of actor a produces a total of *three* tokens: two onto channel ab, and one onto its self-loop. In the HSDF graph, the corresponding actor, a_1, produces a total of *two* tokens. Still, the CSDF graph and its transformation are equivalent, in the sense that the times at which actors in the equivalent HSDF graph fire, can be mapped to the times at which the CSDF actors fire. Consequently, a schedule for the CSDF graph can be constructed by constructing a schedule for the equivalent HSDF graph.

The CSDF-to-HSDF transformation depicted in Figure 4.2 can be understood from a *dependency* perspective: channels represent *data dependencies* and thus impose precedence relations between the firings of actors. The channels in the HSDF graph should reflect these precedence relations. Thus, the transformation connects two HSDF actors v and w with a channel if actor w consumes at least one of the tokens produced by actor v; the precise *number* of tokens is irrelevant.

Note that, despite the difference in perspective of the two transformations outlined above, they have in common that the number of actors in the single-rate equivalent is equal to the number of firings that occur in a single graph iteration. This number may be large; the size of a single-rate equivalent grows exponentially in the size of the original graph, in the worst case.

Both transformations imply a different intuitive notion of equivalence, without defining it precisely. In this chapter we describe transformations that yield temporally equivalent graphs, in the system-theoretic perspective (see Definition 3.10) outlined in the previous chapter: the input-output relation of the graph and its transformation is identical. That is, when feeding both systems with the same input, their outputs are the same, and thus their temporal dynamics can not be distinguished from each other.

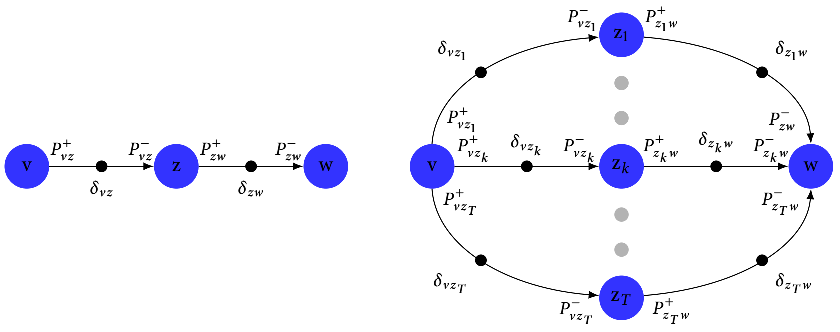
In this chapter we introduce a number of novel transformations. Section 4.1 describes the transformation of a CSDF graph into its *multi-rate equivalent*. The latter is an MRSDF graph that is temporally equivalent to the CSDF graph. This technique is generalised in Section 4.2, to arbitrary *unfoldings* of CSDF and MRSDF actors, where the transformation replaces each actor by a number of actors, each of which represents a subset of the firings of the original actor. For example, an MRSDF graph can be unfolded into a larger MRSDF graph, where actors in the latter distinguish between even and odd firings of actors in the former. This more general transformation subsumes the transformations of CSDF and MRSDF graphs into their single-rate equivalents. Each transformation is accompanied with a formal proof that the graph it produces is temporally equivalent to the graph it transforms.

4.1 Transforming CSDF into MRSDF

The novel approach that we take to transforming CSDF graphs into MRSDF graphs, is similar to the transformation from MRSDF into HSDF, where several firings of an MRSDF actor are each represented by a single HSDF actor, as described earlier. The HSDF graphs obtained by these existing transformations are named *single-rate equivalents*. We follow this naming, and refer to the MRSDF graphs constructed from a CSDF graph as *multi-rate equivalents*.

An intuitive first step into deriving a multi-rate equivalent from a CSDF graph is to let each CSDF actor be represented by as many MRSDF actors as it has phases. As we shall see later, the rates and tokens that must be assigned to channels in the multi-rate equivalent follow from this choice. The principle underlying the transformation from CSDF into MRSDF is that the actors and channels of the latter must, as a whole, impose the same set of functional constraints as their cyclo-static counterparts in the former do. Since these constraints are described in terms of predecessor functions, *rewriting* these functions leaves the set of imposed precedence constraints effectively unchanged. In Section 4.1.1, we outline how this process of rewriting allows us to derive the annotations for the channels in the MRSDF graph. Section 4.1.3 provides the formal proof that the CSDF graph and its multi-rate equivalent are indeed *temporally* equivalent, in the sense that their corresponding discrete event systems are undistinguishable.

The multi-rate equivalents obtained by the transformation given in Section 4.1.1 may consist of many channels, of which some may impose *redundant* constraints; omitting them does not change the graph's temporal dynamics. Section 4.1.4 describes how these channels are identified, and presents an algorithm that constructs a *minimal* multi-rate equivalent, which leaves out redundant channels.



(a) CSDF graph (b) Unfolded actor z into T actors

Figure 4.3 – A partial transformation of actor z.

4.1.1 Mapping actors and channels from CSDF to MRSDF

An intuitive relation between a CSDF graph and its multi-rate equivalent is to let actors in the latter represent individual *phases* of actors in the former. We follow this idea, and indicate the phase that is represented by these MRSDF actors in subscript; actor v in the CSDF graph is represented by actors $v_1, v_2, \ldots, v_{\varphi_v}$ in the MRSDF graph.

For the sake of argument, we restrict ourselves to the specific case that only a single CSDF actor is transformed into a set of MRSDF actors (see Figure 4.3). This means that all other CSDF actors and non-incident channels are maintained in the new graph. We refer to this specific transformation as a *partial transformation* from CSDF into MRSDF (see Figure 4.3). In order to reason about this transformation formally, we use the following definition:

Definition 4.1 (Partial transformation). *A* partial transformation *of a CSDF actor z in CSDF graph $\mathcal{G}$ yields a graph $\mathcal{H}$, where actor z is replaced by actors $z_1, \ldots, z_T$. Formally, we denote the transformation by $\mathcal{H} = \mathcal{E}^T_z(\mathcal{G})$, and define it by*

$$\mathcal{E}^T_z(\mathcal{G}) = \overline{\mathcal{G}_z} \cup \mathcal{G}^*_z,$$

where $\mathcal{G}_z$, with actors $\mathcal{A}_z$ and channels $\mathcal{C}_z$, is the graph induced by z and its (direct) successors and predecessors in $\mathcal{G}$, $\overline{\mathcal{G}_z} = \mathcal{G} - \mathcal{G}_z$, and $\mathcal{G}^*_z$ the graph with actors and channels respectively given by

$$\begin{aligned}
\mathcal{A}^*_z &= \{z_i | 1 \le i \le T\} \\
\mathcal{C}^*_z &= \{z_i w | zw \in \mathcal{C}_z, 1 \le i \le T\} \cup \{vz_i | vz \in \mathcal{C}_z, 1 \le i \le T\}
\end{aligned}$$

A partial transformation replaces a single actor by a set of actors. We refer to the size of this set as the *unfolding factor* of the transformed actor. Note that the partial transformation $\mathcal{H} = \mathcal{E}_z^T(\mathcal{G})$ only defines the *structure* of $\mathcal{H}$. It does not specify the annotations (rates, initial tokens and execution times) of channels and actors in $\mathcal{H}$. As MRSDF actor z_i in $\mathcal{H}$ represents phase i of actor z in $\mathcal{G}$, the execution time for actor z_i (which is independent of the firing index) is defined as

$$\tau_{z_i} = \tau_z(i),$$

and for all other actors in $\mathcal{H}$ the execution time is unchanged by the transformation.

When partially transforming a graph, the parameter T may be chosen arbitrarily. In the remainder of this section, we assume that the CSDF actor, z, is transformed into as many actors as it has phases. That is, we assume $\mathcal{H} = \mathcal{E}_z^{\varphi_z}(\mathcal{G})$.

The approach that we follow to derive annotations for channels in $\mathcal{H}$ is rooted in the fact that the completion of k firings of MRSDF actor z_i in $\mathcal{E}_z^{\varphi_z}(\mathcal{G})$ corresponds to the completion of $i+(k-1)\varphi_z$ firings[1] of CSDF actor z. This also means that a single firing of MRSDF actor z_i consumes $P_{vz}^{\Sigma-}$ tokens from each incoming channel vz_i, and produces $P_{zw}^{\Sigma+}$ tokens onto each outgoing channel z_iw. The production rates of said incoming channels is essentially left unchanged, i.e., the production rate of channel vz_i in $\mathcal{H}$ is equal to the production rate of channel vz in $\mathcal{G}$ (and analogously, consumption rates of outgoing channels follow). Guided by an intuitive translation between the predecessor functions of channels in $\mathcal{G}$ and $\mathcal{H}$, we may now derive the last unknown property, which is the number of initial tokens of the channels in the transformed graph.

For an incoming channel vz_i in $\mathcal{H}$, the associated predecessor function π_{vz_i} maps a number of consuming firings of z_i to the required number of producing firings of v. Making the correspondence between z_i and z explicit gives the following relation between π_{vz_i} and π_{vz}:

$$\pi_{vz_i}(k) = \pi_{vz}(i + (k-1)\varphi_z). \tag{4.1}$$

By expanding the terms $\pi_{vz_i}(k)$ and $\pi_{vz}(k)$ (see Definition 3.2, in particular (3.2)), we obtain the following equation:

$$\min\left\{m \,\middle|\, \sum_{l=1}^{m}\rho_{vz}^{+}(l) + \delta_{vz_i} - kP_{vz}^{\Sigma-} \geq 0\right\} = \min\left\{m \,\middle|\, \sum_{l=1}^{m}\rho_{vz}^{+}(l) + \delta_{vz} - (k-1)P_{vz}^{\Sigma-} \geq 0\right\}.$$

The initial tokens of channel vz_i are now found by solving this equation for δ_{vz_i}, which gives:

$$\delta_{vz_i} = \delta_{vz} + \sum_{l=i+1}^{\varphi_z} \rho_{vz}^{-}(l).$$

[1] The clumsy subtraction of one is necessary to let the first firing of z_1 correspond to the first firing of z.

To determine the annotations for the *outgoing* channels of the unfolded MRSDF actors, we take a slightly different approach: we first derive the relationship that should hold between predecessor functions of these channels and the original channel, zw, and then determine the annotations such that this relationship holds. Recall that, in the original graph, in order to enable the k^{th} firing of actor w, actor z must complete at least $\pi_{zw}(k)$ firings. In the transformed graph, enabling the k^{th} firing of actor w requires that each actor z_i has completed *at least* $\pi_{z_i w}(k)$ firings, which corresponds to $i + (\pi_{z_i w}(k) - 1)\varphi_z$ firings of z. To ensure that the earliest times at which w may fire are left invariant by the transformation, the following relations, between $\pi_{z_i w}$ and π_{zw}, must thus hold:

$$\pi_{zw}(k) = \max_i \{ i + (\pi_{z_i w}(k) - 1)\, \varphi_z \} \tag{4.2a}$$

$$\pi_{z_i w}(k) = \left\lceil \frac{\pi_{zw}(k) - i + 1}{\varphi_z} \right\rceil . \tag{4.2b}$$

The equality of (4.2a) and (3.2) (see Section 3.1) now allows us to solve for $\delta_{z_i w}$, which gives

$$\delta_{z_i w} = \delta_{zw} + \sum_{l=1}^{i-1} \rho^{+}_{zw} .$$

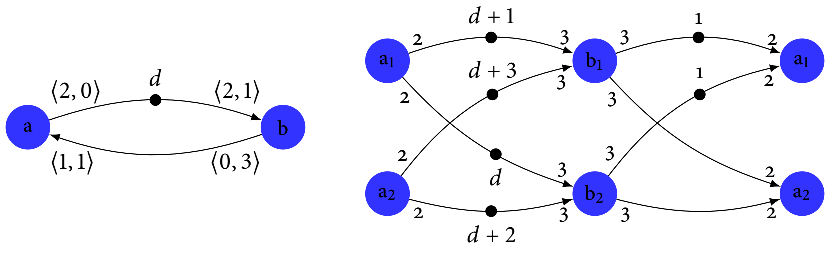
This completes the partial transformation; channels not incident to actor z simply have their annotations unchanged. Note that the partial transformation allows us to transform a CSDF graph into an MRSDF graph, simply by applying the partial transformation to every actor in the CSDF graph. Algorithm 1 combines this sequence of partial transformations into a single transformation. As an example, Figure 4.4 depicts a CSDF graph and its equivalent MRSDF graph, obtained by applying Algorithm 1.

The density of the equivalent MRSDF graph is higher than the density of the CSDF graph, as a single CSDF channel vw is transformed into $\varphi_v \varphi_w$ channels. In Section 4.1.4, we demonstrate how the number of channels in the equivalent MRSDF graph may be reduced.

4.1.2 Temporal equivalence

Using the transformation presented in the previous section, we can transform one system into another. In this view, a CSDF graph represents a *component* that connects inputs and outputs (i.e., source and sink actors). The transformation replaces a CSDF component with an MRSDF component. This new system must be temporally equivalent to the old system. That is, the system constructed around the multi-rate equivalent should be such that, in terms of temporal dynamics, the two systems are indistinguishable. We now turn to proving that this is indeed the case.

In the following, we consider again the partial transformation, of a single CSDF actor into multiple MRSDF actors. That is, we let $\mathcal{G}$ be a CSDF graph, and $\mathcal{H}$ =



(a) CSDF graph.

(b) Equivalent MRSDF graph.

Figure 4.4 – A CSDF graph and its equivalent MRSDF graph. The equivalent MRSDF graph has been unrolled such that all channels point from left to right, and implicit self-loops have been omitted for the sake of clarity.

Algorithm 1: Transforms a CSDF graph into an MRSDF graph.

```
1  input  : A CSDF graph $\mathcal{G}$.
2  output : An MRSDF graph.
3  $\mathcal{G}_{MR} \longleftarrow$ empty graph
4  foreach actor $v$ in $\mathcal{G}$ do
5      for $i = 1$ to $\varphi_v$ do
6          Add actor $v_i$ to $\mathcal{G}_{MR}$
7          $\tau_{v_i} \longleftarrow \tau_v(i)$
8  foreach channel $vw$ in $\mathcal{G}$ do
9      for $j = 1$ to $\varphi_w$ do
10         for $i = 1$ to $\varphi_v$ do
11             Add channel $v_i w_j$ to $\mathcal{G}_{MR}$
12             $\rho^+_{v_i w_j} \longleftarrow P^{\Sigma+}_{vw}$
13             $\rho^-_{v_i w_j} \longleftarrow P^{\Sigma-}_{vw}$
14             $\delta_{v_i w_j} \longleftarrow \delta_{vw} + \sum_{l=1}^{i-1} \rho^+_{vw} + \sum_{l=j+1}^{\varphi_w} \rho^-_{vw}$
15 return $\mathcal{G}_{MR}$
```

$\mathcal{E}_z^{\varphi_z}(\mathcal{G})$ its partial transformation. We furthermore consider a single input and a single output, connected to actor z in graph $\mathcal{G}$. Let the input u be connected to z in $\mathcal{G}$, and z_i in $\mathcal{H}$. Sink actor y_1 is the output of $\mathcal{G}$, and y_2 forms the output of $\mathcal{H}$.

For $\mathcal{G}$, the relation between the output of the system and its input is fully specified by

$$\begin{aligned} y_1(k) &= t_z\left(\pi_{zy}(k)\right), \\ t_z(k) &= t_v\left(\pi_{vz}(k)\right) \otimes \tau_z(k) \oplus t_v\left(k-1\right) \oplus u\left(\pi_{uz}(k)\right), \end{aligned} \tag{4.3}$$

and the relation between the input and output of $\mathcal{H}$ is defined by:

$$\begin{aligned} y_2(k) &= \bigoplus_{i=1}^{\varphi_z} t_{z_i}\left(\pi_{z_i y}(k)\right), \\ t_{z_i}(k) &= \bigoplus_{uz_i \in \mathcal{H}} u\left(\pi_{uz_i}(k)\right) \otimes \tau_{z_i}(k) \oplus t_{z_i}(k-1). \end{aligned} \tag{4.4}$$

In order to prove that $\mathcal{G}$ and $\mathcal{H}$ are temporally equivalent, we first rewrite the output, y_2, of $\mathcal{H}$. The following lemma expresses the output of $\mathcal{H}$ in terms of the predecessor functions associated with channels uz and zy in $\mathcal{G}$.

Lemma 4.1 (Output of the multi-rate equivalent). *Let $m(k)$ and $r(k)$ be functions, such that $m(k)\varphi_z + r(k) = \pi_{zy}(k)$ The output of the MRSDF system can be written in terms $\pi_{uz}(k)$, $r(k)$ and $m(k)$, as follows:*

$$y_2(k) = \bigoplus_{i=r(k)-\varphi_z+1}^{r(k)} u\left(\pi_{uz}\left(m(k)\varphi_z + i\right)\right) \otimes \tau_z(i).$$

Proof. Using (4.1), (4.2a) and (4.2b), the predecessor function associated with a path $uz_i y$ can be written in terms of the predecessor function π_{uz}, and functions $m(k)$ and $r(k)$:

$$\begin{aligned} \pi_{uz_i}\left(\pi_{z_i y}(k)\right) &\overset{(4.2b)}{=} \pi_{uz_i}\left(1 + \left\lfloor \frac{\pi_{zy}(k) - i}{\varphi_z} \right\rfloor\right) \\ &= \pi_{uz_i}\left(m(k) + 1 + \left\lfloor \frac{r(k) - i}{\varphi_z} \right\rfloor\right) \\ &\overset{(4.1)}{=} \pi_{uz}\left(m(k)\varphi_z + r(k) - (r(k) - i) \bmod \varphi_z\right). \end{aligned} \tag{4.5}$$

The output of the MRSDF system is given by:

$$y_2(k) = \bigoplus_{i=1}^{\varphi_z} t_{z_i}(\pi_{z_i y}) = \bigoplus_{j \in \mathbb{N}} \bigoplus_{i=1}^{\varphi_z} u\left(\pi_{uz_i}\left(\pi_{z_i y}(k) - j\right)\right) \otimes \tau_z(i).$$

By assumption, the input sequence u is non-decreasing. Furthermore, since the predecessor function is non-decreasing, the composition of u and π_{uz_i} is non-decreasing as well. The summation over j may thus be eliminated from the above expression, giving:

$$\begin{aligned} y_2(k) &= \bigoplus_{i=1}^{\varphi_z} u\left(\pi_{uz_i}\left(\pi_{z_i y}(k)\right)\right) \otimes \tau_z(i) \\ &\overset{(4.5)}{=} \bigoplus_{i=1}^{\varphi_z} u\left(\pi_{uz}\left(r(k) + m(k)\varphi_z - (r(k) - i) \bmod \varphi_z\right)\right) \otimes \tau_z(i). \end{aligned}$$

Next, we split this into *two* summations, which eliminates the modulo term (note that $\tau_z(i) = \tau_z(i - \varphi_z)$):

$$\begin{aligned} y_2(k) = &\bigoplus_{i=1}^{r(k)} u\left(\pi_{uz}\left(m(k)\varphi_z + i\right)\right) \otimes \tau_z(i) \\ &\oplus \bigoplus_{i=r(k)+1}^{\varphi_z} u\left(\pi_{uz}\left(m(k)\varphi_z + i - \varphi_z\right)\right) \otimes \tau_z(i - \varphi_z). \end{aligned}$$

The lemma is obtained by a substitution of $i - \varphi_z$ for i in the rightmost summation, followed by a rearrangement of terms. □

The equivalence of the two systems, associated with $\mathcal{G}$ and $\mathcal{H}$, now follows from the equality of their outputs, y_1 and y_2, as formulated in the following lemma.

Lemma 4.2 (System equality). *The systems given by* (4.3) *and* (4.4) *are temporally equivalent. That is, for all possible input sequences u, and for all $k \in \mathbb{Z}$, we have $y_1(k) = y_2(k)$.*

Proof. For the CSDF graph we expand the input-output recurrence relation given by (4.4), into:

$$y_1(k) = \bigoplus_{j \in \mathbb{N}} u\left(\pi_{uz}\left(m(k)\varphi_z + r(k) - j\right)\right) \otimes \tau_z\left(r(k) - j\right).$$

Since both the input sequence $(u(k))_{k \in \mathbb{Z}}$ and the predecessor function is non-decreasing, their composition is non-decreasing as well. Furthermore, $\tau_z(i) = \tau_z(i - \varphi_z)$. We therefore may limit the domain of variable j in the above expression, in the following way:

$$y_1(k) = \bigoplus_{j=0}^{\varphi_z - 1} u\left(\pi_{uz}\left(m(k)\varphi_z + r(k) - j\right)\right) \otimes \tau_z\left(r(k) - j\right).$$

Substituting $r(k) - j$ for j in the above expression for $y_1(k)$ gives an expression that, by Lemma 4.1, satisfies $y_1(k) = y_2(k)$, which completes the proof. □

This concludes the proof for the temporal equivalence of graph $\mathcal{G}$, and its partial transformation, $\mathcal{H}$. The multi-rate equivalent of $\mathcal{G}$ may be obtained by repeatedly applying the partial transformation to its CSDF actors, until they have all been replaced by a number of MRSDF actors. Since every partial transformation yields a graph that is temporally equivalent, the multi-rate equivalent is temporally equivalent as well. This is formally stated by the following theorem.

Theorem 4.1 (Temporal equivalence). *Let $\mathcal{G}$ be a CSDF graph, and $\mathcal{H}$ be the MRSDF graph obtained by applying Algorithm 1 to $\mathcal{G}$. Graphs $\mathcal{G}$ and $\mathcal{H}$ are temporally equivalent.*

Proof. By Lemma 4.2, any partial transformation, of a single actor in $\mathcal{G}$, to a number of MRSDF actors, yields an equivalent system. Graph $\mathcal{G}$ can be regarded as a *composition* of systems. Since any composition of equivalent systems is again equivalent [22], the theorem holds true. The stated theorem now follows by induction on the structure of $\mathcal{G}$. □

4.1.3 Mapping admissible schedules

The transformation from a CSDF graph into an MRSDF graph, as described in Algorithm 1, provides a mapping between the firings of actors in the former and firings of actors in the latter: if v is an actor in the CSDF graph, and v_i is a corresponding actor in the multi-rate equivalent, then firings of v_i map to firings $i, i + \varphi_v, \ldots$ of v. Since, by Theorem 4.1, the MRSDF graph is temporally equivalent to the CSDF graph, the mapping between firings also provides a mapping between the schedules of the two graphs. In order to reason about this relation, we first prove the following relation between the predecessor functions of a channel in a CSDF graph, and corresponding channels in its multi-rate equivalent.

Proposition 4.1 (Predecessor function relation). *Let $\mathcal{G}$ be a CSDF graph, and $\mathcal{H}$ the MRSDF graph obtained by applying Algorithm 1. Furthermore, let vw be a channel in $\mathcal{G}$ and v_iw_j a channel in $\mathcal{H}$ that corresponds to vw. The following equivalence relation holds:*

$$\pi_{v_iw_j}(m) \geq k \Leftrightarrow \pi_{vw}(j + (m-1)\varphi_w) \geq i + (k-1)\varphi_v. \tag{4.6}$$

Proof. Using equations (4.1) and (4.2a), function $\pi_{v_iw_j}$ can be rewritten in the following way:

$$\pi_{v_iw_j}(m) = \pi_{v_iw}(j + (m-1)\varphi_w) = 1 + \left\lfloor \frac{\pi_{vw}(j + (m-1)\varphi_w) - i}{\varphi_v} \right\rfloor.$$

The proposition now readily follows from (A.4) (see Appendix A). □

Every schedule that is admissible for the CSDF graph can be mapped to a schedule for the MRSDF graph, through the mapping implied by the transformation.

Such a mapped schedule is admissible for the MRSDF graph. The following lemma provides a proof for this statement.

Lemma 4.3 (Multi-rate schedules). *Let $\mathcal{G}$ be a CSDF graph, and $\mathcal{H}$ the MRSDF graph obtained by applying Algorithm 1. Furthermore, let t be an admissible schedule for $\mathcal{G}$, and let s be a schedule for $\mathcal{H}$, obtained by the following mapping:*

$$s_{v_i}(k) = t_v(i + (k-1)\varphi_v), \tag{4.7}$$

for every actor v_i in $\mathcal{H}$. Then schedule s is admissible for $\mathcal{H}$.

Proof. Admissibility of t gives:

$$k \geq m \Rightarrow t_v(k) \geq t_v(m), \tag{4.8a}$$

$$\pi_{vw}(k) \geq m \Rightarrow t_w(k) \geq t_v(m) \otimes \tau_w(k). \tag{4.8b}$$

In particular, we have:

$$\begin{aligned} k \geq m &\Rightarrow t_v(i + (k-1)\varphi_v) \geq t_v(i + (m-1)\varphi_v) \\ &\Rightarrow s_{v_i}(k) \geq s_{v_i}(m), \end{aligned} \tag{4.9}$$

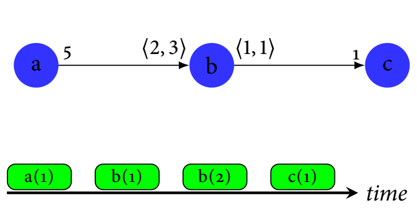
and

$$\begin{gathered} \pi_{vw}(j + (m-1)\varphi_w) \geq i + (k-1)\varphi_v \Rightarrow s_{w_j}(m) \geq s_{v_i}(k) \otimes \tau_v(i) \\ \Leftrightarrow \\ \pi_{v_i w_j}(m) \geq k \Rightarrow s_{w_j}(m) \geq s_{v_i} \otimes \tau_v(i). \end{gathered} \tag{4.10}$$

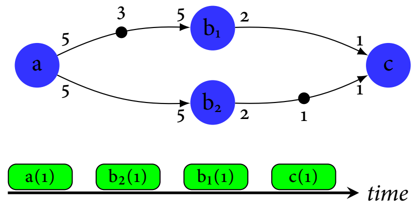
Together, the implied relations (4.9) and (4.10) state that s is admissible, which completes the proof. □

The mapping between firings of a CSDF graph and its multi-rate equivalent give rise to a mapping, from schedules of the latter, to schedules of the former. This mapping maps firings of *multiple* MRSDF actors onto firings of a *single* CSDF actor. Since consecutive phases of the same CSDF actor are represented by different actors MRSDF graph, care must be taken that functional determinacy is not invalidated when mapping an admissible schedule from the MRSDF graph to the CSDF graph, especially when actors in the latter can fire auto-concurrently. To see this, consider Figure 4.5, which depicts a situation in which a single firing of actor a enables a single period of b. Functional determinacy ensures that in the CSDF graph, the first phase of b cannot complete later than its second phase. In the transformed MRSDF graph, however, no such ordering is imposed on actors b_1 and b_2, and the first phase may perfectly well complete later than the second phase.

The mapping defined by the following lemma ensures that any schedule that is admissible for the multi-rate equivalent, is mapped to a schedule that is admissible for the CSDF graph.



(a) CSDF graph and an admissible schedule.



(b) The multi-rate equivalent, with an admissible schedule.

Figure 4.5 – When mapping an admissible schedule for the equivalent MRSDF graph to the original CSDF graph, care must be taken that functional determinacy is not invalidated.

Lemma 4.4 (Cyclo-static schedules). *Let $\mathcal{G}$ be a CSDF graph, and $\mathcal{H}$ the MRSDF graph obtained by applying Algorithm 1. The following relation maps every schedule s that is admissible for $\mathcal{H}$ to an admissible schedule t for $\mathcal{G}$:*

$$t_v(k) = \bigoplus_{i=1}^{\varphi_v} s_{v_i}\left(\left\lfloor \frac{k-i}{\varphi_v} \right\rfloor + 1\right), \tag{4.11}$$

where v is an actor in $\mathcal{G}$, and v_i are corresponding actors in $\mathcal{H}$.

Proof. Let s be an admissible schedule for $\mathcal{H}$, and t be the schedule obtained from s by (4.11). The first implication of the fact that s is admissible is:

$$k \geq m \Rightarrow s_{v_i}(k) \geq s_{v_i}(m),$$

which holds for $1 \leq i \leq \varphi_v$. This directly implies the first necessary condition for the admissibility of t:

$$k \geq m \Rightarrow \bigoplus_{i=1}^{\varphi_v} s_{v_i}(k) \geq \bigoplus_{i=1}^{\varphi_v} s_{v_i}(m),$$

The second implication of the admissibility of s relates to the constraints imposed by the channels of $\mathcal{H}$. This is stated by:

$$\pi_{v_i w_j}(m) \geq k \Rightarrow s_{w_j}(m) \geq s_{v_i}(k) \otimes \tau_w(j).$$

Predecessor function $\pi_{v_i w_j}(m)$ is non-increasing in j (see Proposition 4.1). This means that the above premise implies $\pi_{v_n w_j}(m) \geq k$ for $1 \leq n \leq i$. Furthermore, the same premise implies $\pi_{v_n w_j}(m) \geq k - 1$ for $i < n \leq \varphi_v$. The above implication can thus be replaced by the following, more elaborate, implication:

$$\pi_{v_i w_j}(m) \geq k \Rightarrow s_{w_j}(m) \geq \bigoplus_{l=1}^{i} s_{v_l}(k) \otimes \tau_w(l) \oplus \bigoplus_{l=i+1}^{\varphi_v} s_{v_l}(k-1) \otimes \tau_w(j). \tag{4.12}$$

We rewrite the second (and last) necessary constraint for the admissibility of t, using Proposition 4.1 and (4.11), into the following formulation:

$$\pi_{v_i w_j}(m) \geq k \Rightarrow \bigoplus_{l=1}^{\varphi_w} s_{w_l}\left(\left\lfloor \frac{j-l}{\varphi_w} \right\rfloor + m\right) \geq \bigoplus_{l=1}^{\varphi_v} s_{v_l}\left(\left\lfloor \frac{i-l}{\varphi_v} \right\rfloor + k\right) \otimes \tau_w(j).$$

This is equivalent to the following implication:

$$\pi_{v_i w_j}(m) \geq k \Rightarrow \bigoplus_{l=1}^{\varphi_w} s_{w_l}\left(\left\lfloor \frac{j-l}{\varphi_w} \right\rfloor + m\right) \geq \left(\bigoplus_{l=1}^{i} s_{v_l}(k) \oplus \bigoplus_{l=i+1}^{\varphi_v} s_{v_l}(k-1)\right) \otimes \tau_w(j),$$

which holds, since, by (4.12):

$$\bigoplus_{l=1}^{\varphi_w} s_{w_l}\left(\left\lfloor \frac{j-l}{\varphi_w} \right\rfloor + m\right) \geq s_{w_j}(m) \geq \left(\bigoplus_{l=1}^{i} s_{v_l}(k) \oplus \bigoplus_{l=i+1}^{\varphi_v} s_{v_l}(k-1)\right) \otimes \tau_w(j).$$

Thus, schedule t is admissible for $\mathcal{G}$. □

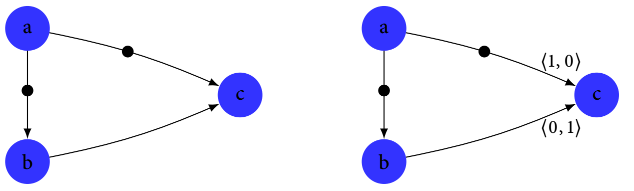
This completes the mapping between the admissible schedules of a CSDF graph and its multi-rate equivalent. A result of Theorem 4.1 and the above two lemmas, is that any (temporal) analysis technique that is applicable to MRSDF graphs, may be applied to CSDF graphs, by applying the analysis to its multi-rate equivalent. Analysis results obtained for the equivalent MRSDF graphs can thus be translated back to the CSDF graph through the mapping defined by Lemma 4.4.

4.1.4 Pruning

The admissible schedules of an SDF graph are defined in terms of the predecessor functions of the graph's channels (see Definition 3.4). Some channels may impose constraints that are not *binding*, i.e., removing the channel has no effect on the admissible schedules of the graph. Pruning these channels simplifies the graph. In particular, it reduces the number of cycles that occur in the graph. As a result, the graph may no longer be strongly, or even weakly, connected. In the following sections, we discuss different types of pruning, and their effects on the graph's admissible schedules.

Pruning non-binding Channels

A channel imposes a precedence constraint on the times at which its consuming actor may fire. In some cases, this constraint is non-binding; omitting the constraint does not relax the constraints on the consumer's admissible firing times. An example of such a case is depicted in Figure 4.6(a), where, in any admissible schedule, path abc imposes a stronger constraint on firings of actor c than channel ac does. As a result, the latter may be pruned from the graph, without changing the set of admissible schedules. The annotations of the channels matter; consider for example Figure 4.6(b), which differs from Figure 4.6(a) only by the consumption rates of



(a) HSDF: ac is non-binding

(b) CSDF: ab and ac are both binding

Figure 4.6 – Non-binding channels are identified from triangles in the graph.

the two incoming channels of c. The result of these consumption rate vectors is that c alternately depends on a and b, and hence none of the channels may be pruned.

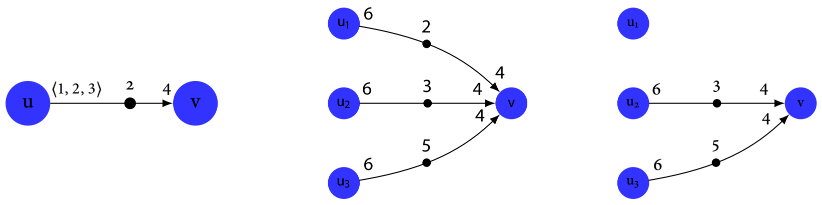
In some cases, channels are only non-binding if we restrict the admissible schedules, e.g. by enforcing extra constraints. As an example, consider the MRSDF graph shown in Figure 4.7. Here, if we consider the subset of admissible schedules in which each actor fires as soon as it is enabled (i.e., self-timed schedules), then channel u_1v_1 never imposes any extra delay on actor u_1, as the k^{th} firing of u_2 never precedes the k^{th} firing of u_1. This restricted notion of a non-binding channel is captured by the following definition:

Definition 4.2 (Non-binding channel). *Let $\mathcal{G}$ be an SDF graph, and T a set of schedules of $\mathcal{G}$. A channel vw in $\mathcal{G}$ is* non-binding *in T, if the precedence constraints that are imposed on the channel's consumer, by some other channel, are at least as strong. That is, channel vw is non-binding in T if there exists an incident channel zw, such that for all $t \in T$:*

$$\forall k \in \mathbb{Z} : t_z(\pi_{zw}(k)) \geq t_v(\pi_{vw}(k)).$$

Graphs that are obtained from the CSDF-to-MRSDF transformation (see the previous section) contain potentially many non-binding channels, if we consider only those schedules that are obtained, through the mapping given by Lemma 4.3, from the schedules of the original CSDF graph. This restricted set of schedules allows one to analyse the schedules of a CSDF graph via the schedules of its multi-rate equivalent.

In the remainder of this section, we assume that CSDF actors either have a self-loop, or a non-varying execution time. This means that functional determinacy is guaranteed, as firings that start later than other firings, never complete earlier. Consequently, the identification of non-binding channels is based solely on the constraints they impose through the availability of tokens, as expressed through their associated predecessor function.



(a) CSDF channel.

(b) Equivalent MRSDF graph.

(c) Pruned.

Figure 4.7 – Channels in (parts of) the equivalent MRSDF graph may be pruned if they are non-binding in a specific subset of the graph's admissible schedules.

The channels that we prune from the multi-rate equivalent are those channels that are created from the same CSDF channel, i.e., in lines 11 – 14 of Algorithm 1. These incident channels have similar properties: since they are derived from the same CSDF channel, they have the same production and consumption rates, but their number of initial tokens differs. The following lemma describes how the non-bindingness of a channel in the multi-rate equivalent may be determined from its predecessor functions.

Lemma 4.5. *Let $\mathcal{G}$ be a CSDF graph, and $\mathcal{H} = \mathcal{E}_v^T(\mathcal{G})$ be the partially transformed CSDF graph. Furthermore, let $v_i w$ be a channel in $\mathcal{H}$, and S be the set of schedules obtained by mapping each schedule in $\mathcal{S}(\mathcal{G})$ to $\mathcal{H}$, using the mapping given by* (4.7)*. Channel $v_i w$ is non-binding in S if the following holds for all k:*

$$\bigoplus_{k\in\mathbb{Z}} \{\pi_{v_i w}(k) - \pi_{v_{i+1} w}(k)\} \le 0, \tag{4.13}$$

Proof. Channel $v_i w$ is non-binding if the maximum of the terms $s_{v_{i+1}}(\pi_{v_{i+1}w}(k))$ and $s_{v_i}(\pi_{v_i w}(k))$ is attained by the former, for all $s \in S$ and for all $k \in \mathbb{Z}$. We rewrite this maximum of firing times in s to a maximum of firing times in t, in the following way:

$$\begin{aligned} & s_{v_i}\left(\pi_{v_i w}(k)\right) \oplus s_{v_{i+1}}\left(\pi_{v_{i+1} w}(k)\right) \\ \overset{(4.7)}{=} \quad & t_v\left(i + (\pi_{v_i w}(k) - 1)\, T\right) \oplus t_v\left(i + 1 + (\pi_{v_{i+1} w}(k) - 1)\, T\right) \\ \overset{(3.5)}{=} \quad & t_v\left(i + (\pi_{v_i w}(k) - 1)\, T \oplus i + 1 + (\pi_{v_{i+1} w}(k) - 1)\, T\right). \end{aligned}$$

Given the premise, that is, $\pi_{v_i w}(k) \le \pi_{v_{i+1} w}(k)$ for all $k \in \mathbb{Z}$, the last expression reduces to:

$$t_v\left(i + 1 + (\pi_{v_{i+1} w}(k) - 1)\, T\right) \overset{(4.7)}{=} s_{v_{i+1}}\left(\pi_{v_{i+1} w}(k)\right).$$

Thus, the maximum of the terms $s_{v_{i+1}}(\pi_{v_{i+1}w}(k))$ and $s_{v_i}(\pi_{v_iw}(k))$ is attained by the former, for all $k \in \mathbb{Z}$. By Definition 4.2, channel v_i is thus non-binding. □

Using the steps outlined in the previous section, together with Lemma 4.5, we may transform a CSDF graph into an equivalent MRSDF graph and prune its non-binding channels. A more efficient approach avoids creating these non-binding channels in the first place, rather than pruning them in a later step. If we were to use Lemma 4.5 to determine which channels in the multi-rate equivalent are non-binding, then the pruning phase would have quite some overhead; every channel that is created by the transformation needs to be "checked" for non-bindingness, and this check is relatively expensive. Rather than *determining* whether a channel is non-binding, the following theorem characterises those channels that are *not* (known to be) non-binding. The theorem uses identity (A.9) (see Appendix A).

Theorem 4.2. *Let $\mathcal{G}$ be a CSDF graph, and $\mathcal{H}$ be the partial transformation of $\mathcal{G}$, given by $\mathcal{H} = \mathcal{E}_v^{\varphi_v}(\mathcal{G})$. Let vw be a channel in $\mathcal{G}$, and v_iw a corresponding channel in $\mathcal{H}$. Channel v_iw is not non-binding if, for some $n < \varphi_w$, the following inequality is satisfied:*

$$\left\lfloor \frac{\Delta_{vw}(i,n)}{\gcd\left(P_{vw}^{\Sigma-}, P_{vw}^{\Sigma+}\right)} \right\rfloor > \left\lfloor \frac{\Delta_{vw}(i-1,n)}{\gcd\left(P_{vw}^{\Sigma-}, P_{vw}^{\Sigma+}\right)} \right\rfloor,$$

Proof. By Lemma 4.5 and (3.3), channel v_iw is non-binding if

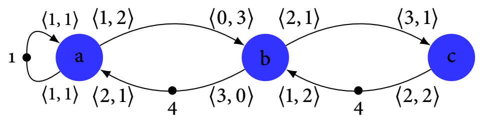
$$\max_{n<\varphi_w} \max_{k\in\mathbb{Z}} \left\{ \left\lceil \frac{kP_{vw}^{\Sigma-} - \Delta_{vw}(i-1,n)}{P_{vw}^{\Sigma+}} \right\rceil - \left\lceil \frac{kP_{vw}^{\Sigma-} - \Delta_{vw}(i,n)}{P_{vw}^{\Sigma+}} \right\rceil \right\} \le 0.$$

this can be simplified using (A.1a) into:

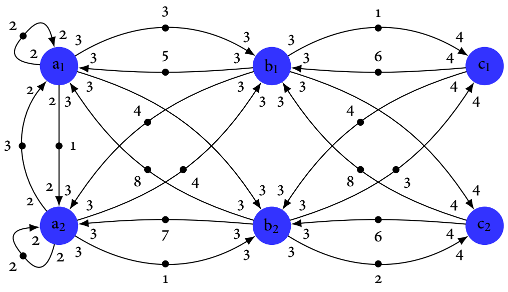
$$\max_{n<\varphi_w} \left\{ \left\lfloor \frac{\Delta_{vw}(i,n)}{\gcd\left(P_{vw}^{\Sigma-}, P_{vw}^{\Sigma+}\right)} \right\rfloor - \left\lfloor \frac{\Delta_{vw}(i-1,n)}{\gcd\left(P_{vw}^{\Sigma-}, P_{vw}^{\Sigma+}\right)} \right\rfloor \right\} \le 0. \tag{4.14}$$

The negation of this gives the inequality stated by the theorem. □

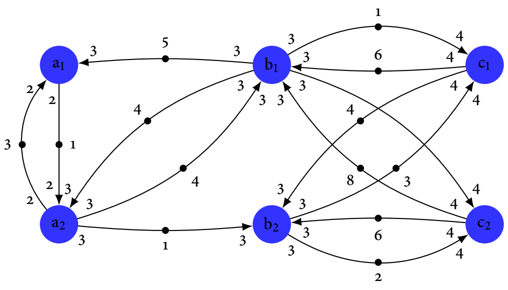
Theorem 4.2 leads to a revised version of Algorithm 1, listed as Algorithm 2. The algorithm transforms a CSDF graph into an MRSDF graph that has as many actors as the CSDF actors have phases. The number of edges in the multi-rate equivalent constructed by Algorithm 2 is, in the worst case, equal to the number of edges in the graph constructed by Algorithm 1. This is illustrated in Figure 4.8, which depicts an example CSDF graph along with the two MRSDF graphs obtained by Algorithm 1 and Algorithm 2. As the figure shows, the latter prunes 6 of the 20 channels of the former.



(a) CSDF graph



(b) Equivalent MRSDF graph, obtained by Algorithm 1



(c) Pruned MRSDF graph, obtained by Algorithm 2

Figure 4.8 – A CSDF graph, its equivalent MRSDF graph obtained by applying Algorithm 1, and its pruned multi-rate equivalent, obtained by applying Algorithm 2 to the CSDF graph.

Algorithm 2: Transforms a CSDF graph into a pruned and temporally equivalent MRSDF graph.

```
input  : A CSDF graph $\mathcal{G}$.
output : A temporally equivalent MRSDF graph with $\sum_v \varphi_v$ actors.
$\mathcal{G}_{MR} \longleftarrow$ empty graph
foreach actor $v$ in $\mathcal{G}$ do
    for $i = 1$ to $\varphi_v$ do
        Add actor $v_i$ to $\mathcal{G}_{MR}$
        $\tau_{v_i} \longleftarrow \tau_v(i)$
foreach channel $vw$ in $\mathcal{G}$ do
    for $j = 1$ to $\varphi_w$ do
        for $i = 1$ to $\varphi_v$ do
            $c_{\min} \longleftarrow \left\lfloor \frac{\Delta_{vw}(i,j)+P^{\Sigma-}_{vw}}{g_{vw}} \right\rfloor$
            $c_{\max} \longleftarrow \left\lceil \frac{\Delta_{vw}(i-1,j)+P^{\Sigma-}_{vw}}{g_{vw}} \right\rceil$
            if $c_{\min} > c_{\max}$ then
                Add channel $v_i w_j$ to $\mathcal{G}'$
                $\rho^+_{v_i w_j} \longleftarrow P^{\Sigma+}_{vw}$
                $\rho^-_{v_i w_j} \longleftarrow P^{\Sigma-}_{vw}$
                $\delta_{v_i w_j} \longleftarrow \delta_{vw} + \sum_{l=1}^{i-1} \rho^+_{vw}(l) + \sum_{l=j+1}^{\varphi_w} \rho^-_{vw}(l)$
return $\mathcal{G}_{MR}$
```

Pruning non-critical actors

Due to the pruning of channels from a graph, the graph may no longer be strongly connected. In particular, some actors may no longer lie on a cycle. Since the maximum rate at which actors may fire in any feasible schedule is determined by the cycles in the graph, actors that do not lie on a cycle do not impact the throughput of the graph. When the sole purpose of transforming a CSDF graph into its multi-rate equivalent is to analyse the resulting graph for its throughput, these non-critical actors may thus be pruned from the graph.

Pruning non-critical actors may decrease the size of the graph significantly, whilst leaving its throughput invariant. Although in a graph's multi-rate equivalent typically a few non-critical actors may be pruned, the number of non-critical actors is often higher for *single-rate equivalents*. Construction of single-rate equivalents is described in more detail in Section 4.3. Chapter 6 describes how the pruning of non-critical actors from single-rate equivalents leads to an efficient method for computing a graph's throughput.

4.2 Unfolding CSDF actors

Transforming a CSDF graph into its multi-rate equivalent, as presented in the previous sections, replaces each CSDF actor with as many MRSDF actors as it has phases. The correspondence between the MRSDF and CSDF actors is natural; MRSDF actor v_i corresponds to firings $i, i + \varphi_v, \ldots$ of CSDF actor v. If we consider the firing times of CSDF actor v as a (discrete-time) *signal*, then each of the corresponding MRSDF actors can be regarded as *downsampled* (or *decimated*) signals. In this view, the transformation from CSDF to MRSDF or HSDF can be seen as downsampling, where the sampling period for an actor is equal to its number of phases. There is, however, no restriction on this sampling period. In this section, we show how an actor may be *unfolded* an arbitrary number of times. In case an actor the unfolding factor is not an integer multiple of the actor's period, the result will not be a set of MRSDF actors, but CSDF actors with periods at most equal to the original actor's period. This is described in the following section, which follows an approach that is similar to the one followed in Section 4.1. We describe pruning of arbitrarily unfolded CSDF graphs in Section 4.2.2. A final result of the transformations presented in this section is the transformation of a CSDF graph into its *single-rate equivalent*, which differs from the transformation of [60, 88] and [11].

4.2.1 Mapping channels and actors

To determine the production and consumption rates of the outgoing and incoming channels of the actors in an unfolded graph, we follow an approach similar to the one followed in the previous section. We consider first the partial transformation (see Definition 4.1) $\mathcal{H}$ of CSDF graph $\mathcal{G}$, but now choose an arbitrary unfolding factor. That is, we let $\mathcal{H} = \mathcal{E}_z^T(\mathcal{G})$.

An actor z_i in $\mathcal{H}$ represents firings $\{i + mT | m \in \mathbb{N}\}$. If we choose T to be an integer multiple of the number of phases of z, then the behaviour (i.e., execution times and production and consumption rates) of z_i does not vary; the behaviour of z_i is defined by phase $i \bmod_1 \varphi_z$ of actor z. If T is not an integer multiple of φ_z, then the behaviour of z_i varies cyclically. The period of this cyclically varying sequence is determined by the greatest common divisor of T and φ_z, and is given by:

$$\varphi_{z_i} = \frac{\varphi_z}{\gcd(T, \varphi_z)}, \tag{4.15}$$

for all $i \in \{1, \ldots, T\}$.

Given the relation between actors in $\mathcal{H}$ and those in $\mathcal{G}$, we now turn to deriving the annotations of the channels in $\mathcal{H}$. In graph $\mathcal{H}$, the number of tokens on a channel $z_i w$, after k producing firings, is equal to the number of tokens on CSDF channel zw after $i - 1 + kT$ producing firings. On an incoming channel vz_i, the number of tokens after k consuming firings is equal to the number of tokens after $i + (k-1)T$ firings of actor z. In other words, we have the following relations between the balance functions in $\mathcal{G}$ and $\mathcal{H}$:

$$\Delta_{z_i w}(m,k) = \Delta_{zw}(i-1+mT,k) \tag{4.16a}$$
$$\Delta_{vz_i}(m,k) = \Delta_{vz}(m,i+(k-1)T). \tag{4.16b}$$

The token balance function $\Delta_{vw}(i,j)$ gives the number of tokens on a channel vw after i producing and j consuming firings. We may thus obtain the number of initial tokens of channels in $\mathcal{H}$ from (4.16a) and (4.16b), using $\Delta(0,0)$:

$$\delta_{z_i w} = \Delta_{z_i w}(0,0) = \delta_{zw} + \sum_{l=1}^{i-1} \rho^+_{zw}(l) \tag{4.17a}$$
$$\delta_{vz_i} = \Delta_{vz_i}(0,0) = \delta_{vz} + \sum_{l=1}^{T-i} \rho^-_{vz}(1-l). \tag{4.17b}$$

Finally, the production and consumption rate vectors for outgoing and incoming channels of the actors z_i in $\mathcal{H}$ can now be obtained by solving equations (4.16a) and (4.16b) for periodic functions $\rho^+_{z_i w}$ and $\rho^-_{vz_i}$. Expanding the difference of the token balance functions $\Delta_{z_i w}(m,k)$ and $\Delta_{zw}(i-1+mT,k)$ gives, after cancelling and rearranging terms:

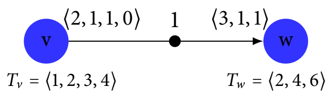
$$\begin{aligned}\sum_{l=1}^{m} \rho^+_{z_i w}(l) &= \sum_{l=1}^{mT} \rho^+_{zw}(i-1+l)\\ &= \sum_{l=1}^{m}\sum_{n=1}^{T} \rho^+_{zw}(i-1+n+(l-1)T).\end{aligned}$$

In a similar way, by expanding $\Delta_{vz_i}(m,k) - \Delta_{vz}(m,i+(k-1)T)$, we obtain the sequence of consumption rates associated with incoming channels of actors z_i in $\mathcal{H}$:

$$\begin{aligned}\sum_{l=1}^{k} \rho^-_{vz_i}(l) &= \sum_{l=1-T}^{0} \rho^-_{vz}(i+l) + \sum_{l=1}^{(k-1)T} \rho^-_{vz}(i+l)\\ &= \sum_{l=1}^{k}\sum_{n=1}^{T} \rho^-_{vz}(i+n+(l-2)T).\end{aligned}$$

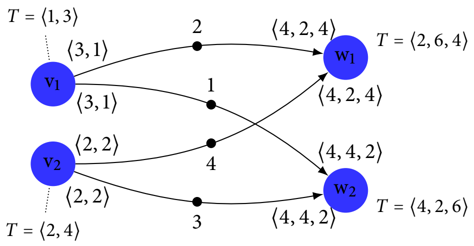
The production and consumption rate vectors now readily follow:

$$\rho^+_{z_i w}(l) = \sum_{n=1}^{T_z} \rho^+_{zw}(i-1+n+(l-1)T) \tag{4.18a}$$
$$\rho^-_{vz_i}(l) = \sum_{n=1}^{T_z} \rho^-_{vz}(i+n+(l-2)T). \tag{4.18b}$$



⟨2, 1, 1, 0⟩
1
⟨3, 1, 1⟩
v
w
$T_v = \langle 1, 2, 3, 4\rangle$
$T_w = \langle 2, 4, 6\rangle$

(a) CSDF channel.



$T = \langle 1, 3\rangle$
⟨3, 1⟩
v1
⟨3, 1⟩
⟨2, 2⟩
v2
⟨2, 2⟩
$T = \langle 2, 4\rangle$
2
1
4
3
⟨4, 2, 4⟩
w1
$T = \langle 2, 6, 4\rangle$
⟨4, 2, 4⟩
⟨4, 4, 2⟩
w2
$T = \langle 4, 2, 6\rangle$
⟨4, 4, 2⟩

(b) Equivalent CSDF graph.

Figure 4.9 – An example CSDF channel, and an equivalent graph in which both actors are unfolded twice.

Given the number of phases of actors z_i by (4.15), the sums of the production and consumption rate vectors associated with incoming and outgoing channels of z_i are the following:

$$P^{\Sigma+}_{z_i w} = P^{\Sigma+}_{zw} \frac{T}{\gcd(\varphi_z, T)}, \tag{4.19a}$$

$$P^{\Sigma-}_{v z_i} = P^{\Sigma-}_{vz} \frac{T}{\gcd(\varphi_z, T)}. \tag{4.19b}$$

Note that, the sums of the rate vectors associated with channels of vz_i in $\mathcal{H}$ are all the same. This is illustrated in Figure 4.9, which depicts an example of unfolding an actor using the procedure outlined in this section.

Theorem 4.3 (Temporal equivalence of unfoldings). *Let $\mathcal{G}$ be a CSDF graph, and $\mathcal{H}$ be the CSDF graph obtained by applying Algorithm 3 to $\mathcal{G}$. Then graphs $\mathcal{G}$ and $\mathcal{H}$ are temporally equivalent.*

Proof. Consider the partial transformation of $\mathcal{G}$, $\mathcal{G}^* = \mathcal{E}^T_z(\mathcal{G})$. A straightforward generalisation of Lemma 4.2, where φ_z is replaced by T, gives that the two graphs $\mathcal{G}$ and $\mathcal{G}^*$ are temporally equivalent. The stated theorem now follows, analogous to Theorem 4.1, by induction on the structure of $\mathcal{G}$. □

4.2.2 Pruning the unfolded graph

In the same spirit as we did earlier for multi-rate equivalents of CSDF graphs, we may prune non-binding channels from the graph obtained by applying Algorithm 3. Although the approach is similar, it is slightly more involved than the procedure outlined in Algorithm 1. Rather than pruning channels from the unfolded graph,

Algorithm 3: Transforms a CSDF graph into another CSDF graph by unfolding actors an arbitrary number of times.

```
input  : A CSDF graph $\mathcal{G}$, and vector $T$ where $T_v$ gives the number of firings of actor
         $v$ that are to be unfolded.
output : A CSDF graph with $\sum_v (T_v \varphi_v)$ actors.
$\mathcal{H} \longleftarrow$ empty graph
foreach actor $v$ in $\mathcal{G}$ do
    for $i = 1$ to $T_v$ do
        Add actor $v_i$ to $\mathcal{H}$ with execution time $\tau_v(i)$
foreach channel $vw$ in $\mathcal{G}$ do
    for $j = 1$ to $T_w$ do
        for $i = 1$ to $T_v$ do
            for $k = 1$ to $\frac{\varphi_w}{\gcd(T_w, \varphi_w)}$ do
                $\rho^+(k) \longleftarrow \sum_{n=1}^{T_v} \rho^+_{vw}(i - 1 + n + (k-1)T_v)$
            for $k = 1$ to $\frac{\varphi_v}{\gcd(T_v, \varphi_v)}$ do
                $\rho^-(k) \longleftarrow \sum_{n=1}^{T_w} \rho^-_{vw}(j + n + (k-2)T_w)$
            $\delta \longleftarrow \delta_{vw} + \sum_{l=1}^{i-1} \rho^+_{vw}(l) + \sum_{l=1}^{T_w - j} \rho^-_{vw}(\varphi_w - l + 1)$
            Add channel $v_i w_j$, with production rates $\rho^+$, consumption rates $\rho^-$ and
            tokens $\delta$ to $\mathcal{H}$
return $\mathcal{H}$
```

we first describe which channels may be pruned in case each actor is unfolded an integer multiple of its period. That is, we consider the graph obtained by applying Algorithm 3, where the unfolding factor T_v for actor v satisfies $T_v = m_v \varphi_v$, with m_v a non-zero positive integer. The following lemma is a generalisation of Theorem 4.2.

Lemma 4.6. *Let $\mathcal{G}$ be a CSDF graph, and $\mathcal{H}$ be the partial transformation of $\mathcal{G}$, given by $\mathcal{H} = \mathcal{E}_v^{m\varphi_v}(\mathcal{G})$. That is, an integer number of periods of CSDF actor v is unfolded. Let vw be a channel in $\mathcal{G}$, and $v_i w$ a channel in $\mathcal{H}$. Channel $v_i w$ is not non-binding if i satisfies the congruence relation*

$$i \equiv k \left(\operatorname{mod}_1 \frac{g_{v_i w} \varphi_v}{g_{vw}} \right),$$

where k is constrained by:

$$k \in \left\{ j + cx\varphi_v \middle| \left\lceil \frac{-\Delta_{vw}(j,n)}{g_{vw}} \right\rceil \leq c < \left\lceil \frac{-\Delta_{vw}(j-1,n)}{g_{vw}} \right\rceil, 0 \leq n < \varphi_w, 1 \leq j \leq \varphi_v \right\}.$$

Proof. By Lemma 4.5 and (3.3), channel $v_{i+k\varphi_v} w$ is non-binding if

$$\max_{n < \varphi_w} \max_{l \in \mathbb{Z}} \left\{ \left\lceil \frac{l P_{vw}^{\Sigma-} - k P_{vw}^{\Sigma+} - \Delta_{vw}(i-1,n)}{m P_{vw}^{\Sigma+}} \right\rceil - \left\lceil \frac{l P_{vw}^{\Sigma-} - k P_{vw}^{\Sigma+} - \Delta_{vw}(i,n)}{m P_{vw}^{\Sigma+}} \right\rceil \right\} \leq 0.$$

this can be simplified using (A.1a) into:

$$\max_{n<\varphi_w}\left\{\left\lfloor\frac{kP^{\Sigma+}_{vw}+\Delta_{vw}(i,n)}{\gcd\left(P^{\Sigma-}_{vw},mP^{\Sigma+}_{vw}\right)}\right\rfloor-\left\lfloor\frac{kP^{\Sigma+}_{vw}+\Delta_{vw}(i-1,n)}{\gcd\left(P^{\Sigma-}_{vw},mP^{\Sigma+}_{vw}\right)}\right\rfloor\right\}\leq 0. \tag{4.20}$$

The negation of this says that channel $v_{i+k\varphi_v}w$ is *not* non-binding, if there exists $n<\varphi_w$ such that

$$\left\lfloor\frac{kP^{\Sigma+}_{vw}+\Delta_{vw}(i-1,n)+\rho^{+}_{vw}(i)}{\gcd\left(P^{\Sigma-}_{vw},mP^{\Sigma+}_{vw}\right)}\right\rfloor>\left\lfloor\frac{kP^{\Sigma+}_{vw}+\Delta_{vw}(i-1,n)}{\gcd\left(P^{\Sigma-}_{vw},mP^{\Sigma+}_{vw}\right)}\right\rfloor, \tag{4.21}$$

which holds for those values of k that satisfy the set of congruence relations

$$\left(kP^{\Sigma+}_{vw}+\Delta_{vw}(i-1,n)+\rho^{+}_{vw}(i)\right)\in\{0,\dots,\rho^{+}_{vw}(i)-1\}\pmod{g_m}, \tag{4.22}$$

where $g_m=\gcd\left(P^{\Sigma-}_{vw},mP^{\Sigma+}_{vw}\right)$. Since the congruence relation $ax\equiv b\pmod m$ has a solution x if and only if b is a multiple of the greatest common divisor of a and m, the solutions k that adhere to (4.22) are given by (note that $\gcd\left(g_m,P^{\Sigma+}_{vw}\right)=g_{vw}$)

$$k\frac{P^{\Sigma+}_{vw}}{g_{vw}}\in\left\{\left\lceil\frac{-\Delta_{vw}(i,n)}{g_{vw}}\right\rceil,\dots,\left\lfloor\frac{-1-\Delta_{vw}(i-1,n)}{g_{vw}}\right\rfloor\right\}\left(\bmod\frac{g_m}{g_{vw}}\right).$$

Multiplying both sides of the equation with the multiplicative inverse of $\frac{P^{\Sigma+}_{vw}}{g_{vw}}$ modulo $\frac{g_m}{g_{vw}}$ yields the solutions for k. The theorem follows from substitution of these solutions into $i+k\varphi_v$. □

Lemma 4.6 does not directly allow one to prune graphs that are obtained by unfolding an actor an *arbitrary* number of times. In order to obtain a general procedure for pruning non-binding channels, we formulate it in terms of a pruning procedure for multi-rate equivalents, as provided by Lemma 4.6. To describe this general procedure, we introduce three graphs: a CSDF graph, $\mathcal{G}$, its partial transformation, $\mathcal{H}=\mathcal{E}^T_z(\mathcal{G})$, and the (pruned) multi-rate equivalent of $\mathcal{H}$, obtained using Algorithm 1, which we denote $\mathcal{M}$. As usual, we denote the actors in $\mathcal{H}$ that correspond to actor z in $\mathcal{G}$ by z_i (where $1\leq i\leq T$). Corresponding actors in $\mathcal{M}$ have a double subscript; actor z_{i_k} in $\mathcal{M}$ corresponds to the k^{th} phase of actor z_i in $\mathcal{H}$. This means that an actor z_{i_k} in $\mathcal{M}$ corresponds to phase $(i+(k-1)T)\bmod_1\varphi_z$ of actor z in $\mathcal{G}$.

Now, the presence or absence of certain channels in $\mathcal{M}$ reflects the fact whether the corresponding channels in $\mathcal{H}$ are binding or not, in the following way. A channel $z_{i_k}w$ in $\mathcal{M}$ corresponds to the dependency between phases $(i+(k-1)T)\bmod_1\varphi_v$ of actor z and phases of actor v. If the channel was pruned by Algorithm 1, then this would imply that the dependency between these phases is not binding.

Theorem 4.4 (Pruning arbitrary unfoldings). *Let $\mathcal{G}$ be a CSDF graph, $\mathcal{H}=\mathcal{E}^T_z(\mathcal{G})$ be the partial transformation of $\mathcal{G}$, where actor z is unfolded T times. Furthermore,* *let $\mathcal{M}$ be the multi-rate equivalent of $\mathcal{H}$, obtained by applying Algorithm 1 to $\mathcal{H}$. Let zw be a channel in $\mathcal{G}$, and $z_i w$ a channel in $\mathcal{H}$.*

Channel $z_i w$ is non-binding, if $z_{i+kT} w_m$ is non-binding in $\mathcal{M}$, for all $k, m \in \mathbb{Z}$ such that $i + kT \leq \frac{T\varphi_z}{\gcd(T,\varphi_z)}$ and $1 \leq m \leq \varphi_w$.

Proof. This is a straightforward generalisation of the proof of Lemma 4.6. □

Theorem 4.4 gives rise to Algorithm 4, which allows one to choose arbitrary unfolding factors for actors, and construct a pruned and temporally equivalent CSDF graph.

4.3 Single-rate equivalents

The transformation outlined in the previous section serves as a first step in the transformation of a consistent CSDF graph into a temporally equivalent HSDF graph. Let $\mathcal{G}$ be a consistent CSDF graph, with repetition vector q, and let $\mathcal{H}$ be the graph, obtained from $\mathcal{G}$, by unfolding each actor v as many times as it fires in a single iteration, i.e., q_v, times. As a result, the production and consumption rate, associated with each channel in $\mathcal{H}$, are now the same (this follows from the balance equations, see Section 2.1.3). Consequently, every actor in $\mathcal{H}$ has a repetition vector entry of one. Thus, each actor fires at an equal rate, which implies that the temporal dynamics of the graph matches that of an HSDF graph. Technically, though, the graph is not an HSDF graph, as this requires, by the usual definition, that the rates associated with each channel are one. However, we may use a simple transformation (see also [8, 65]) to transform the graph into an HSDF graph. This transformation is related to the following identity that holds for the predecessor function of an MRSDF channel vw:

$$\pi_{vw}(k) = \left\lceil \frac{k\rho^-_{vw} - \delta_{vw}}{\rho^+_{vw}} \right\rceil \overset{(A.6)}{=} \left\lceil \frac{\frac{k\rho^-_{vw}}{g_{vw}} + \left\lceil \frac{-\delta_{vw}}{g_{vw}} \right\rceil}{\frac{\rho^+_{vw}}{g_{vw}}} \right\rceil \overset{(A.5)}{=} \left\lceil \frac{\frac{k\rho^-_{vw}}{g_{vw}} - \left\lfloor \frac{\delta_{vw}}{g_{vw}} \right\rfloor}{\frac{\rho^+_{vw}}{g_{vw}}} \right\rceil$$

The above identity can be interpreted as follows: for any MRSDF channel vw, we may divide the rates and initial tokens by the greatest common divisor of the channel's rates (rounding the quotients down to the next integer), without changing the predecessor function (and thus the graph's temporal dynamics). Since, for each channel in $\mathcal{H}$, the production rate equals the consumption rate, applying this transformation results in a graph in which all rates are one. In other words, the resulting graph is an HSDF graph. We refer to this graph as the *single-rate equivalent* of $\mathcal{G}$.

The single-rate equivalent of a CSDF graph may thus be obtained by unfolding every actor as many times as it fires in a single graph iteration, followed by a scaling and rounding of rates and tokens, as described above. Due to the pruning of non-binding channels, each actor w_j in the unfolded graph has one incoming channel for

Algorithm 4: Transforms a CSDF graph into an equivalent and pruned CSDF graph by unfolding actors an arbitrary number of times.

```
1  input  : A CSDF graph $\mathcal{G}$, and vector $T$ where $T_v$ gives the number of firings of actor
            $v$ that are to be unfolded.
2  output : A CSDF graph with $\sum_v T_v$ actors.
3  $\mathcal{H} \longleftarrow$ empty graph
4  foreach actor $v$ in $\mathcal{G}$ do
5      for $i = 1$ to $T_v$ do
6          Add actor $v_i$ to $\mathcal{H}$
7          $\tau_{v_i} \longleftarrow \tau_v(i)$
8          $\varphi_{v_i} \longleftarrow \frac{\varphi_v}{\gcd(T_v,\varphi_v)}$
9  foreach channel $vw$ in $\mathcal{G}$ do
10     $g_1 \longleftarrow \gcd\left(P^{\Sigma+}_{vw}, \frac{T_w}{\gcd(T_w,\varphi_w)} P^{\Sigma-}_{vw}\right)$
11     $g_2 \longleftarrow \gcd\left(\frac{T_v}{\gcd(T_v,\varphi_v)} P^{\Sigma+}_{vw}, \frac{T_w}{\gcd(T_w,\varphi_w)} P^{\Sigma-}_{vw}\right)$
12     $x \longleftarrow$ solution to: $P^{\Sigma+}_{vw} x \equiv g_1 \pmod{\frac{g_2}{g_1}}$
13     for $j = 1$ to $T_v$ do
14         $S \longleftarrow$ empty set
15         for $n = 1$ to $\varphi_{w_j}$ do
16             for $l = 1$ to $\varphi_v$ do
17                 $c_{\min} \longleftarrow \left\lceil \frac{-\Delta_{vw}(l, j+(n-1)T_w)}{g_1} \right\rceil$
18                 $c_{\max} \longleftarrow \left\lceil \frac{-\Delta_{vw}(l-1, j+(n-1)T_w)}{g_1} \right\rceil$
19                 for $c = c_{\min}$ to $c_{\max} - 1$ do
20                     $S \longleftarrow S \cup \left\{(l + cx\varphi_v) \bmod_1 \gcd\left(T_v, \varphi_v \frac{g_2}{g_1}\right)\right\}$
21         foreach $r$ in $S$ do
22             foreach $i = r$ to $T_v$ step $\gcd\left(T_v, \varphi_v \frac{g_2}{g_1}\right)$ do
23                 Add channel $v_i w_j$ to $\mathcal{H}$
24                 $\rho^+_{v_i w_j} \longleftarrow \frac{T_v}{\gcd(T_v,\varphi_v)} P^{\Sigma+}_{vw}$
25                 $\rho^-_{v_i w_j} \longleftarrow \frac{T_w}{\gcd(T_w,\varphi_w)} P^{\Sigma-}_{vw}$
26                 $\delta_{v_i w_j} \longleftarrow \delta_{vw} + \sum_{l=1}^{i-1} \rho^+_{vw}(l) + \sum_{l=j+1}^{m_w \varphi_w} \rho^-_{vw}(l)$
27 return $\mathcal{H}$
```

each incoming channel vw of actor w in the CSDF graph. The source actor for this channel is given by the predecessor function associated with vw: the channel in the unfolded graph connects v_i with w_j, with $i = \pi_{vw}(j) \bmod_1 q_v$. The production and consumption rates associated with channel $v_i w_j$ are equal to the number of tokens produced onto vw, during a single graph iteration. In the following, we denote this number by N_{vw}. The number of tokens, on a channel $v_i w_j$ in the unfolded graph, is given by line 26 of Algorithm 4, and may be rewritten into the following:

$$\begin{aligned}\delta_{v_{\pi_{vw}(j) \bmod_1 q_v} w_j} &= \Delta_{vw}\left(\pi_{vw}(j) \bmod_1 q_v - 1, j\right) + N_{vw} \\ &\overset{(A.3)}{=} \Delta_{vw}\left(\pi_{vw}(j) - 1, j\right) - N_{vw}\left(\left\lfloor \frac{\pi_{vw}(j) - 1}{q_v} \right\rfloor - 1\right).\end{aligned}$$

Note that in the above, the expression $\Delta_{vw}(\ldots)$ satisfies:

$$-N_{vw} \leq \Delta_{vw}\left(\pi_{vw}(j) - 1, j\right) < 0.$$

Dividing the initial number of tokens of channel $v_i w_j$ by the greatest common divisor of the production and consumption rate of $v_i w_j$, which is N_{vw}, rounding the result down to the nearest integer, gives:

$$\left\lfloor \frac{\delta_{v_{\pi_{vw}(j)} w_j}}{N_{vw}} \right\rfloor = -\left\lfloor \frac{\pi_{vw}(j) - 1}{q_v} \right\rfloor \overset{(A.5)}{=} \left\lceil \frac{1 - \pi_{vw}(j)}{q_v} \right\rceil \overset{(A.2)}{=} \left\lfloor \frac{q_v - \pi_{vw}(j)}{q_v} \right\rfloor.$$

The transformation of a CSDF graph into a graph in which each actor is unfolded as many times as it fires in a single graph iteration, followed by the scaling (and rounding) of rates and tokens described above, is combined into Algorithm 5, listed below. This algorithm is a generalisation of the algorithm, presented in [11], to construct the single-rate equivalent of a CSDF graph.

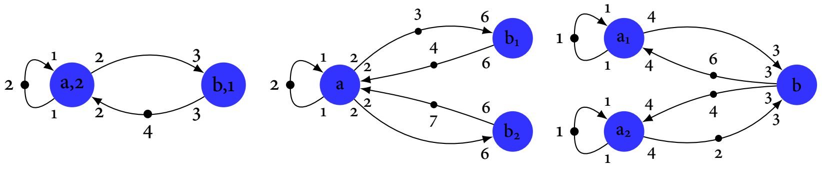
4.4 Unfolding MRSDF graphs

Every MRSDF graph is a CSDF graph, and thus the transformations presented earlier may be applied to MRSDF graphs as well. This is illustrated by Figure 4.10, which depicts an MRSDF graph that is *partially unfolded*. Two transformations are applied: the unfolding of actor a, respectively b, by a factor of two. Although the transformation gives a graph that is temporally equivalent to the original graph, the throughput of the graph may change as a result of the transformation. This is due to the fact that the unfolding may cause the graph's repetition vector to change: a single iteration of the graph obtained when unfolding actor a (Figure 4.10(c)) consists of *ten* firings. In particular, the repetition vector entry of actor b has changed from two in the original graph, to four in the unfolded graph. Consequently, as the length of an iteration has doubled, the throughput of the graph is halved.

One may view the unfolding of MRSDF graphs as a two-step process: in the first step, we replace each scalar (production and consumption) rate by a vector that repeats the rate as many times as the corresponding actor is to be unfolded. This

Algorithm 5: Transforms a CSDF graph into a pruned and temporally equivalent HSDF graph.

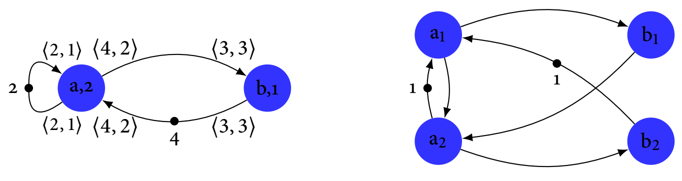
```
1  input  : A CSDF graph $\mathcal{G}$.
2  output : A temporally equivalent HSDF graph.
3  $\mathcal{H} \longleftarrow$ empty graph
4  foreach actor $v$ in $\mathcal{G}$ do
5      for $i = 1$ to $q_v$ do
6          Add actor $v_i$ to $\mathcal{H}$
7          $\tau_{v_i} \longleftarrow \tau_v(i)$
8  foreach channel $vw$ in $\mathcal{G}$ do
9      for $j = 1$ to $q_w$ do
10         $i \longleftarrow \pi_{vw}(j) \bmod_1 q_v$
11         Add channel $v_i w_j$ to $\mathcal{H}$
12         $\delta_{v_i w_j} \longleftarrow \left\lfloor \frac{q_v - \pi_{vw}(j)}{q_v} \right\rfloor$
13 return $\mathcal{H}$
```



(a) MRSDF graph. (b) Unfolded actor b. (c) Unfolded actor a.

Figure 4.10 – The unfolding transformation, applied to different actors in an MRSDF graph.

first step transforms the MRSDF graph into a CSDF graph, which may then, as a second step, be transformed into its multi-rate equivalent. We may adapt this transformation slightly, by *grouping* several firings together in the first step. This is analogous to enforcing that certain firings are executed in parallel. For example, consider the cycle aba, in the MRSDF graph depicted in Figure 4.10(a), in isolation (that is, ignore the restriction imposed by the self-loop). Actor a fires three times in a graph iteration, the first two of these three can run in parallel. We model this (see Figure 4.11(a)) by replacing the scalar rates associated with the actor with a vector of length two. The first entry in this vector reflects the number of tokens that are produced or consumed by *two* parallel executions, the second entry is equal to the original rate. Actor b can't start more than one firings in parallel, as the number of tokens in the cycle is insufficient for that (this requires that the cycle has at least *six* tokens). We model this by a vector of length two, matching the repetition vector



(a) CSDF graph. (b) Equivalent HSDF graph.

Figure 4.11 – Grouping together the first two, out of every three, firings of actor a in Figure 4.10(a), by transforming the graph into an CSDF graph. This forces firings to be executed in parallel, without changing the graph's temporal dynamics.

entry of actor b.

The throughput of the resulting CSDF graph may be analysed from its single-rate equivalent, given in Figure 4.11(b). Maximum cycle ratio of the latter is attained by cycle $a_1 b_1 a_2 b_2 a_1$, and equals six. The throughput of the graph is thus one sixth, which is equal to the throughput of the CSDF graph. In other words, the grouping of firings is feasible, i.e., it has not changed the graph's throughput.

The grouping of specific firings, modelling their (forced) parallel execution, has been presented earlier in [80], in the context of *scalable* synchronous dataflow. The authors apply a technique, which is similar to the grouping of firings outlined in this section, to MRSDF graphs, but restrict it to replacing rates with scalar integer multiples, rather than vectors. This scaling of rates is called *vectorization* in [80], and is used to decrease the switching overhead of task activations by increasing the extent of vector processing.

Grouping several firings of an actor together restricts the scheduling freedom for that actor. As a result, schedules that are admissible for the MRSDF graph may not be admissible for the CSDF graph. The difficulty in obtaining an equivalent CSDF graph with this transformation lies in choosing which firings to group together. Making this choice requires more information than can be obtained from the structure of the graph. It requires knowledge of the graph's schedule. In Chapter 6 we show how this parallelism can be inferred in an iterative fashion, and how identifying this parallelism plays a crucial role in composing the throughput of connected MRSDF cycles.

4.5 Discussion

This chapter builds upon the mathematical characterisation of SDF graphs presented in the previous chapter. The transformations presented in this chapter allow one to replace one SDF graph by one that is temporally equivalent. This means that the (temporal) behaviour of the graph, as observed from its input and output, is the same.

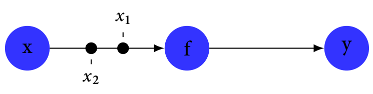
Graphs are transformed by rewriting the predecessor functions associated with its channels. This view on constructing equivalent HSDF or MRSDF graphs differs fundamentally from existing SDF graph transformations. In particular, Algorithm 1, which transforms a CSDF graph into its multi-rate equivalent, is the first such transformation. While the known transformation of CSDF into its lumped representation (see [45, 75] and Section 2.3.1) gives an MRSDF graph that is *functionally*, but not temporally equivalent, the transformation outlined in Section 4.1 is proven to yield a temporally equivalent graph.

The difference between transformations that exist in literature and those presented in this chapter is further emphasised by the fact that those presented in this thesis allow initial tokens to be *parameterised*. In particular, whereas the current convention in literature is that the initial token distribution affects the *structure* of the graph's single-rate equivalent, our transformations allow the single-rate equivalent to be constructed even if the number of tokens is specified by a parameter. This allows optimisation techniques (such as the minimisation of initial tokens under a throughput constraint) that work for HSDF graphs to be applied to CSDF graphs. Another (but perhaps less useful) property of the transformation is that it may be applied to graphs that are *not* consistent.

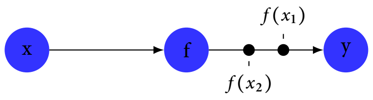
The transformations presented in this chapter differ from the transformations of MRSDF into HSDF, as described in [60, 88], and also from the transformation of CSDF into HSDF described in [11]. As the main goal of all our presented transformations is the construction of temporally equivalent graphs, we have ignored the *functional* aspect of SDF graphs. A formal treatment of the effect of the transformation on these functional aspects is beyond the scope of this thesis. Instead, we argue that the transformation leaves the functional behaviour intact, if the transformation includes simple adaptations of the way in which data is consumed and produced. We motivate this proposition with an illustrative example. For this, consider the graphs that are depicted in Figure 4.12.

The graphs in figures 4.12(a) and 4.12(b) show the processing of input tokens x_1 and x_2, by actor f. The actor applies a function f to the data represented by the token. The dataflow process associated with actor f thus maps an input stream $x_1, \dots x_n$ to an output stream $f(x_1), \dots f(x_n)$. If we apply Algorithm 3 to unfold actor f into two actors, we obtain the graphs shown in figures 4.12(c) and 4.12(d). Here, actors f_1 and f_2 represent odd and even firings of actor f. The transformation assigns a token to channel xf_1, and also one to channel f_2y. To distinguish these added tokens from tokens that represent data, they are coloured red in Figure 4.12(c).

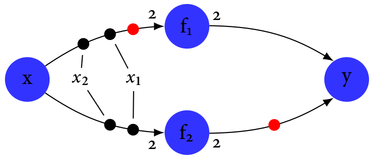
In the unfolded graph, a single firing of actor f_1 or f_2 consumes and produces *two* tokens, instead of one. The first firing of f_1 first consumes a red token, followed by the token that represents x_1. Actor f_1 may thus discard the first of these two tokens, processing only x_1, and producing $f(x_1)$. The actor must produce two tokens, so



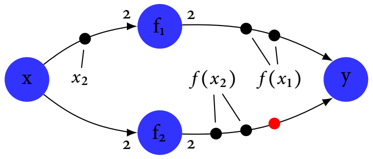
(a) After two firings of x.



(b) After two firings of f.



(c) After two firings of x.



(d) After one firing of f_1 and f_2.

Figure 4.12 – When unfolding an actor, the processing of input and output data must be adapted as well. Empty tokens (represented by *red* dots) are used to ensure that the processing of data is analogous to that of the original graph.

we let the actor produce $f(x_1)$ twice. We apply the same reasoning to the first firing of actor f_2: we let the actor discard the first of its two consumed tokens, such that it processes only x_2, producing $f(x_2)$ twice. As a result of the firings of f_1 and f_2 (see Figure 4.12(d)), two firings of actor y are enabled. The first firing reads a red token and the token representing f_1. If we let consecutive firings of y discard one of these two tokens, in an alternating fashion, then y processes the sequence of tokens $f(x_1), \ldots, f(x_n)$. That is, the sequence of tokens processed by y in the unfolded graph is equivalent to the sequence processed by y in the original graph.

The above example shows how, with a simple adaptation of the processing of data, the functional behaviour of the unfolded graph may be aligned with that of the original graph. This adaptation is simple, and is characterised by two rules: first, unfolded actors discard a prefix of the token sequence they consume, and duplicate their output. Second, actors that consume data from unfolded actors do so in an alternating fashion. In this view, the tokens assigned to unfolded channels (coloured red in the example figures), by the unfolding transformation, serve to shift the input and output token sequences in such a way that these two rules yield functionally equivalent behaviour.

Although an example is not a formal proof, the above provides evidence that given this slight adaptation of the reading and writing of data, the graphs produced by the (non-pruning) transformation algorithms listed in this chapter are, in fact, both temporally and functionally equivalent. That is, the dataflow process network associated with the transformed graph is the same as the one associated with the original graph. Again, note that this functional view on the transformations differs from the one put forward by Lee in [60]. Whereas in the transformation of Lee (see also Section 2.3.1) the produced data is *distributed* over outgoing channels, in our view (see Figure 4.12) data is *copied* onto each outgoing channel.

The purpose of the transformations presented in this chapter is to reveal dependencies between firings in greater detail. In this sense, the transformation is *exact*: all dependencies that are imposed by channels in the original graph, are also imposed by the channels in the transformed graph, and vice versa. The next chapter presents another kind of transformation, which has a different purpose. It is *approximate* rather than exact: the graphs it yields are temporal *abstractions*, rather than *equivalents*, of the original graph. Together, these exact and approximate transformations form the ingredients of an incremental approach to throughput analysis, which we present in Chapter 6.

5 Single-rate approximations

Abstract – *An exact analysis of a non-homogeneous synchronous dataflow graph suffers from the length of a single graph iteration, which is exponential in the size of the graph. Because the typical aim of dataflow analysis is to provide guarantees with respect to performance, a conservative estimation of the graph's performance is often sufficient. In this chapter, we describe how such an estimation may be obtained, and how its error may be assessed. The approximations that we derive are homogeneous synchronous dataflow graphs, which may be analysed efficiently using existing techniques.*

Synchronous dataflow graphs find their application in the modelling and analysis of real-time systems. The typical purpose of analyses is to provide the systems designer with *guarantees* regarding the system's performance. For example, an analysis of the real-time system that controls the engine thrust of a rocket may reveal whether the engine is guaranteed to respond, in time, to its control system. These guarantees can be given if the system's worst-case behaviour is known. Analysing the worst-case behaviour may be computationally expensive; the analysis of a CSDF graph has a worst-case complexity that is exponential in the size of the graph [71, 77]. This justifies the use of analyses that are *conservative* with respect to the "exact" worst-case performance; if the braking system of a car is *guaranteed* to respond *within* five milliseconds, then we may safely dimension the rest of the system under the assumption that the response time is precisely five milliseconds.

The cost of using conservative performance estimates to design a system, is that the system will be *over-dimensioned*: buffers used for communication links are larger than necessary, and processing elements are configured to run at higher frequencies than required. To minimise these costs, an assessment of the extent

Large parts of this chapter have been published in [RdG:3] and [RdG:2].

to which the estimates are conservative is highly desirable. This can be achieved by computing both a conservative upper bound and an *optimistic* lower bound on the worst-case behaviour; the tightness of the conservative estimate is never worse than the *difference* between the two bounds.

Abstraction of a model such that its analysis becomes cheaper, at the expense of a loss in accuracy, is common in various fields where mathematical models serve to model real-world phenomena. In this chapter, we introduce such an abstraction for CSDF graphs: we show how a CSDF graph can be transformed into an HSDF graph that is either *pessimistic* (that is, conservative), or *optimistic* with respect to the CSDF graph. This transformation comes at the expense of a loss in accuracy: the temporal constraints on the firings of actors are, depending on the kind of approximation, relaxed or tightened with respect to the constraints imposed by the approximated graph.

The approximation involves the computation of *linear bounds* on the predecessor functions (see Chapter 3) associated with actors and channels. We give approximations for both the *actor firing* and the *token transfer* perspective, and compare their quality in terms of their estimation error.

The conservative approximations are *temporal abstractions* of the original graph: when feeding input to a system at a low enough rate, their temporal behaviour is indistinguishable from that of the approximated graph. At higher input rates, however, the system's lower throughput becomes visible. Its performance is never better than that of the original graph. We highlight a particular application of such a temporal abstraction: any of its admissible schedules can be mapped to one that is admissible for the original graph. This provides a computationally cheaper approach to computing admissible schedules for CSDF graph, at the expense of their not attaining maximum throughput.

We start this chapter with a brief discussion on a class of discrete event systems that are relatively easy to analyse: those that are linear and shift-invariant. This sets a goal for the approximating transformations: they must yield linear and shift-invariant systems. The transformation is described in Sections 5.2 and 5.3. Section 5.4 discusses the quality of the approximations, and the chapter concludes with a discussion on related approaches in Section 5.5.

5.1 Linear shift-invariant systems

Linearity and shift-invariance are two most welcome properties of a discrete event system. A discrete event system is linear and shift-invariant if it can be described by the following set of equations in max-plus algebra (see Chapter 2):

$$\begin{aligned} t(k+1) &= A \otimes t(k) \oplus B \otimes u(k), \\ y(k) &= C \otimes t(k), \end{aligned} \tag{5.1}$$

where A, B and C are matrices, u is the system's input, t is the system's state, and y the system's output. Matrix A is the system's *state matrix*. From this matrix, we may analyse the maximum possible rate at which events may occur in the system, and compute an eigenschedule that attains this maximum rate.

Every HSDF graph corresponds to a shift-invariant system (see Section 3.2.1), but non-homogeneous SDF graphs may correspond to shift-invariant systems as well. Whether an SDF graph corresponds to a shift-invariant system may be deduced from the (firing) predecessor functions, associated with the channels in the graph. If the firing predecessor function of every channel vw in the graph can be written as a *translation*, i.e.,

$$\forall k \in \mathbb{Z} : \; \pi_{vw}(k) = k - c, \tag{5.2}$$

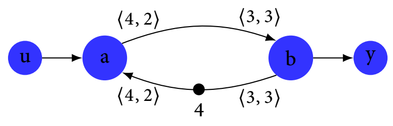
for some constant $c \in \mathbb{Z}$, then the corresponding system is shift-invariant. To illustrate this, consider the CSDF graph depicted in Figure 5.1(a). No execution times are specified for actors a and b (the execution times of u and y is assumed to be zero); we simply assume that the execution times are independent of the firing index. The actor firing perspective associated with the graph is shift-invariant. To see that this is indeed the case, consider the predecessor function associated with channel ab. It is resource-class-wise affine modulo two (see Definition 3.2), and the function satisfies:

$$\pi_{ab}(2k) = \min_{i<2} \left\{ 2 \left\lceil \frac{-\Delta_{ab}(i, 2k)}{6} \right\rceil + i \right\} = 2k + \min\{0, 1\} = 2k$$
$$\pi_{ab}(2k+1) = \min_{i<2} \left\{ 2 \left\lceil \frac{-\Delta_{ab}(i, 2k+1)}{6} \right\rceil + i \right\} = 2k + \min\{2, 1\} = 2k+1,$$

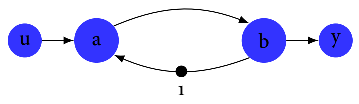
and thus the predecessor function associated with channel ab is fully defined by the simpler function $\pi_{ab}(k) = k$. If we apply the same reasoning to channel ba, we find that $\pi_{ba}(k) = k - 1$. The predecessor function associated with channels ua and by is the identity function, i.e., $\pi_{\mathsf{ua}}(k) = k$. Since all predecessor functions are translations, the system is shift-invariant. Consequently, the system may be represented by a simpler HSDF graph, which is depicted in Figure 5.1(b). Observe that the predecessor functions associated with the channels in the HSDF graph match those derived from the CSDF graph.

If the autonomous part of a system is not shift-invariant but periodically shift-varying, then we may replace it with one that is shift-invariant, and *approximates* the shift-varying one. A useful approximation allows one to translate analysis results, such as throughput, obtained for the latter, to results that hold for the former. We distinguish between approximations that are *conservative*, and those that are *optimistic*. The performance of the conservative approximation translates to *guarantees* on the performance of the approximated system, and the optimistic approximation gives an upper bound on the original system's performance.

Computing a conservative or optimistic approximation involves replacing predecessor functions with functions that are translations. These translations must be such that the temporal dynamics of the approximation is either not better (for a



(a) A shift-invariant CSDF graph.



(b) An equivalent HSDF graph.

Figure 5.1 – An example of a CSDF graph that corresponds to a *shift-invariant* system. The same system can be described by a simpler HSDF graph.

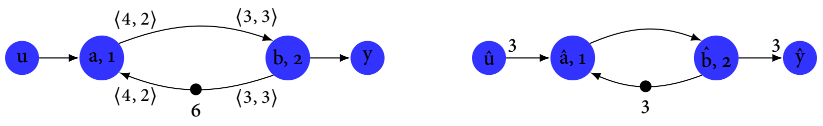
conservative approximation), or not worse (for an optimistic approximation), than that of the original graph.

As an illustration of how a CSDF graph that is not shift-invariant may be conservatively modelled by an HSDF graph, consider the CSDF graph shown in Figure 5.2(a). Note that the graph is only slightly different from Figure 5.1(a); the only difference is the number of tokens placed on channel ba. The graph does not correspond to a shift-invariant system. To see this, consider the dependency imposed by channel ba on firings of a and b: both the first and the second firing of actor a depend on the same firing of actor b. This is reflected by the predecessor function of the channel: it is not a translation, since $\pi_{ba}(2k) = 2k - 2$ and $\pi_{ba}(2k+1) = 2k$.

If we let every three consecutive firings of actor â correspond to a single firing of actor a, and every three consecutive firings of $\hat{\text{b}}$ to a single firing of b, then the predecessor function associated with channel $\hat{\text{b}}$â forms an *upper bound* on the predecessor function associated with channel *ab*: $\pi_{\hat{\text{b}}\hat{\text{a}}}(3k) \geq 3\pi_{\text{ba}}(k)$. Given the correspondence between firings of the two graphs, the predecessor functions of the other three channels are equivalent. For example, we have $\pi_{\hat{\text{u}}\hat{\text{a}}}(3k) = 3\pi_{\text{ua}}(k)$ and $\pi_{\hat{\text{b}}\hat{\text{y}}}(k) = 3\pi_{\text{by}}(k)$.

As a result of the bounding relation between the predecessor functions of the two graphs, the temporal dynamics of the graph of Figure 5.2(a) is reflected in the graph of Figure 5.2(b), which forms a temporal abstraction of the CSDF graph. In both graphs, the sink actor fires once for every firing of the source actor, and, if the latter fires at a low enough rate, the time between the k^{th} firings of u and y is equal to three. At a higher firing rate of source actor u, however, the three tokens in cycle aba cause the firings of the sink actor of the temporal abstraction to lag behind the firings of the sink actor in the CSDF graph.

Furthermore, given the above correspondence between firings of actors in the two graphs, we may map any schedule that is admissible for Figure 5.2(b) to one that is admissible for Figure 5.2(a). That is, we may compute a schedule for the temporal abstraction, and map every third firing of actor â to a single firing of a, and every third firing of $\hat{\text{b}}$ to a firing of b. Observe that in any schedule that is obtained for



(a) A CSDF graph that is not shift-invariant. (b) A temporal abstraction.

Figure 5.2 – A CSDF subgraph that does not correspond to a *shift-invariant* system may be conservatively approximated by a simpler graph that does correspond to a shift-invariant system.

the CSDF graph in this way, actors a and b fire in a strictly sequential fashion.

This example sets the stage for the transformations described in the following sections, where we derive both conservative and optimistic shift-invariant representations from CSDF graph, in a systematic manner.

5.2 Transforming the predecessor function

The novel approach that we take to transforming a CSDF graph into an approximating HSDF graph involves two main steps. As a first step, we derive linear bounds on the predecessor function associated with each channel in the CSDF graph. Here we differ from existing approaches: rather than making assumptions on the temporal behaviour of the graph (such as assuming strictly periodic arrival of tokens), and deriving approximations from these assumptions, we take, as a starting point, the mathematical characterisation as presented in Chapter 3, from which which the temporal behaviour follows.

The derivation of linear bounds on the predecessor function involves basic integer arithmetic, in particular identities (A.13b) and (A.13a) described in Appendix A. These linear bounds change the discrete system into one that is continuous, also called *fluid* by some authors [23]. This continuous system is not shift-invariant, as events occur at different rates. Analysis of such a system may still be performed by formulating and solving a linear program [48, 69]. However, we take an extra step, which is the second main step in our approach, to transform the system into one that is linear, discrete and shift-invariant. This step involves changing the system's *counting* units in such a way, that the predecessor functions of the system are *translations*.

As a result of the second step, the system may be represented by an HSDF graph. As such, existing, well-known analysis techniques may be applied, and analysis results obtained for the HSDF graph can be naturally translated back to the CSDF graph.

In the following two sections, we apply this approach to both the *actor firing* and *token transfer* perspective, which we introduced in Chapter 3.

5.2.1 The actor firing perspective

The actor firing perspective (see Chapter 3) is the periodically shift-varying system that captures the interdependencies between individual firings of actors. The perspective is formulated in terms of the firing predecessor function (see Definition 3.2, which is associated with each of the graph's channels. We derive a linear and shift-invariant approximation of the autonomous system of the firing perspective, by computing linear bounds on the predecessor function. The predecessor function of a CSDF channel may be formulated as the following maximum:

$$\pi_{vw}(k) = \max_{1 \le i \le \varphi_v} \left\{ \left(\left\lceil \frac{-\Delta_{vw}(i-1,k)}{P^{\Sigma+}_{vw}} \right\rceil - 1 \right) \varphi_v + i \right\}. \tag{5.3}$$

The function is residue-class-wise affine (see Proposition 3.1), and its graph follows a repetitive staircase pattern. Tight upper and lower linear bounds may be constructed by choosing an appropriate slope and intercept, where the slope of both bounds is the same. These linear bounds thus have the following form:

$$\hat{\gamma}_{vw}(k) = \alpha_{vw} k - \beta^{\mathrm{up}}_{vw} \tag{5.4a}$$

$$\check{\gamma}_{vw}(k) = \alpha_{vw} k - \beta^{\mathrm{lo}}_{vw}. \tag{5.4b}$$

The predecessor function is residue-class-wise affine, modulo q_v. The (average) slope of the staircase pattern is given by the ratios of the repetition vector entries associated with the actors connected by the channel, in the following way (see Proposition 3.1):

$$\pi_{vw}(k + mq_w) = \pi_{vw}(k) + mq_v.$$

We thus set the slope of the linear bounds on the channel's predecessor function to the ratio of the repetition vector entries of the actors connected by the channel:

$$\alpha_{vw} = \frac{q_v}{q_w},$$

Computation of the intercepts, β^{up} and β^{lo}, requires more effort. These intercepts should be chosen such that the linear bounds are *tight*, i.e., the minimum error between the predecessor function and its linear bound must be zero. Figure 5.3 gives an illustration of how the linear bounds, which we construct in the following two sections, relate to the predecessor function.

Constructing an upper bound

We construct a tight linear upper bound by choosing β^{up} such that, for all $k \in \mathbb{Z}$, we have $\hat{\gamma}_{vw}(k) \geq \pi_{vw}(k)$, and for at least one k, $\hat{\gamma}_{vw}(k) = \pi_{vw}(k)$. This is achieved by setting β^{up} to:

$$\beta^{\mathrm{up}}_{vw} = \min_{k\in\mathbb{Z}} \left\{ \alpha_{vw} k - \pi_{vw}(k) \right\}. \tag{5.5}$$

Rather than defining β^{up}_{vw} as a minimum over the set of integers, as expressed by the above formulation, we may exploit the fact that π_{vw} is resource-class-wise affine modulo the repetition vector of w, and compute β^{up}_{vw} over a smaller set. The repetition vector entry of w depends on the sums of the production and consumption rate vectors of vw; in the worst case, the number of values over which the minimum (5.5) must be taken, is equal to the least common multiple of $P^{\Sigma+}_{vw}$ and $P^{\Sigma-}_{vw}$.

The following rewriting steps reduce the complexity of this computation to one that depends solely on the lengths of the production and consumption rate vectors. As a first step, we expand (5.5) into:

$$\begin{aligned}
\beta^{\mathrm{up}}_{vw} &= \min_{k} \left\{ \frac{q_v}{q_w} k - \max_{i<\varphi_v} \left\{ \left\lceil \frac{-\Delta_{vw}(i,k)}{P^{\Sigma+}_{vw}} \right\rceil \varphi_v + i \right\} \right\} + \varphi_v - 1 \\
&= \min_{k} \left\{ \min_{i<\varphi_v} \left\{ \left(\frac{kP^{\Sigma-}_{vw}}{\varphi_w P^{\Sigma+}_{v}} - \left\lceil \frac{-\Delta_{vw}(i,k)}{P^{\Sigma+}_{vw}} \right\rceil \right) \varphi_v - i \right\} \right\} + \varphi_v - 1.
\end{aligned}$$

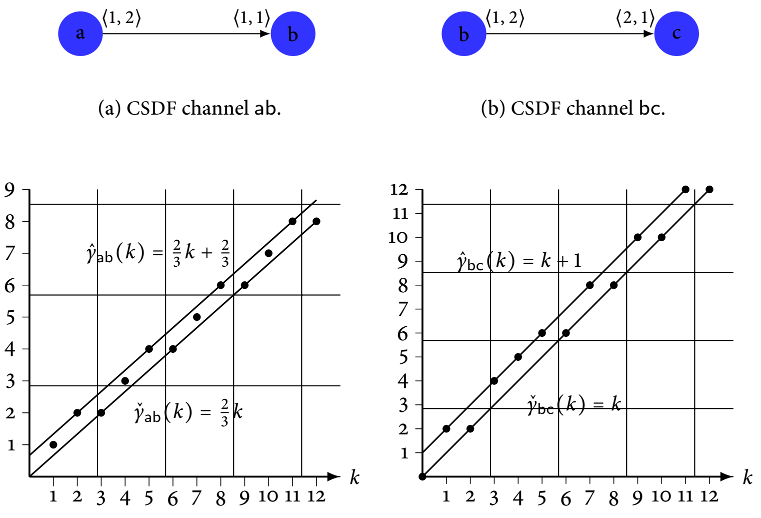
We further reduce the complexity of the computation by exploiting the periodicity of $\Delta_{vw}(i,k)$ in k. Substituting $k'\varphi_w + j$ for k gives:

$$\begin{aligned}
\beta^{\mathrm{up}}_{vw} &= \min_{k'} \min_{\substack{i<\varphi_v \\ j<\varphi w}} \left\{ \left(\frac{k'P^{\Sigma-}_{vw}}{P^{\Sigma+}_{vw}} - \left\lceil \frac{k'P^{\Sigma-}_{vw} - \Delta_{vw}(i,j)}{P^{\Sigma+}_{vw}} \right\rceil \right) \varphi_v - i + j\frac{q_v}{q_w} \right\} + \varphi_v - 1 \\
&= \min_{\substack{i<\varphi_v \\ j<\varphi_w}} \left\{ \min_{k'} \left\{ \frac{k'P^{\Sigma-}_{vw}}{P^{\Sigma+}_{vw}} - \left\lceil \frac{k'P^{\Sigma-}_{vw} - \Delta_{vw}(i,j)}{P^{\Sigma+}_{vw}} \right\rceil \right\} \varphi_v - i + j\frac{q_v}{q_w} \right\} + \varphi_v - 1 \\
&\overset{\text{(A.13a)}}{=} \min_{\substack{i<\varphi_v \\ j<\varphi_w}} \left\{ \frac{g_{vw}\varphi_v \left\lceil \frac{\Delta_{vw}(i,j)+1}{g_{vw}} \right\rceil}{P^{\Sigma+}_{vw}} - i + j\frac{q_v}{q_w} \right\} - 1.
\end{aligned}$$

Putting everything together gives the following linear predecessor function:

$$\hat{\gamma}_{vw}(k) = \frac{q_v}{q_w} k - \min_{\substack{i<\varphi_v \\ j<\varphi_w}} \left\{ \frac{g_{vw}\varphi_v \left\lceil \frac{\Delta_{vw}(i,j)+1}{g_{vw}} \right\rceil}{P^{\Sigma+}_{vw}} - i + j\frac{q_v}{q_w} - 1 \right\}. \tag{5.6}$$

A tight linear bound on the predecessor function associated with CSDF channel vw may thus be computed in time proportional to the product of the periods of actors v and w, i.e., $\varphi_v\varphi_w$. Note that in case the greatest common divisor g_{vw} equals one, the bound may be computed in time proportional to $\varphi_v + \varphi_w$, by separating the terms in i from the terms in j.



(a) CSDF channel ab.

(b) CSDF channel bc.

(c) Linear bounds on π_{ab}.

(d) Linear bounds on π_{bc}.

Figure 5.3 – Tight linear lower and upper bounds on the predecessor functions of channels ab and bc.

Constructing a lower bound

We may construct a lower bound, in a way similar to the construction of an upper bound, described above. Rather than using the max-formulation (5.3) of the predecessor function, we now use its formulation as a minimum, given by (3.2):

$$\pi_{vw}(k) = \min_{0 \le i < \varphi_v} \left\{ \left\lceil \frac{-\Delta_{vw}(i,k)}{P^{\Sigma+}_{vw}} \right\rceil \varphi_v + i \right\}.$$

For the lower bound, we compute the intercept β^{lo}_{vw} using:

$$\beta^{\text{lo}}_{vw} = \max_k \left\{ \alpha_{vw} k - \pi_{vw}(k) \right\}, \tag{5.7}$$

which we expand into:

$$\begin{aligned} \beta^{\text{lo}}_{vw} &= \max_k \left\{ \frac{q_v}{q_w} k - \min_{i < \varphi_v} \left\{ \left\lceil \frac{-\Delta_{vw}(i,k)}{P^{\Sigma+}_{vw}} \right\rceil \varphi_v + i \right\} \right\} \\ &= \max_k \left\{ \max_{i < \varphi_v} \left\{ \left(\frac{k P^{\Sigma-}_{vw}}{\varphi_w P^{\Sigma+}_{vw}} - \left\lceil \frac{-\Delta_{vw}(i,k)}{P^{\Sigma+}_{vw}} \right\rceil \right) \varphi_v - i \right\} \right\}. \end{aligned}$$

Again, by substituting $k'\varphi_w + j$ for k, we rewrite this into:

$$\begin{aligned}
\beta^{\mathrm{lo}}_{vw} &= \max_{k'} \max_{\substack{i<\varphi_v \\ j<\varphi_w}} \left\{ \left(\frac{k' P^{\Sigma-}_{vw}}{P^{\Sigma+}_{vw}} - \left\lceil \frac{k' P^{\Sigma-}_{vw} - \Delta_{vw}(i,j)}{P^{\Sigma+}_{vw}} \right\rceil \right) \varphi_v - i + j \frac{q_v}{q_w} \right\} \\
&= \max_{\substack{i<\varphi_v \\ j<\varphi_w}} \left\{ \max_{k'} \left\{ \frac{k' P^{\Sigma-}_{vw}}{P^{\Sigma+}_{vw}} - \left\lceil \frac{k' P^{\Sigma-}_{vw} - \Delta_{vw}(i,j)}{P^{\Sigma+}_{vw}} \right\rceil \right\} \varphi_v - i + j \frac{q_v}{q_w} \right\} \\
&\overset{\text{(A.13b)}}{=} \max_{\substack{i<\varphi_v \\ j<\varphi_w}} \left\{ \frac{g_{vw} \varphi_v \left\lfloor \frac{\Delta_{vw}(i,j)}{g_{vw}} \right\rfloor}{P^{\Sigma+}_{vw}} - i + j \frac{q_v}{q_w} \right\}.
\end{aligned}$$

This gives the following tight linear lower bound on the CSDF predecessor function:

$$\check{\gamma}_{vw}(k) = \frac{q_v}{q_w} k - \max_{\substack{i<\varphi_v \\ j<\varphi_w}} \left\{ \frac{g_{vw} \varphi_v \left\lfloor \frac{\Delta_{vw}(i,j)}{g_{vw}} \right\rfloor}{P^{\Sigma+}_{vw}} - i + j \frac{q_v}{q_w} \right\}. \tag{5.8}$$

Figure 5.3 depicts two sets of linear bounds, constructed using (5.6) and (5.8), for the predecessor function of two different channels.

Changing counting units

The linear bounds that we derived in the previous section do not adhere to (5.2). That is, they do not correspond to translations, and as such, can not directly be used to construct a shift-invariant system. Shift-invariance signifies that firing times progress at the same rate. As the rates at which CSDF actors fire are interrelated through the graph's repetition vector, counting the number of *iterations* an actor has completed, rather than its completed firings, yields a shift-invariant system.

Such a change in counting is obtained by scaling both the domain and co-domain of (5.6) and (5.8). We scale down the domains of $\hat{\gamma}_{vw}$ and $\check{\gamma}_{vw}$ by a factor of q_w, and the co-domain by a factor q_v. This gives the following transformed linear bounds:

$$\frac{1}{q_v} \hat{\gamma}_{vw}(k q_w) = k - \min_{\substack{i<\varphi_v \\ j<\varphi_w}} \left\{ \frac{g_{vw} \varphi_v \left\lceil \frac{\Delta_{vw}(i,j)+1}{g_{vw}} \right\rceil}{q_v P^{\Sigma+}_{vw}} - \frac{i+1}{q_v} + \frac{j}{q_w} \right\}, \tag{5.9a}$$

$$\frac{1}{q_v} \check{\gamma}_{vw}(k q_w) = k - \max_{\substack{i<\varphi_v \\ j<\varphi_w}} \left\{ \frac{g_{vw} \varphi_v \left\lfloor \frac{\Delta_{vw}(i,j)}{g_{vw}} \right\rfloor}{q_v P^{\Sigma+}_{vw}} - \frac{i}{q_v} + \frac{j}{q_w} \right\}. \tag{5.9b}$$

These two transformed functions are translations, but not integer: they map an integer to a *rational number*, as the intercept (i.e., the term min$\{\ldots\}$) in (5.9a) and (5.9b) is, in general, not an integer. We again perform a scaling, of both the domain and co-domain, such that the right-hand sides in (5.9a) and (5.9b) are integer translations. As a scaling factor, we use the scalar structural invariant associated with a graph, $\mathcal{N}$ (see Section 2.1.3). This gives the following linear upper bound:

$$\begin{aligned}\hat{\pi}_{vw}(k) &= \frac{\mathcal{N}}{q_v}\hat{\gamma}_{vw}\left(k\frac{q_w}{\mathcal{N}}\right)\\ &= k - s_{vw}\min_{\substack{i<\varphi_v\\ j<\varphi_w}}\left\{g_{vw}\left\lceil\frac{\Delta_{vw}(i,j)+1}{g_{vw}}\right\rceil - \frac{(i+1)P^{\Sigma+}_{vw}}{\varphi_v} + \frac{jP^{\Sigma-}_{vw}}{\varphi_w}\right\}.\end{aligned} \tag{5.10}$$

Note that the min-term in this expression is indeed an integer, by definition of the flow conservation vector, (2.2). In a similar fashion, we derive the following linear lower bound:

$$\begin{aligned}\check{\pi}_{vw}(k) &= \frac{\mathcal{N}}{q_v}\check{\gamma}_{vw}\left(k\frac{q_w}{\mathcal{N}}\right)\\ &= k - s_{vw}\max_{\substack{i<\varphi_v\\ j<\varphi_w}}\left\{g_{vw}\left\lfloor\frac{\Delta_{vw}(i,j)}{g_{vw}}\right\rfloor - \frac{iP^{\Sigma+}_{vw}}{\varphi_v} + \frac{jP^{\Sigma-}_{vw}}{\varphi_w}\right\}.\end{aligned} \tag{5.11}$$

The scaling performed above is analogous to multiplying the number of tokens, on each channel in an HSDF graph, with the same factor. Multiplying each channel's number of tokens by a factor f scales the cycle ratio of each cycle (and thus the graph's maximum cycle ratio) by a factor of $\frac{1}{f}$. As we shall see later, the throughput computed for the HSDF graphs constructed from the derived predecessor functions, $\hat{\pi}$ and $\check{\pi}$, must thus be scaled by $\mathcal{N}$, to obtain (a bound on) the throughput of the original CSDF graph.

5.2.2 The token transfer perspective

For the token transfer perspective, we have the transfer predecessor function as given by Definition 3.7, which we repeat here for convenience:

$$\pi^{\bullet}_{uv,vw}(k) = \min_{i<\varphi_v}\left\{\left\lceil\frac{k-\sum_{l=1}^{i}\rho^{+}_{vw}(l)}{P^{\Sigma+}_{vw}}\right\rceil P^{\Sigma-}_{uv} + \sum_{l=1}^{i}\rho^{-}_{uv}(l) - \delta_{uv}\right\}.$$

It may be formulated as a maximum, by:

$$\pi^{\bullet}_{uv,vw}(k) = \max_{i<\varphi_v}\left\{\left\lfloor\frac{k-1-\sum_{l=1}^{i}\rho^{+}_{vw}(l)}{P^{\Sigma+}_{vw}}\right\rfloor P^{\Sigma-}_{uv} + \sum_{l=1}^{i+1}\rho^{-}_{uv}(l) - \delta_{uv}\right\}. \tag{5.12}$$

This function is resource-class-wise affine, modulo the number of tokens produced (and consumed) from the corresponding channel, $\frac{q_v P^{\Sigma+}_{vw}}{\varphi_v}$. We thus set the slope of the linear bound to:

$$\alpha_{uvw} = \varphi_v\frac{\pi^{\bullet}_{uvw}\left(k+\frac{q_vP^{\Sigma+}_{vw}}{\varphi_v}\right) - \pi^{\bullet}_{uvw}(k)}{q_vP^{\Sigma+}_{vw}} \overset{(3.9)}{=} \frac{\varphi_v q_u P^{\Sigma+}_{uv}}{\varphi_u q_v P^{\Sigma+}_{vw}} \overset{(2.1)}{=} \frac{P^{\Sigma-}_{vw}}{P^{\Sigma+}_{vw}}.$$

The computation of upper and lower bounds on the transfer predecessor function follows the same steps as those applied, in the previous section, to the firing predecessor function. We denote these upper and lower bounds by $\hat{\gamma}^{\bullet}$ and $\check{\gamma}^{\bullet}$. Similar to the bounds obtained in the previous section, they are formulated as follows:

$$\hat{\gamma}^{\bullet}_{uvw}(k) = \alpha_{uvw}k - \beta^{\text{up}}_{uvw} \tag{5.13a}$$

$$\check{\gamma}^{\bullet}_{uvw}(k) = \alpha_{uvw}k - \beta^{\text{lo}}_{uvw}. \tag{5.13b}$$

We derive each of the above two linear bounds in a separate section, below. Because of the overlap between this and the previous section and for the sake of compactness, we explain the derivations that follow in fewer steps.

We compute the intercept β^{lo}_{uvw} using the formulation of the predecessor function as a maximum, (5.12). To eliminate the variable k, we use the same approach as we did for the actor perspective. That is, we use (A.13a) to compute the minimum of the difference between the bound and the predecessor function:

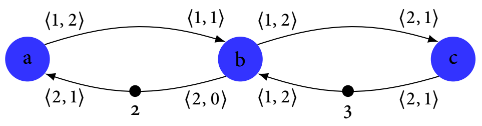
$$\begin{aligned}
\beta^{\text{up}}_{uvw} &= \min_{k\in\mathbb{Z}}\{\alpha_{vw}k - \pi^{\bullet}_{uvw}(k)\} \\
&= \min_{k}\left\{\alpha_{vw}k - \max_{i<\varphi_v}\left\{\left\lfloor \frac{k-1-\sum_{l=1}^{i}\rho^{+}_{vw}(l)}{P^{\Sigma+}_{vw}}\right\rfloor P^{\Sigma-}_{uv} - \Delta_{uv}(0,i+1)\right\}\right\} \\
&= \min_{k}\left\{\min_{i<\varphi_v}\left\{\left(\frac{k}{P^{\Sigma+}_{vw}} - \left\lfloor \frac{k-1-\sum_{l=1}^{i}\rho^{+}_{vw}(l)}{P^{\Sigma+}_{vw}}\right\rfloor\right) P^{\Sigma-}_{uv} + \Delta_{uv}(0,i+1)\right\}\right\} \\
&\overset{\text{(A.13a)}}{=} \min_{i<\varphi_v}\left\{P^{\Sigma-}_{uv}\left(\frac{\sum_{l=1}^{i}\rho^{+}_{vw}(l)+1}{P^{\Sigma+}_{vw}}\right) + \Delta_{uv}(0,i+1)\right\}.
\end{aligned}$$

In a similar fashion, this time using (A.13b) to eliminate k, we compute the intercept β^{lo}_{uvw}:

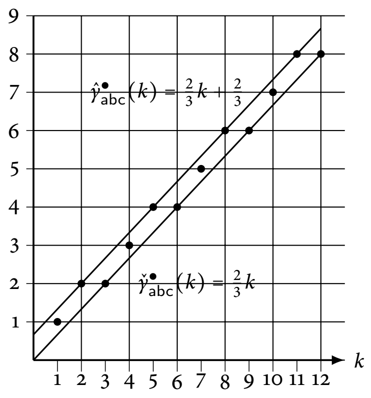
$$\begin{aligned}
\beta^{\text{lo}}_{uvw} &= \max_{k}\{\alpha_{vw}k - \pi^{\bullet}_{uvw}(k)\} \\
&= \max_{k}\left\{\alpha_{vw}k - \min_{i<\varphi_v}\left\{\left\lceil \frac{k-\sum_{l=1}^{i}\rho^{+}_{vw}(l)}{P^{\Sigma+}_{vw}}\right\rceil P^{\Sigma-}_{uv} - \Delta_{uv}(0,i)\right\}\right\} \\
&\overset{\text{(A.13b)}}{=} \max_{i<\varphi_v}\left\{P^{\Sigma-}_{uv}\left(\frac{\sum_{l=1}^{i}\rho^{+}_{vw}(l)}{P^{\Sigma+}_{vw}}\right) + \Delta_{uv}(0,i)\right\}.
\end{aligned}$$

In the previous section we changed the counting units of the linear bounds in such a way that the linear bounds are functions that map completed iterations of one actor to completed iterations of another actor. If we follow the same argument for the token perspective, then we must rather scale by the *number of tokens*, produced onto or consumed from a channel, during a single iteration. We thus have, for the linear upper bound:

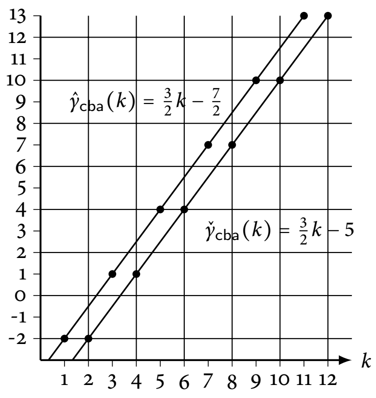
$$\frac{\varphi_v}{P^{\Sigma-}_{uv}q_v}\hat{\gamma}^{\bullet}_{uvw}\left(\frac{kP^{\Sigma+}_{vw}q_v}{\varphi_v}\right) = k - \frac{\varphi_v}{q_v}\min_{i<\varphi_v}\left\{\frac{\sum_{l=1}^{i}\rho^{+}_{vw}(l)+1}{P^{\Sigma+}_{vw}} + \frac{\Delta_{uv}(0,i+1)}{P^{\Sigma-}_{uv}}\right\}.$$



(a) CSDF graph.



(b) Linear bounds on $\pi^{\bullet}_{\mathsf{abc}}$.



(c) Linear bounds on $\pi^{\bullet}_{\mathsf{cba}}$.

Figure 5.4 – Tight linear lower and upper bounds on the transfer predecessor functions of paths abc and cba.

Figure 5.4 gives an illustration of the construction of linear upper and lower bounds, for two different paths in a graph, using the steps given in this section.

If we again multiply the shift with the scalar structural invariant $\mathcal{N}$, we obtain the following predecessor function in $\mathbb{Z}$:

$$\hat{\pi}^{\bullet}_{uvw}(k) = s_{uv}\hat{\gamma}^{\bullet}_{uvw}\left(\frac{k}{s_{vw}}\right) = k - \min_{i<\varphi_v}\left\{ s_{vw}\sum_{l=1}^{i}\rho^{+}_{vw}(l) + s_{vw} + s_{uv}\Delta_{uv}(0, i+1)\right\}. \tag{5.14}$$

For the linear lower bound, we obtain, after changing counting units and scaling by $\mathcal{N}$, the following integer-valued predecessor function:

$$\check{\pi}^{\bullet}_{uvw}(k) = s_{uv}\check{\gamma}^{\bullet}_{uvw}\left(\frac{k}{s_{vw}}\right) = k - \max_{i<\varphi_v}\left\{ s_{vw}\sum_{l=1}^{i}\rho^{+}_{vw}(l) + s_{uv}\Delta_{uv}(0, i)\right\}. \tag{5.15}$$

5.3 Single-rate approximations

The two pairs of predecessor functions constructed in the previous section give rise to *four* shift-invariant systems. To obtain such a system, we need, as a final step, to replace each actor's execution time *vector* by a *scalar*. A *pessimistic* system is obtained by replacing each execution time vector by its maximum. Similarly, by replacing each vector with its minimum, we obtain an *optimistic* system.

5.3.1 Optimistic and pessimistic systems

For the actor firing perspective, we obtain a pessimistic and optimistic system by replacing the predecessor function, associated with each channel in the original CSDF graph, with its upper and lower bound, given by (5.10) and (5.11). If we furthermore replace, for each actor, the vector of execution times with, respectively, its maximum and minimum, then we obtain the following two systems:

$$\hat{t}_w(k) = \bigoplus_{vw} \hat{t}_v\left(\hat{\pi}_{vw}(k)\right) \otimes \max_i \tau_w(i) \tag{5.16a}$$

$$\check{t}_w(k) = \bigoplus_{vw} \check{t}_v\left(\check{\pi}_{vw}(k)\right) \otimes \min_i \tau_w(i), \tag{5.16b}$$

where $\hat{t}_w(k)$ and $\check{t}$ denote the time at which actor w completes its k^{th} firing, in respectively the pessimistic and optimistic system. These two systems may be represented by HSDF graphs: each predecessor function of the form $\pi_{vw}(k) = k - m$ corresponds to an HSDF channel with m tokens. We refer to these HSDF graphs as *single-rate approximations*. Algorithm 6 gives the transformation of a CSDF graph into its single-rate approximation, in a procedural way.

As an example, consider the graph of Figure 5.5. It is consistent: its repetition vector is given by: $q_{\mathsf{a}} = 4$, and $q_{\mathsf{b}} = q_{\mathsf{c}} = 6$, and its flow normalisation vector is as follows: $s_{\mathsf{ab}} = s_{\mathsf{ba}} = 6$, and $s_{\mathsf{bc}} = s_{\mathsf{cb}} = 4$, and $s_{\mathsf{aa}} = 9$. If we compute linear bounds on each of the channels' predecessor functions, and replace each CSDF channel by the HSDF channel that corresponds to a linear bound, we obtain the optimistic and pessimistic single-rate approximations, shown in Figure 5.6. In these figures, we have followed a naming convention for actors in the single-rate graphs, which indicates their correspondence to actors in the original graph: an actor v in the latter corresponds to actor $\hat{v}$ in the pessimistic, and actor $\check{v}$ in the optimistic single-rate approximation.

Actors in the single-rate approximation represent *partial iterations*: a single firing in the original CSDF graph corresponds to a set of consecutive firings in the single-rate approximation. This is due to the change in counting units, which was necessary to obtain shift-invariance. The relation between the number of HSDF actor firings and the number of firings of the corresponding CSDF actor is determined by the graph's scalar invariant, $\mathcal{N}$, and its repetition vector. A single graph iteration of the CSDF graph corresponds to the completion of $\mathcal{N}$ firings of each of the HSDF actors, in the single-rate approximation. In other words, q_v firings of CSDF actor v correspond to $\mathcal{N}$ firings of HSDF actors $\hat{v}$ and $\check{v}$.

Algorithm 6: Transforms a CSDF graph into optimistic and pessimistic single-rate approximations, derived from the graph's actor firing perspective.

```
input  : A consistent CSDF graph $\mathcal{G}$, with flow normalisation vector $s$.
output : An optimistic and pessimistic single-rate approximation of $\mathcal{G}$.
$\mathcal{H}_{\text{opt}} \longleftarrow$ empty graph
$\mathcal{H}_{\text{pess}} \longleftarrow$ empty graph
foreach actor $v$ in $\mathcal{G}$ do
    Add actor $\hat{v}$ to $\mathcal{H}_{\text{pess}}$
    Add actor $\check{v}$ to $\mathcal{H}_{\text{opt}}$
    $\tau_{\hat{v}} \longleftarrow \max_{i=1}^{\varphi_v} \tau_v(i)$
    $\tau_{\check{v}} \longleftarrow \min_{i=1}^{\varphi_v} \tau_v(i)$
foreach channel $vw$ in $\mathcal{G}$ do
    Add HSDF channel $\hat{v}\hat{w}$ to $\mathcal{H}_{\text{pess}}$
    Add HSDF channel $\check{v}\check{w}$ to $\mathcal{H}_{\text{opt}}$
    $\delta_{\hat{v}\hat{w}} \longleftarrow s_{vw} \min_{\substack{i<\varphi_v \\ j<\varphi_w}} \left\{ g_{vw} \left\lceil \frac{\Delta_{vw}(i,j)+1}{g_{vw}} \right\rceil - \frac{(i+1)P^{\Sigma+}_{vw}}{\varphi_v} + \frac{jP^{\Sigma-}_{vw}}{\varphi_w} \right\}$
    $\delta_{\check{v}\check{w}} \longleftarrow s_{vw} \max_{\substack{i<\varphi_v \\ j<\varphi_w}} \left\{ g_{vw} \left\lfloor \frac{\Delta_{vw}(i,j)}{g_{vw}} \right\rfloor - \frac{iP^{\Sigma+}_{vw}}{\varphi_v} + \frac{jP^{\Sigma-}_{vw}}{\varphi_w} \right\}$
return $\mathcal{H}_{\text{opt}}$, $\mathcal{H}_{\text{pess}}$
```

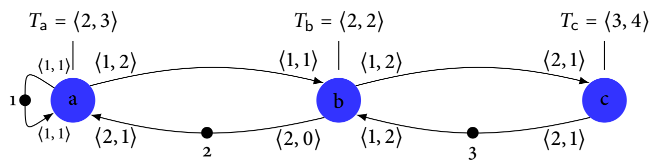
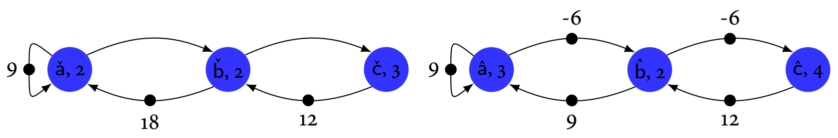

$T_a = \langle 2, 3\rangle$
$T_b = \langle 2, 2\rangle$
$T_c = \langle 3, 4\rangle$
⟨1, 1⟩
⟨1, 2⟩
⟨1, 1⟩
⟨1, 2⟩
⟨2, 1⟩
1
a
b
c
⟨1, 1⟩
⟨2, 1⟩
2
⟨2, 0⟩
⟨1, 2⟩
3
⟨2, 1⟩

Figure 5.5 – An example of a consistent CSDF graph.



(a) Optimistic HSDF graph

(b) Pessimistic HSDF graph

Figure 5.6 – Optimistic and pessimistic single-rate approximations derived from the actor firing perspective of the CSDF shown in Figure 5.5, using Algorithm 6.

The scalar invariant, $\mathcal{N}$, of the graph shown in Figure 5.5, equals 36. Thus, every $\frac{\mathcal{N}}{q_{\mathsf{a}}} = 9$ consecutive firings of actor $\hat{\mathsf{a}}$ in the graph of Figure 5.6(b), correspond to a single firing of CSDF actor a in Figure 5.5. Similarly, six firings of $\hat{\mathsf{b}}$ correspond to a single firing of CSDF actor b.

The correspondence between firings, illustrated above, is a consequence of the relation between the predecessor function associated with channels in the CSDF graph, and the linear lower and upper bounds derived earlier. This relation may be generalised to *paths* in a straightforward manner, by applying Equation 5.10 to the definition of the predecessor function. This gives rise to the following proposition.

Proposition 5.1. *Let $\mathcal{G}$ be a consistent CSDF graph, with repetition vector q, and scalar invariant $\mathcal{N}$. Furthermore, let $\mathcal{H}_{pess}$ and $\mathcal{H}_{opt}$ be the pessimistic and optimistic single-rate approximations of $\mathcal{G}$, obtained from $\mathcal{G}$ by applying Algorithm 6. The following relation, between the predecessor functions associated with paths in the three graphs, holds:*

$$\forall k \in \mathbb{Z} : \frac{q_{v_1}}{\mathcal{N}_{\mathcal{G}}} \pi_{\hat{v}_1 \ldots \hat{v}_n}\left(\frac{k\mathcal{N}_{\mathcal{G}}}{q_{v_n}}\right) \geq \pi_{v_1 \ldots v_n}(k) \geq \frac{q_{v_1}}{\mathcal{N}_{\mathcal{G}}} \pi_{\check{v}_1 \ldots \check{v}_n}\left(\frac{k\mathcal{N}_{\mathcal{G}}}{q_{v_n}}\right),$$

for any path $p = v_1 \ldots v_n$ in $\mathcal{G}$.

For the token transfer perspective, we obtain, in a similar way, two systems that are formulated in terms of the transfer predecessor functions given by (5.15) and (5.14). We denote by $\hat{t}$ the state of the pessimistic system, and by $\check{t}$ the state of the optimistic system:

$$\hat{t}^{\bullet}_{vw}(k) = \bigoplus_{uv} \hat{t}^{\bullet}_{uv}\left(\hat{\pi}_{uvw}(k)\right) \otimes \max_{i} \tau_v(i) \tag{5.17a}$$

$$\check{t}^{\bullet}_{vw}(k) = \bigoplus_{uv} \check{t}^{\bullet}_{uv}\left(\check{\pi}_{uvw}(k)\right) \otimes \min_{i} \tau_v(i). \tag{5.17b}$$

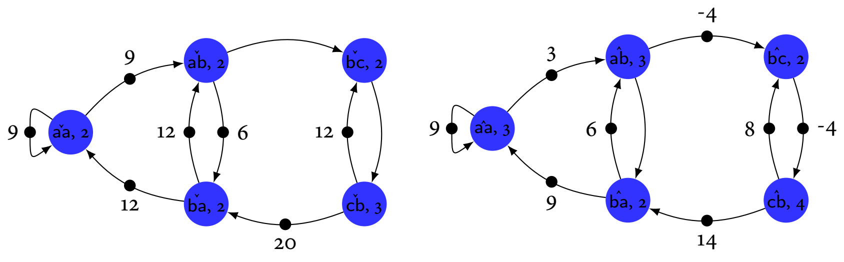
In the single-rate approximations, derived from these two systems, each actor corresponds to a *channel* in the CSDF graph. Again, there is a correspondence, between firings of actors in the approximation on the one hand, and transfer of tokens over channels in the CSDF graph. The completion of a single graph iteration corresponds to the completion of $\mathcal{N}$ firings of each actor in the single-rate approximation. That is, the number of tokens transferred over a channel vw in the CSDF graph, during a single graph iteration, corresponds to $\mathcal{N}$ firings of actor vw in the approximation. This means that s_{vw} firings of actor vw (in the approximation) correspond to the transfer of a single token over CSDF channel vw. Again, the pessimistic and optimistic systems, given above, correspond to single-rate approximations. The transformation of a CSDF graph into its single-rate approximation, derived from the token transfer perspective of the former, is given in Algorithm 7.

Figure 5.7 illustrates the single-rate approximations derived from the token transfer perspective. Every actor in these graphs corresponds to a *channel* in the original

Algorithm 7: Transforms a CSDF graph into optimistic and pessimistic single-rate approximations, derived from the graph's token transfer perspective.

```
input  : A consistent CSDF graph $\mathcal{G}$, with flow normalisation vector $s$, repetition
         vector $q$, and scalar invariant $\mathcal{N}_{\mathcal{G}}$.
output : An optimistic and pessimistic single-rate approximation of $\mathcal{G}$.

$\mathcal{H}_{\text{opt}} \longleftarrow$ empty graph
$\mathcal{H}_{\text{pess}} \longleftarrow$ empty graph
foreach channel $vw$ in $\mathcal{G}$ do
    Add actor $\hat{vw}$ to $\mathcal{H}_{\text{pess}}$
    Add actor $\check{vw}$ to $\mathcal{H}_{\text{opt}}$
    $\tau_{\hat{vw}} \longleftarrow \max_{i=1}^{\varphi_v} \tau_v(i)$
    $\tau_{\check{vw}} \longleftarrow \min_{i=1}^{\varphi_v} \tau_v(i)$
foreach channel $vw$ in $\mathcal{G}$ do
    foreach incoming channel $uv$ of $v$ in $\mathcal{G}$ do
        Add HSDF channel $\hat{uv}, \hat{vw}$ to $\mathcal{H}_{\text{pess}}$
        Add HSDF channel $\check{uv}, \check{vw}$ to $\mathcal{H}_{\text{opt}}$
        $\delta_{\hat{uv}\hat{vw}} \longleftarrow \min_{i<\varphi_v} \left\{ s_{vw} \sum_{l=1}^{i} \rho^{+}_{vw}(l) + s_{vw} + s_{uv}\Delta_{uv}(0, i+1) \right\}$
        $\delta_{\check{uv}\check{vw}} \longleftarrow \max_{i<\varphi_v} \left\{ s_{vw} \sum_{l=1}^{i} \rho^{+}_{vw}(l) + s_{uv}\Delta_{uv}(0, i) \right\}$
return $\mathcal{H}_{\text{opt}}, \mathcal{H}_{\text{pess}}$
```



(a) Optimistic HSDF graph (b) Pessimistic HSDF graph

Figure 5.7 – Optimistic and pessimistic single-rate approximations derived from the token transfer perspective of the CSDF graph of Figure 5.5.

CSDF graph. The optimistic and pessimistic graphs are larger than their counterparts in the actor firing perspective, since the CSDF graph has more channels than actors.

If we compare the cycles in the single-rate approximations of the two perspectives, then we find that for the optimistic approximations, the number of tokens on the cycles are the same: for example, cycle aba in the graph of Figure 5.6(a) has 18 tokens, which corresponds to the number of tokens on cycle ab, ba, ab in Figure 5.7(a). Likewise, cycle ab, ba, aa, ab (in Figure 5.7(a)) has 27 tokens, and so does cycle abaa (in Figure 5.6(a)). However, if we compare the pessimistic approximations, then there are differences: cycle aba in Figure 5.6(b) has three tokens, whereas the corresponding cycle in Figure 5.7(b) has six. Also, cycle bcb in the former has six tokens, whereas the corresponding cycle in the latter has four.

This difference in the number of tokens on channels translates to a difference in maximum cycle ratio, of the pessimistic approximations: in the graph of Figure 5.6(b), the critical cycle is aba, and has a (maximum) cycle ratio of $\frac{5}{3}$. For Figure 5.7(b), we find a lower maximum cycle ratio, of $\frac{6}{4}$, attained by critical cycle bc, cb, bc. To translate these maximum cycle ratios to a throughput for the CSDF graph, we must take their inverse, and multiply it by $\mathcal{N}$, to undo the scaling step of the transformation. From the single-rate approximation of the actor firing perspective, we now learn that the throughput, of the CSDF graph of Figure 5.5, is at least $\frac{3}{5\cdot 36} = \frac{1}{60}$. Likewise, the token transfer perspective gives a lower bound of $\frac{1}{54}$. The latter is higher, which means that, for this example, the token transfer perspective gives a tighter lower bound on throughput than the actor firing perspective (the exact throughput, computed from the graph's single-rate *equivalent*, equals $\frac{1}{24}$). In Section 5.3.4, we study the difference in the approximation accuracy, obtained by the two perspectives, in more detail.

We conclude this section with the following proposition, which is the dual to Proposition 5.1, and gives the relationship between the transfer predecessor functions, of a CSDF graph, and its single-rate approximations constructed using Algorithm 7:

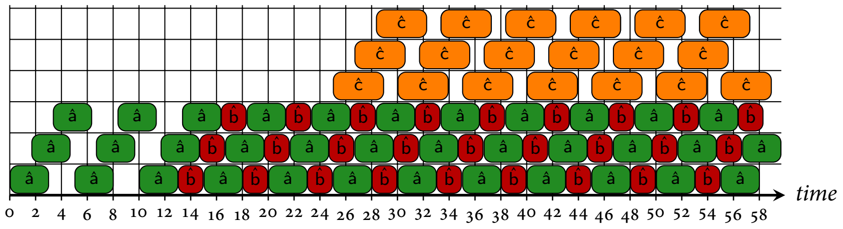
Proposition 5.2. *Let $\mathcal{G}$ be a consistent CSDF graph, with flow normalisation vector s, and scalar invariant $\mathcal{N}$. Furthermore, let $\mathcal{H}_{pess}$ and $\mathcal{H}_{opt}$ be the pessimistic and optimistic single-rate approximations of $\mathcal{G}$, obtained from its token transfer perspective. Each channel vw in $\mathcal{G}$ maps onto an actor $\widehat{vw}$ in $\mathcal{H}_{pess}$, and $\widecheck{vw}$ in $\mathcal{H}_{opt}$. The following relation, between the predecessor functions associated with paths in the three graphs, holds:*

$$\forall k \in \mathbb{Z} : \frac{1}{s_{v_1v_2}} \pi_{\widehat{v_1v_2}\ldots\widehat{v_{n-1}v_n}}\left(ks_{v_{n-1}v_n}\right) \geq \pi^{\bullet}_{v_1\ldots v_n}(k) \geq \frac{1}{s_{v_1v_2}} \pi_{\widecheck{v_1v_2}\ldots\widecheck{v_{n-1}v_n}}\left(ks_{v_{n-1}v_n}\right),$$

for any path $p = v_1 \ldots v_n$, with $n > 2$, in $\mathcal{G}$.

5.3.2 Computing strictly periodic schedules

Using the correspondence between the firings of actors, in the CSDF graph and its single-rate approximations, we can derive a schedule for the former, from the


(a) The eigenschedule of the pessimistic single-rate approximation of Figure 5.6(b).

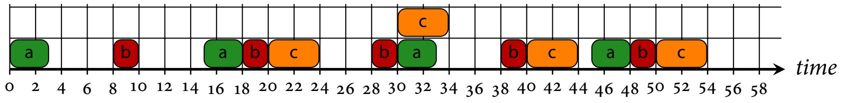
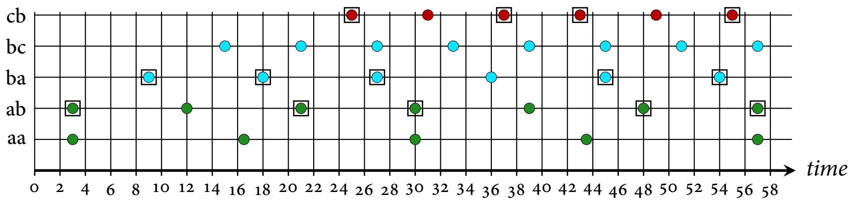

(b) A strictly periodic schedule for the CSDF graph of Figure 5.5.

Figure 5.8 – A strictly periodic schedule for a CSDF graph can be constructed by sampling the eigenschedule of its pessimistic single-rate approximation.

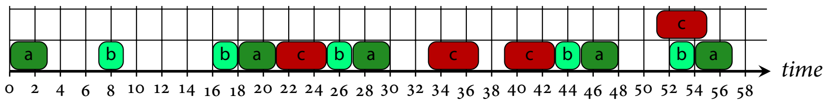
schedule of the latter. We compute this schedule from the pessimistic single-rate approximation, as the maximum rate at which actors in this graph fire, gives a lower bound on throughput. We start this section with an example that continues the example of the previous section.

Consider again the graph depicted in Figure 5.6(b). The graph has a maximum cycle ratio of $\frac{5}{3}$. This means that, in the associated *eigenschedule* (see Section 2.2.4), each actor fires in a strictly periodic fashion, completing a firing every $\frac{5}{3}$ time units. This eigenschedule is shown in Figure 5.8(a). The schedule shows that actor $\hat{a}$ must complete several firings, before actors $\hat{b}$ and $\hat{c}$ may start their firings: the first firing of $\hat{b}$ starts at time 13, and the first firing of $\hat{c}$ starts at time 25.

Not every firing that appears in the eigenschedule of the single-rate approximation corresponds to a firing of an actor in the approximated CSDF graph. Every nine consecutive firings of HSDF actor $\hat{a}$ correspond to a single firing of CSDF actor a. For both actors $\hat{b}$ and $\hat{c}$, six of their firings correspond to a single firing of their CSDF counterparts, b and c. This correspondence allows us to derive a schedule for the CSDF graph, by an appropriate sampling of the eigenschedule of the approximation. To illustrate this sampling in Figure 5.8(a), firings that correspond to firings in the CSDF graph are highlighted. If we now translate the schedule such that the first firing of CSDF actor a starts at time zero, we obtain the schedule depicted in Figure 5.8(b).



(a) Token transfer schedule, obtained from the pessimistic approximation. Colours indicate the corresponding producing actor, squares indicate tokens that mark the completion of a firing.



(b) Schedule for the CSDF graph, obtained from the token transfer schedule.

Figure 5.9 – Construction of an admissible schedule for the CSDF graph of Figure 5.5, from the strictly periodic token transfer eigenschedule computed from its pessimistic single-rate approximation (shown in Figure 5.7(b)).

For the token transfer perspective, the construction of a schedule for the CSDF graph is slightly more involved. As a starting point, we again take the eigenschedule of the pessimistic approximation (shown in Figure 5.7(b)), and sample it in such a way, that we obtain a schedule for the transfer of tokens in the CSDF graph. Now, the sampling is given by the graph's flow normalisation vector, s: every nine consecutive firings of actor $\hat{aa}$ in Figure 5.7(b) correspond to the transfer of a single token over CSDF channel aa in Figure 5.5. For actors $\hat{ab}$, $\hat{ba}$, $\hat{bc}$ and $\hat{cb}$, this number is, respectively, six, six, four and four. The sampled eigenschedule is shown in Figure 5.9(a), where each dot indicates the scheduled transfer of a single token, and colours indicate the producing actor.

In the CSDF graph, a single actor firing involves the (production and) transfer of multiple tokens over outgoing channels. As such, we map the token transfer schedule of Figure 5.9(a) to a schedule for actor firings, by letting the completion time of a firing coincide with the latest transfer of the tokens it produces. For example, the second firing of CSDF actor a produces two tokens onto channel ab, and one token onto channel aa. From Figure 5.9(a), we derive that the last of the two tokens, produced onto ab, is transferred at $t = 21$, and the token produced onto aa is transferred at $t = 16.5$. Hence, we let the second firing of a complete at the maximum of these two timestamps, at $t = 21$. Applying this reasoning to the full eigenschedule yields the schedule for the CSDF, depicted in Figure 5.9(b).

We now formalise the illustrated relation between the schedules of a CSDF graph and its pessimistic approximation. In the remainder of this section, we shall prove that the pessimistic approximations are *temporal abstractions* of the CSDF graph, and that the latter is a temporal abstraction of the optimistic approximation. Furthermore, in the spirit of Chapter 4, where we showed that a mapping exists between the schedules of a CSDF graph and its unfolding, we describe and prove a relation between the schedules of a CSDF graph and its four different single-rate approximations. Let $\mathcal{G}$ be a consistent CSDF graph, with m source actors $u_1, \ldots, u_m$, and p sink actors $y_1, \ldots, y_p$.

The actor firing perspective

Any schedule that is admissible for the pessimistic approximation may be mapped to a schedule that is admissible for the approximated CSDF graph. This mapping follows from the correspondence between the firings of the CSDF graph and its single-rate approximation, as illustrated in the example at the beginning of this section. The following lemma proves the validity of this mapping.

Lemma 5.1 (Schedule mapping). *Let $\mathcal{G}$ be a consistent CSDF graph, and $\mathcal{H}$ be the pessimistic single-rate approximation of $\mathcal{G}$, derived from the actor firing perspective of $\mathcal{G}$, using Algorithm 6. Furthermore, let t be an admissible schedule for $\mathcal{H}$, and s a schedule for $\mathcal{G}$, which is defined in terms of t, for every actor v in $\mathcal{G}$, as follows:*

$$s_v(k) = t_{\hat{v}}\left(k\frac{\mathcal{N}_\mathcal{G}}{q_v}\right), \tag{5.18}$$

for all $k \in \mathbb{Z}$. Schedule s is admissible for $\mathcal{G}$.

Proof. By Definition 3.4, schedule s is admissible, if the following two conditions hold:

(i) $k \geq m \Rightarrow s_v(k) \geq s_v(m)$.

(ii) $\pi_{vw}(k) \geq m \Rightarrow s_w(k) \geq s_v(m) \otimes \tau_w(k)$.

We derive condition (i) from the admissibility of t:

$$k \geq m \Rightarrow t_{\hat{v}}\left(k\frac{\mathcal{N}_\mathcal{G}}{q_v}\right) \geq t_{\hat{v}}\left(m\frac{\mathcal{N}_\mathcal{G}}{q_v}\right) \Rightarrow s_v(k) \geq s_v(m).$$

This leaves condition (ii). By Proposition 5.1, the former implies:

$$\pi_{vw}(k) \geq m \Rightarrow \pi_{\hat{v}\hat{w}}\left(k\frac{\mathcal{N}_\mathcal{G}}{q_w}\right) \geq m\frac{\mathcal{N}_\mathcal{G}}{q_v}.$$

Using the fact that t is admissible, we may now derive the following conclusion (observe that $\tau_{\hat{w}} \geq \tau_w(k)$, for all $k \in \mathbb{Z}$):

$$t_{\hat{w}}\left(k\frac{\mathcal{N}_\mathcal{G}}{q_w}\right) \geq t_{\hat{v}}\left(m\frac{\mathcal{N}_\mathcal{G}}{q_v}\right) \otimes \tau_{\hat{w}} \Rightarrow s_w(k) \geq s_v(m) \otimes \tau_w(k).$$

This proves that (ii) holds, which completes the proof. □

The optimistic single-rate approximation has a temporal dynamics that is never worse than that of the CSDF graph. This implies that a relation between the schedules of the former and the latter exists, similar to the relation between the schedules of the pessimistic approximation and the CSDF graph, given by Lemma 5.1. Indeed, every schedule that is admissible for the CSDF graph may be mapped to one that is admissible for the optimistic approximation. This mapping is such that a single firing in the former is mapped to multiple, parallel executing, firings in the latter. We characterise and prove the mapping in the following lemma.

Lemma 5.2 (Optimistic schedule mapping). *Let $\mathcal{G}$ be a consistent CSDF graph, and $\mathcal{H}$ be the optimistic single-rate approximation of $\mathcal{G}$, derived from the actor firing perspective of $\mathcal{G}$, using Algorithm 6. Furthermore, let t be an admissible schedule for $\mathcal{G}$. Let s be a schedule for $\mathcal{H}$, defined in terms of t, for every actor $\check{v}$ in $\mathcal{H}$, as follows:*

$$s_{\check{v}}(k) = t_v\left(\left\lceil \frac{kq_v}{\mathcal{N}_\mathcal{G}} \right\rceil\right), \tag{5.19}$$

for all $k \in \mathbb{Z}$. Schedule s is admissible for $\mathcal{H}$.

Proof. The temporal ordering of firings, of the same actor, in graph $\mathcal{H}$ satisfies:

$$k \geq m \Rightarrow t_v\left(\left\lceil \frac{kq_v}{\mathcal{N}_\mathcal{G}} \right\rceil\right) \geq t_v\left(\left\lceil \frac{mq_v}{\mathcal{N}_\mathcal{G}} \right\rceil\right) \Rightarrow s_{\check{v}}(k) \geq s_{\check{v}}(m). \tag{5.20}$$

From (5.11), and by the linearity of $\check{\gamma}$, we have:

$$\frac{q_v}{\mathcal{N}_\mathcal{G}} \pi_{\check{v}\check{w}}(k) = \check{\gamma}_{vw}\left(k \frac{q_w}{\mathcal{N}_\mathcal{G}}\right) \leq \check{\gamma}_{vw}\left(\left\lceil \frac{kq_w}{\mathcal{N}_\mathcal{G}} \right\rceil\right) \leq \pi_{vw}\left(\left\lceil \frac{kq_w}{\mathcal{N}_\mathcal{G}} \right\rceil\right).$$

This allows us to derive the following relation between the predecessor functions of $\mathcal{G}$ and $\mathcal{H}$:

$$\pi_{\check{v}\check{w}}(k) \geq m \Rightarrow \pi_{vw}\left(\left\lceil \frac{kq_w}{\mathcal{N}_\mathcal{G}} \right\rceil\right) \geq m \frac{q_v}{\mathcal{N}_\mathcal{G}} \overset{\text{(A.4)}}{\Rightarrow} \pi_{vw}\left(\left\lceil \frac{kq_w}{\mathcal{N}_\mathcal{G}} \right\rceil\right) \geq \left\lceil \frac{mq_v}{\mathcal{N}_\mathcal{G}} \right\rceil,$$

Since t is admissible for $\mathcal{G}$, this implies, by Definition 3.4, the following:

$$t_w\left(\left\lceil \frac{kq_w}{\mathcal{N}_\mathcal{G}} \right\rceil\right) \geq t_v\left(\left\lceil \frac{mq_v}{\mathcal{N}_\mathcal{G}} \right\rceil\right) \otimes \tau_w\left(\left\lceil \frac{kq_w}{\mathcal{N}_\mathcal{G}} \right\rceil\right) \Rightarrow s_{\check{w}}(k) \geq s_{\check{v}}(m) \otimes \tau_{\check{w}},$$

which, with (5.20) and by Definition 3.4, states that schedule s is admissible. This completes the proof. □

The token transfer perspective

As we have seen in the example that we discussed in the beginning of this section, a schedule for token transfers, derived from the pessimistic single-rate approximation, may be mapped to a schedule for actor firings. In the remainder of this section, we formalise and prove this mapping. The following lemma proves that any schedule that is admissible for the pessimistic token transfer approximation can be mapped to an admissible schedule for the CSDF graph.

Lemma 5.3 (Pessimistic schedule mapping - token transfers). *Let $\mathcal{G}$ be a consistent CSDF graph, and $\mathcal{H}_{pess}$ be the pessimistic single-rate approximation of $\mathcal{G}$, derived from $\mathcal{G}$ using Algorithm 7. Furthermore, let t be an admissible schedule for $\mathcal{H}_{pess}$. Let z be a token transfer schedule for $\mathcal{G}$, defined in terms of t, for every channel vw in $\mathcal{G}$, as follows:*

$$z^{\bullet}_{vw}(k) = t_{\hat{v}\check{w}}\left(s_{vw}k\right), \tag{5.21}$$

for all $k \in \mathbb{Z}$. Schedule z is admissible for $\mathcal{G}$.

Proof. Due to the admissibility of t, we have:

$$k \geq m \Rightarrow t_{\hat{v}\check{w}}\left(s_{vw}k\right) \geq t_{\hat{v}\check{w}}\left(s_{vw}m\right) \Rightarrow z^{\bullet}_{vw}(k) \geq z^{\bullet}_{vw}(m), \tag{5.22}$$

and, by Proposition 5.2 and admissibility of t, we have:

$$\begin{aligned} \pi^{\bullet}_{uvw}(k) \geq m &\Rightarrow \pi_{\hat{u}\check{v}\hat{v}\check{w}}(s_{vw}k) \geq s_{uv}m \\ &\Rightarrow t_{\hat{v}\check{w}}(s_{vw}k) \geq t_{\hat{u}\check{v}}(s_{uv}m) \otimes \max_{1 \leq i \leq \varphi_v} \tau_v(i) \\ &\Rightarrow z^{\bullet}_{vw}(k) \geq z^{\bullet}_{uv}(m) \otimes \tau_v(k). \end{aligned} \tag{5.23}$$

By Definition 3.4, (5.22) and (5.23) together imply that z is admissible for $\mathcal{H}$, which completes the proof. □

In line with Lemma 5.2, which gives a mapping from a schedule for the optimistic approximation, to a schedule for the CSDF graph, we give such a mapping for the token transfer perspective, in the following lemma. This completes the mapping between schedules of the four different single-rate approximations and the CSDF graph.

Lemma 5.4 (Optimistic schedule mapping - token transfers). *Let $\mathcal{G}$ be a consistent CSDF graph, and $\mathcal{H}$ be the optimistic single-rate approximation of $\mathcal{G}$, derived from $\mathcal{G}$, using Algorithm 7. Furthermore, let $t^{\bullet}$ be an admissible token transfer schedule for $\mathcal{G}$. Let z be a token transfer schedule for $\mathcal{H}$, defined in terms of t, for every actor $\hat{v}\check{w}$ in $\mathcal{H}$, as follows:*

$$z_{\hat{v}\check{w}}(k) = t^{\bullet}_{vw}\left(\left\lceil \frac{k}{s_{vw}} \right\rceil\right), \tag{5.24}$$

for all $k \in \mathbb{Z}$. Schedule z is admissible for $\mathcal{H}$.

Proof. The proof is analogous to the proof of Lemma 5.2. □

Given the mappings between schedules of a CSDF graph and its single-rate approximations, we may derive a relation between the throughputs of the graphs. Recall that the throughput of a graph is attained by a self-timed schedule, and the throughput of a graph is related to the throughput of an actor, through a scaling by the actor's repetition vector entry. Using this relation, and the relation between the admissible schedules of the graphs, as given by the previous four lemmas, we can derive bounds on the throughput of the approximated CSDF graph, as stated by the following theorem.

Theorem 5.1 (Bounds on throughput). *Let $\mathcal{G}$ be a consistent CSDF graph, with scalar invariant $\mathcal{N}_{\mathcal{G}}$. Furthermore, let $\mathcal{H}_{opt}$, $\mathcal{H}_{pess}$, $\mathcal{H}^{\bullet}_{opt}$ and $\mathcal{H}^{\bullet}_{pess}$ be the optimistic and pessimistic single-rate approximations of $\mathcal{G}$, obtained by applying, respectively, Algorithm 6 and Algorithm 7. Let $Th(G)$ denote the throughput of graph G. The throughputs of $\mathcal{H}_{pess}$ and $\mathcal{H}^{\bullet}_{pess}$ bound the throughput of $\mathcal{G}$, in the following way:*

$$\max\left\{Th(\mathcal{H}_{pess}), Th(\mathcal{H}^{\bullet}_{pess})\right\} \le \mathcal{N}_{\mathcal{G}} Th(\mathcal{G}) \le \min\left\{Th(\mathcal{H}_{opt}), Th(\mathcal{H}^{\bullet}_{opt})\right\}.$$

Proof. This follows from lemmas 5.1, 5.2, 5.3, and 5.4. In the following, let q and s be the repetition and flow normalisation vector of $\mathcal{G}$. Any admissible schedule for $\mathcal{H}_{\text{pess}}$ can be mapped to a schedule that is admissible for $\mathcal{G}$. Let t be a self-timed schedule of $\mathcal{H}_{\text{pess}}$, which is mapped through (5.18) to schedule s for $\mathcal{G}$. In t, actor $\hat{v}$ fires at an average rate of $Th(\mathcal{H}_{\text{pess}})$, and in s, corresponding actor v fires at a rate of $Th(\mathcal{H})\frac{q_v}{\mathcal{N}_{\mathcal{G}}}$. This means that, in s, the number of graph iterations of $\mathcal{G}$ that are completed per time unit, is equal to $\frac{Th(\mathcal{H}_{\text{pess}})}{\mathcal{N}_{\mathcal{G}}}$. Actors in $\mathcal{G}$ may fire at a higher rater than schedule s prescribes. The rate at which actors fire in schedule s thus gives a *lower bound* on the throughput of $\mathcal{G}$. In other words, we obtain, from the mapping of schedule t to schedule s, that $\mathcal{N}_{\mathcal{G}} Th(\mathcal{G}) \ge Th(\mathcal{H}_{\text{pess}})$. Following a similar reasoning, we may derive that the optimistic approximation, $\mathcal{H}_{\text{opt}}$, gives an upper bound on the throughput of $\mathcal{G}$.

Next, consider an admissible schedule $t^{\bullet}$ for $\mathcal{H}^{\bullet}_{\text{pess}}$, which is mapped to an admissible schedule s for $\mathcal{G}$, through (5.21). In $t^{\bullet}$, actor $\hat{vw}$ fire at an average rate of $Th(\mathcal{H}^{\bullet}_{\text{pess}})$. This corresponds to an average rate at which tokens are produced onto channel vw in $\mathcal{G}$ of, on average, $\frac{Th(\mathcal{H}^{\bullet}_{\text{pess}})}{s_{vw}}$. The average number of tokens produced, per firing of actor v, onto vw, is given by $\frac{P^{\Sigma+}_{vw}}{\varphi_v}$. Thus, the average rate at which actor v must fire, in s, is $\frac{Th(\mathcal{H}^{\bullet}_{\text{pess}})\varphi_v}{s_{vw}P^{\Sigma+}_{vw}}$. If we divide this by the repetition vector entry of v, we obtain the average number of graph iterations of $\mathcal{G}$, completed per time unit, which is equal to: $\frac{Th(\mathcal{H}^{\bullet}_{\text{pess}})}{\mathcal{N}_{\mathcal{G}}}$. This rate gives a lower bound on the throughput of $\mathcal{G}$. An upper bound on the throughput of $\mathcal{G}$ may be obtained from $\mathcal{H}^{\bullet}_{\text{opt}}$ by following a similar reasoning.

From the above, we have two upper bounds and two lower bounds. By taking the minimum of the former and the maximum of the latter we obtain the stated theorem. □

The pessimistic bound on throughput is useful in the sense that, given that the execution times of actor firings are conservative, the bound gives a guarantee on the performance of the system that is modelled by the approximated CSDF graph. In the modelled system, these conservative execution times may never be attained. This means that the optimistic bound pertains to the model (i.e., the CSDF graph), not to the modelled system: the bound gives a lower bound on performance, given that actor firings take as long as their associated execution times specify. In this sense, the optimistic bound primarily gives an indication of the *error* of the estimated throughput; if this error is (too) large, an exact transformation into a larger graph (see Chapter 4), which exposes interactions between individual phases of the CSDF actors, may give a more accurate estimation (see Chapter 6).

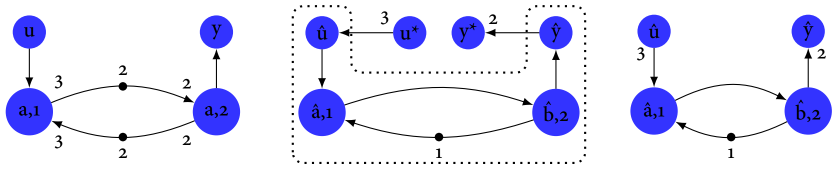
5.3.3 Constructing temporal abstractions

The pessimistic and optimistic single-rate approximations provide bounds on the performance of the approximated CSDF graph. This implies a *temporal abstraction* relation (see Section 3.3). However, a temporal abstraction relation between two graphs implies that, at an input rate that is low enough, the output of the two graphs have the same rate. Such is not the case for the single-rate approximations of a CSDF graph: as the mapping between the admissible schedules of the graphs has indicated, output actors in the former fire at a higher rate than those in the latter. Consequently, by definition, a temporal abstraction relation between the approximations and the CSDF graph does not hold. We may, however, obtain a temporal abstraction, by appropriately setting rates, on channels connecting source actors, which we refer to as *input channels*, and channels connecting sink actors, which we refer to as *output channels*. Here, consumption rates (on output channels) serve to sample the output, and production rates (on input channels) increase the rate at which input is fed into the system. These rates must be such that they reflect the relations, between the input and outputs of the CSDF graph and its single-rate approximations, as given in the following lemma.

Lemma 5.5. *Let $\mathcal{G}$ be a consistent CSDF graph, and $\mathcal{H}_{opt}$ and $\mathcal{H}_{pess}$ be the optimistic and pessimistic single-rate approximation of $\mathcal{G}$, derived using Algorithm 6. Furthermore, let u and y be the input and output of the actor firing perspective of $\mathcal{G}$, and $\hat{u}$ (resp. $\check{u}$) and $\hat{y}$ ($\check{y}$) the corresponding input and output to the actor firing perspective of $\mathcal{H}_{pess}$ ($\mathcal{H}_{opt}$). The following relations hold, for all $k \in \mathbb{Z}$:*

$$u(k) = \hat{u}\left(k\frac{\mathcal{N}_{\mathcal{G}}}{q_u}\right) \Rightarrow y(k) \le \hat{y}\left(k\frac{\mathcal{N}_{\mathcal{G}}}{q_y}\right), \tag{5.25a}$$

$$\check{u}(k) = u\left(\left\lceil \frac{kq_u}{\mathcal{N}_{\mathcal{G}}}\right\rceil\right) \Rightarrow \check{y}(k) \le y\left(\left\lceil \frac{kq_y}{\mathcal{N}_{\mathcal{G}}}\right\rceil\right). \tag{5.25b}$$



(a) MRSDF graph, $\mathcal{N}$ = 6. (b) Temporal abstraction. (c) Simplified.

Figure 5.10 – An MRSDF graph and its single-rate approximation (indicated by the dotted line), extended with channels such that the graph forms a temporal abstraction.

Proof. This follows from the mappings between the admissible schedules of the three graphs, given by lemmas 5.2 and 5.1. Any *self-timed* schedule t for $\mathcal{H}_{\text{pess}}$ is mapped to an admissible schedule s for $\mathcal{G}$. Schedule s may not be a self-timed schedule: some firings are not scheduled to occur as soon as they are enabled, but rather somewhat later. This means that an output $\hat{y}(k)$ of the system associated with $\mathcal{H}_{\text{pess}}$, in response to the mapped input of $\mathcal{G}$, occurs no later than $s_{\hat{y}}(k)$. Consequently, (5.25a) holds. The fact that (5.25b) holds is derived following a similar reasoning. □

The above lemma implies that the pessimistic single-rate approximation, obtained by Algorithm 6, becomes a temporal abstraction of the approximated CSDF graph, if we transform its inputs and outputs, in the manner given by (5.25a) and (5.25b). This transformation is analogous to a change in the rates associated with source and sink actors in the single-rate approximation. To illustrate this, consider Figure 5.10. The graph shown in Figure 5.10(b) is a supergraph of the pessimistic single-rate approximation of the graph shown in Figure 5.10(a). The subgraph corresponding to the approximation is indicated by the dotted line. The graph's sink actor, y*, essentially samples the output, ŷ, of the single-rate approximation, and a single firing of the graph's source actor, u*, is mapped to three firings of actor û of the single-rate approximation. As a result, the output of the graph of Figure 5.10(b) is given by $y^*(k) = \hat{y}(2k)$, where $\hat{y}$ is the output of the pessimistic approximation of the MRSDF graph. The input of the pessimistic approximation is now given by $\hat{u}(k) = u^*\left(\left\lceil \frac{k}{3} \right\rceil\right)$. By Lemma 5.5 (note that the MRSDF graph has $\mathcal{N} = 6$, $q_{\text{a}} = 2$ and $q_{\text{b}} = 3$), the graph of Figure 5.10(b) is a temporal abstraction of the graph of Figure 5.10(a).

The graph of Figure 5.10(b) may be further simplified. Observe that, since actors û and ŷ have an execution time of *zero*, we may replace the path u*ûâ by a single channel that has a production rate of three, and a consumption rate of one. Similarly, we may replace the path b̂ŷy* with a path that has a production rate of one, and a consumption rate of two. These replacements results in the graph shown in Figure 5.10(c). In a similar fashion, we may apply the same reasoning to the optimistic approximation of the graph of Figure 5.10(a), to obtain a graph that is *temporally abstracted* by it. The following theorem formalises the illustrated transformation of the single-rate approximations of a graph, and their temporal interrelationships.

Theorem 5.2 (Temporal abstraction relation - actor firings). *Let $\mathcal{G}$ be a consistent CSDF graph, and $\mathcal{H}_{opt}$ and $\mathcal{H}_{pess}$ be the optimistic and pessimistic single-rate approximation of $\mathcal{G}$, derived from the actor firing perspective of $\mathcal{G}$. Let $\mathcal{H}^*_{pess}$ be the graph obtained from $\mathcal{H}_{pess}$ by setting the* production rate *of each channel $\hat{v}\hat{w}$, where $\hat{v}$ is a source, to $\frac{\mathcal{N}_\mathcal{G}}{q_v}$, and the consumption rate of each channel $\hat{u}\hat{v}$, where $\hat{v}$ is a sink, to $\frac{\mathcal{N}_\mathcal{G}}{q_v}$. Likewise, let $\mathcal{H}^*_{opt}$ be the graph obtained from $\mathcal{H}_{opt}$ in an equivalent way. Graph $\mathcal{H}^*_{pess}$ is a temporal abstraction of $\mathcal{G}$, and the latter is a temporal abstraction of $\mathcal{H}^*_{opt}$. Formally:*

$$\mathcal{H}^*_{pess} \sqsupseteq_T \mathcal{G} \sqsupseteq_T \mathcal{H}^*_{opt}.$$

Proof. This follows from Lemma 5.5. By appropriate substitutions, of k, in the conclusion of (5.25b) and the premise of (5.25a), we obtain:

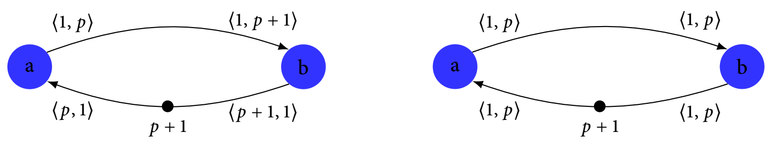
$$\check{u}(k) = \hat{u}(k) = u\left(\left\lceil \frac{kq_u}{\mathcal{N}_\mathcal{G}} \right\rceil\right) \Rightarrow \hat{y}\left(k\frac{\mathcal{N}_\mathcal{G}}{q_y}\right) \geq y(k) \geq \check{y}\left(k\frac{\mathcal{N}_\mathcal{G}}{q_y}\right),$$

which, by Definition 3.9, proves the theorem. □

5.3.4 Comparing the two perspectives

Consider the single-rate approximations (Figures Figure 5.6 and Figure 5.7) of the CSDF graph depicted earlier in Figure 5.5, as obtained from each of the two perspectives. The optimistic *throughput* approximations they provide are the same, but the pessimistic approximations differ; the bounds on throughput obtained from the token transfer perspective are tighter than those obtained from the actor firing perspective. An obvious question is whether this holds in general. The answer to this question is negative, which we illustrate with an example comparison. Rather than exploring the accuracy of the two perspectives in full, we highlight the difference in accuracy for two graphs that are small enough to serve as intuitive examples, and different enough to give widely varying outcomes for each of the perspectives.

These two graphs are depicted in Figure 5.11. For the first graph, shown in Figure 5.11(a), the pessimistic approximation derived from the actor firing perspective has $2 - 2p$ tokens on channel ab, and 3 tokens on channel ba. As a result, cycle aba has a total of $5 - 2p$ tokens, and its throughput is estimated to be negative. If we approximate using the token transfer perspective, we find that the cycle has 4 tokens, regardless of the value of p. This means that increasing p decreases the accuracy of the throughput estimation, obtained from the actor firing perspective, whereas the accuracy of the token transfer perspective is unaffected.



(a) Poor results in the actor firing perspective.

(b) Poor results in the token transfer perspective.

Figure 5.11 – Two CSDF graphs that show two extremes for the approximation error in the actor firing and token transfer perspective.

For the other graph, depicted in Figure 5.11(b), we find the opposite: if we choose $p = 1$, then both single-rate approximations have a total of 2 tokens on the cycle. Increasing p monotonically decreases the number of tokens in the pessimistic approximation; for $p = 8$ the pessimistic approximation has -10 tokens, whereas the actor firing perspective has 18 tokens. This means that for this graph, the token transfer gives a worse accuracy than the actor firing perspective. Again, the difference in approximation accuracy increases, as p increases.

When comparing the two perspectives for the approximations they deliver, there is no clear winner, as illustrated by these two examples. There are cases where one gives tighter bounds than the other, and vice versa. This is true for CSDF graphs. For MRSDF graphs, however, the conclusion is different. To see this, consider the predecessor functions of the pessimistic single-rate approximations for an MRSDF graph, for respectively the token transfer and actor firing perspectives:

$$\begin{aligned}\hat{\pi}^{\bullet}_{uvw}(k) &= k - s_{vw} - s_{uv}\delta_{uv} + s_{uv}\rho^{-}_{uv} \\ &\overset{(2.2)}{=} k - s_{vw}(1 - \rho^{+}_{vw}) - s_{uv}\delta_{uv}.\end{aligned} \tag{5.26}$$

$$\hat{\pi}_{vw}(k) = k - s_{vw}\left(g_{vw} - \rho^{+}_{vw} + g_{vw}\left\lfloor \frac{\delta_{vw} + g_{vw}}{g_{vw}} \right\rfloor\right). \tag{5.27}$$

These two functions are similar in case $g_{vw} = 1$. For this case, the bounds on throughput provided by the two functions are the same. In case $g_{vw} > 1$, the bounds obtained from the actor firing perspective are never worse than those obtained from the token transfer perspective. This is stated and proven by the following theorem.

Theorem 5.3 (Actor firing perspective for MRSDF). *Let $\mathcal{G}$ be an MRSDF graph, and $\mathcal{H}$ and $\mathcal{H}^{\bullet}$ its pessimistic single-rate approximations derived from, respectively, the actor firing and token transfer perspective of $\mathcal{G}$.*

Graph $\mathcal{H}$ gives a tighter lower bound on throughput than $\mathcal{H}^{\bullet}$.

Proof. First note that there is a natural correspondence between the cycles of $\mathcal{H}$ and $\mathcal{H}^\bullet$; a cycle $v_1, \dots v_n$ in $\mathcal{H}$ corresponds to a cycle $v_1v_2, v_2v_3, \dots, v_{n-1}v_n$ in $\mathcal{H}^\bullet$. With this correspondence, we can compare the maximum cycle ratios of $\mathcal{H}$ and $\mathcal{H}^\bullet$. A higher maximum cycle ratio means a lower throughput.

Consider a cycle $C = v_1, \dots, v_n$ $\mathcal{H}$, and the corresponding cycle $C^\bullet$ in $\mathcal{H}^\bullet$. The cycle ratio of $C^\bullet$, denoted $\lambda(C^\bullet)$, is given by

$$\lambda(C^\bullet) = \frac{\sum_{uvvw \in C} \tau_v}{\sum_{uvvw \in C} s_{vw}(\delta_{vw} + 1 - \rho^+_{vw})}.$$

For the cycle ratio of C, we derive an upper bound:

$$\lambda(C) = \frac{\sum_{vw \in C} \tau_v}{\sum_{vw \in C} s_{vw}\left(g_{vw}\left\lfloor \frac{\delta_{vw}+g_{vw}}{g_{vw}} \right\rfloor - \rho^+_{vw}\right)} \leq \frac{\sum_{vw \in C} \tau_v}{\sum_{vw \in C} s_{vw}\left(\delta_{vw} + 1 - \rho^+_{vw}\right)}.$$

With the natural correspondence given above, we have $\lambda(C) \leq \lambda(C^\bullet)$ for any pair of corresponding cycles C and $C^\bullet$ in $\mathcal{H}$ and $\mathcal{H}^\bullet$. In particular, this relation holds for the cycle that attains maximum cycle ratio. In other words, the maximum cycle ratio of $\mathcal{H}$ is at most the maximum cycle ratio of $\mathcal{H}^\bullet$. Since the throughput of an HSDF graph is the inverse of the graph's maximum cycle ratio, the throughput of $\mathcal{H}$ is at least the throughput of $\mathcal{H}^\bullet$. By Theorem 5.1, the throughput of $\mathcal{G}$ is bounded from below by the maximum of the throughputs of $\mathcal{H}$ and $\mathcal{H}^\bullet$. Thus, the stated theorem is true. □

The accuracy offered by each of the perspectives is just one aspect to consider when deciding which one to use. Another is the size of the single-rate approximations they yield. A single-rate approximation obtained from the actor firing perspective has the same number of actors and channels as the original graph has. The token transfer perspective, however, leads to a larger single-rate approximation, with an actor for every channel in the CSDF graph. As a result, the size of these graphs is in the order of the square of the size of the CSDF graph. This has an effect on the efficiency of analyses applied to the single-rate approximation. For example, computation of the maximum cycle ratio of the resulting graphs has a time complexity that is quadratic in the number of actors in the graph, and linear in the number of channels. This means that the *potentially* better accuracy of the token transfer perspective must be weighed against the increase in runtime of its analysis.

Any CSDF graph can be transformed into an equivalent MRSDF graph, and, by Theorem 5.3, for MRSDF graphs, the actor firing perspective never yields a worse conservative bound than the token transfer perspective does. Furthermore, considering the sizes of the single-rate approximations, the actor firing perspective scales better than the token transfer perspective. In the remainder of this thesis we therefore restrict our attention to the actor firing perspective.

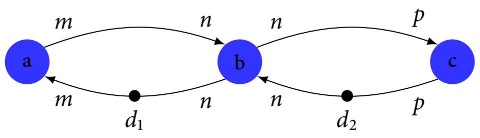


Figure 5.12 – The error in the estimated throughput of the graph depends on the graph's structural invariants and the number of tokens in the graph's critical subgraph. Here, d_2 is much larger than d_1; as a result, cycle aba is the bottleneck of the graph.

5.4 Quality of the approximation

The simplification of the analysis of a CSDF graph, through its single-rate approximation, comes at the expense of a loss in accuracy with respect to the temporal dynamics of the former graph. For example, schedules derived from the pessimistic approximation may not attain the throughput of the CSDF graph. The error in throughput depends for one part on the execution time vectors of actors; if the maximum of a vector is much larger than most of its entries, then the conservatively estimated throughput will be rather pessimistic. Another factor that the estimated throughput depends on consists of the rates and tokens on the graph's channels. To explore this relationship, we consider an example that is simple enough to be explored in detail, yet sufficient to understand how rates and tokens affect the estimated throughput.

In Figure 5.12, a simple MRSDF graph is shown, with execution times, rates and tokens left unspecified. If we take $p = 1$, then the repetition vector entries for actors a, b and c are respectively n, m, and mn. The normalisation vector entries are all one, and the scalar invariant $\mathcal{N}$ for the graph is equal to the *least common multiple* of m and n: $\mathcal{N} = \operatorname{lcm}(m, n)$. The estimated period (i.e., the inverse of its throughput) λ_{aba} of the leftmost cycle is bounded as follows:

$$\frac{\operatorname{lcm}(m,n)(\tau_{\mathsf{a}} + \tau_{\mathsf{b}})}{d_1 - m - n + 2\gcd(m,n)} \leq \lambda_{\mathsf{aba}} \leq \frac{\operatorname{lcm}(m,n)(\tau_{\mathsf{a}} + \tau_{\mathsf{b}})}{d_1}.$$

If we now choose p such that p is relatively prime to the product mn, then the repetition vector entries for a, b and c are pn, pm, and pmn, and $\mathcal{N} = p\operatorname{lcm}(m,n)$. This means that, for p such that $\gcd(p, mn) = 1$, the difference between the bounds on the throughput of cycle aba is p times the difference that is obtained when considering the cycle in isolation.

To summarise, the error in the throughput, computed from the bounds provided by the optimistic and pessimistic approximations, depends both on the structural invariants of the graph and the rates and number of tokens in the cycle. As a result, the accuracy of an initial estimation of a graph's throughput may be increased by considering the cycle in isolation. This, however, implicitly assumes that the throughputs of cycles are *composable*, which, in general, is not true. In Chapter 6, we present an approach that tackles this problem.

5.5 Discussion

Single-rate approximations of CSDF graphs have an important practical application: they allow for conservative analyses of the performance characteristics of a system, avoiding the computational costs of analysing the system's *equivalent* shift-invariant system. As such, many approaches to approximate the temporal dynamics of CSDF graphs have been presented in literature.

One of the most straightforward transformations of a CSDF graph into an MRSDF graph is obtained by replacing each actor by a so-called "lumped" actor, replacing each vector by the sum of its elements [45, 88]. The benefit of this transformation is that it simplifies the analysis, while still giving guarantees. It is obvious that this transformation allows for conservative analyses; the transformation sacrifices the scheduling freedom of the individual phases of a CSDF actor for simpler analysis. However, the approximation error made by the lumped representation may be considerable.

The predominant approach to approximate analysis involves the assumption of so-called strict periodicity. Under this assumption, the time between either two subsequent firings of an actor [5, 14], or between the production of two tokens onto a channel [38, 48, 98], is assumed to be constant. To the best of our knowledge, it is not known how these two subtly different approaches differ in accuracy. In this chapter, these two approaches are developed from the two different characterisations of the temporal dynamics of a CSDF graph that were introduced in Chapter 3: Section 5.3 shows how the actor firing and token transfer perspectives give rise to related but different single-rate approximations. In Section 5.3.4 we show that, for CSDF graphs, neither of the two approaches is to be preferred over the other. For the approximation of MRSDF graphs, though, interrelating actor firings is the preferred approach, as the systems it yields not only give tighter bounds on throughput, but are also smaller than those obtained when interrelating the transfer of individual tokens.

A disadvantage of existing approximate analyses is their restriction to the computation of a *single*, conservative, bound on throughput. Although such a conservative bound allows one to give guarantees on the performance of the modelled system, it leaves the error of the approximation unknown. The single-rate approximations described in this chapter allow for the computation of both a conservative and an optimistic bound. This gives a designer the ability to assess the approximation error, by taking the difference between these bounds.

A further contribution of this thesis over existing work is that our approximation of a CSDF graph is an HSDF graph, which may be analysed using existing, well-known efficient techniques. This is in contrast to the approach followed, for example, in

[48], where a throughput-optimal schedule is computed by formulating and solving a linear program.

The presented approximations combine naturally with the graph transformations given in the previous chapter. Transforming a graph by unfolding several firings of an actor, followed by an approximation, gives more scheduling freedom to the unfolded actors. In schedules computed from these unfolded approximations, the scheduling period of an actor may span multiple firings (these schedules are called K-periodic schedules in [13]). As such, the schedules may better approximate those schedules that attain the graph's throughput. This gives rise to an incremental approach, where the bounds on throughput are tightened in an incremental fashion. The next chapter describes this approach in more detail.

6

Throughput analysis

Abstract – *The throughput of a synchronous dataflow graph gives the maximum rate at which actors in the graph may fire. This maximum rate is determined by the properties of cyclic dependencies in the graph. Whereas for* homogeneous *synchronous dataflow graphs the throughput of a graph can be computed from its simple cycles, this is not the case for multi-rate and cyclo-static dataflow graphs. Classically, throughput analysis requires a graph to be transformed into its single-rate equivalent. In this chapter, we present a method that avoids this costly transformation, by explicitly grouping parallel actor firings into a single firing. The presented method combines the transformations described in chapter 4, and the approximate methods presented in chapter 5.*

The throughput of a system is a measure of the maximum amount of work the system can do in a given time period. For a discrete event system, throughput refers to the maximum rate at which events may occur; this could refer to the production speed of parts in a manufacturing system, or the rate at which data is processed in a stream processing system.

For an SDF graph, throughput relates to the fastest (average) rate at which the graph may complete iterations; a single iteration consists of a set of actor firings, the size of which is given by the graph's repetition vector. The throughput of an SDF graph is limited by cyclic dependencies; a graph that has no cycles has an unlimited throughput, as the sources of such a graph do not depend on any other vertices they may simultaneously start an infinite number of firings.

When computing the throughput of a graph, we may restrict the analysis to its strongly connected components. Throughout this chapter, we assume that SDF

Large parts of this chapter have been published in [RdG:8] and [RdG:7].

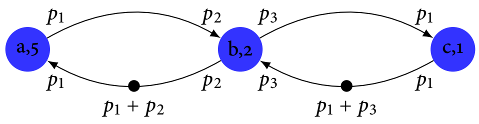


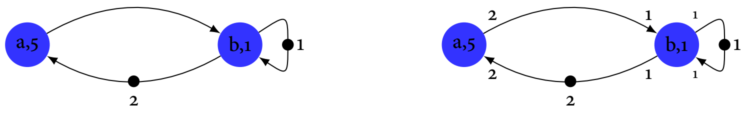
Figure 6.1 – The single-rate equivalent of the composition of two cycles has, in the best case, at least as many actors as the single-rate equivalents of the individual cycles have in total. In the worst case, however, its size is in the order of the product of the sizes of the smaller single-rate equivalents.

graphs are strongly connected. In case this condition does not hold one may compute the graph's throughput by composing the throughputs of the strongly connected components, in a straightforward way [40, 41, 49].

Throughput is tightly coupled with the *eigenvalue* associated with the state transition matrix of a max-plus system (see Chapter 2): the eigenvalue is given by the *maximum cycle ratio* of the system's graphical representation, and throughput is the (multiplicative) inverse of the graph's maximum cycle ratio. For HSDF graphs, throughput can be computed from the simple cycles in the graph [78]. For non-homogeneous SDF graphs, throughput may be computed from the graph's single-rate equivalent. This is a potentially expensive (in terms of computing time) approach to computing throughput, as the size of the single-rate equivalent grows exponentially in the size of the original graph [77].

For example, consider the graph depicted in Figure 6.1, which consists of two cycles, with rates given by p_1, p_2 and p_3. Assume that these rates are, pairwise, relatively prime. The sizes (number of actors) of the single-rate equivalents of the two cycles are then respectively $p_2 + p_1$ and $p_3 + p_1$. The single-rate equivalent of their composition has $(p_1 + p_2)p_3 + p_1 p_2$ actors, which is in the order of the product of the sizes of the individual single-rate equivalents.

In Section 6.2 we describe how the highly regular structure of a graph's single-rate equivalent may be exploited to optimise its analysis, by restricting the search for the graph's maximum cycle ratio to a potentially much smaller subgraph. In particular, the presented method allows for efficient throughput analysis of SDF cycles, avoiding a full expansion of the cycle into its single-rate equivalent. The improvement in efficiency one obtains using the approach is strongly dependent on the *structure* of the single-rate equivalent of the graph it is applied to. In the worst case, only a few channels in the single-rate equivalent are pruned by the analysis. We therefore introduce a second, novel, approach in Section 6.3. This approach is *incremental*: it selectively analyses cycles in the SDF graph that are estimated to be critical, and composes the schedules and throughputs of these cycles by making assumptions on which firings of an actor can be joined into parallel firing groups. The approach builds upon the various transformations and approximations presented in the previous two chapters.



(a) HSDF graph. (b) MRSDF graph.

Figure 6.2 – The throughput of the composition of two non-homogeneous cycles depends not only on the throughputs of the cycles considered in isolation, but on their parallelism as well.

We discuss and compare the two methods with each other and with related approaches, in Section 6.4.

6.1 Throughput, parallelism, and maximum cycle ratio

The throughput of a cycle in an HSDF graph is computed as the ratio between the number of tokens on the cycle, and the total execution time of actors on the cycle. As Reiter showed, the throughput of cycles in an HSDF graph composes: the throughput of the composition of the two cycles (i.e., the graph induced by the two cycles) is equal to the minimum of the throughputs of the cycles considered in isolation [78]. This is not true for MRSDF graphs. Whereas the throughput of an HSDF graph is fully determined by (one or more) *simple* cycles, the throughput of an MRSDF graph is attained by a *closed walk*.

We give a small example to illustrate this. Consider the two graphs depicted in Figure 6.2. The only difference between the graphs shown in Figure 6.2(a) and Figure 6.2(b) is the number of tokens that is consumed and produced by actor a. In the HSDF graph of Figure 6.2(a), maximum cycle ratio is equal to *three*, and is attained by cycle aba. Any admissible schedule for this graph forbids the parallel execution of two firings of actor b, due to the self-loop bb.

For the MRSDF graph of Figure 6.2(b), cycle aba attains maximum throughput only if the two tokens produced by firings of actor a are consumed by two *parallel* firings of actor b. This is, however, not possible, again due to the self-loop, which enforces strictly sequential scheduling of firings of b. Hence, the throughput of the MRSDF graph is given by neither of the two simple cycles, but rather by the non-simple cycle abba, which has a throughput of $^1/_7$: it involves two sequential executions of actor b, followed by one execution of actor a. In order to compute the throughput of the composition of cycles aba and bb, this restriction of parallelism must thus be taken into account.

The most straightforward way to deal with the non-trivial composition of MRSDF cycles, is to transform the MRSDF graph into its single-rate equivalent. Since the latter is an HSDF graph, its analysis is again straightforward, albeit that its size might be prohibitively large. In order to reduce the size, we may prune its actors and channels: when analysing the single-rate equivalent for its maximum cycle ratio, actors and channels that do not lie on a cycle in the graph may be pruned safely. Furthermore, actors that do not lie on a *critical* cycle (i.e., a cycle that attains maximum cycle ratio) in the graph may be pruned as well.

This is roughly the approach described in Section 6.2: rather than constructing and analysing the full single-rate equivalent, it optimises the search for the critical cycle by restricting it to a possibly small subgraph, by pruning actors and channels that do not lie on a cycle.

Section 6.3 describes a more effective means to reduce the size of the analysed graph. It may prune actors and channels that do lie on one of the cycles in the single-rate equivalent, but never prunes actors or channels that lie on a *critical cycle*. This approach involves joining actors in the single-rate equivalent, forming parallel executing groups. This gives an iterative approach, where assumptions are made, verified, and possibly revised. As a result, only a small part of the single-rate equivalent needs to be constructed. Furthermore, as the approach uses the approximations described in the previous chapter, it provides bounds on the graph's throughput that are tightened in an iterative fashion.

6.2 Analysis of the single-rate equivalent

The throughput of an MRSDF graph is equal to the reciprocal of the maximum cycle ratio of its single-rate equivalent. There are several efficient algorithms that compute the maximum cycle ratio in time polynomial in the size of the graph [29]. However, when applied to the single-rate equivalent of an MRSDF graph, the main bottleneck in the analysis is not the efficiency of the algorithm, but rather the graph's size, which is exponential in the size of the MRSDF graph. Therefore, *pruning* the single-rate equivalent, which discards channels and actors that do not impact the graph's throughput, has a strong potential to speed up the analysis. In this section we show that many channels may be pruned from a graph's single-rate equivalent. As a result, the analysis may be restricted to a, typically much smaller, subgraph.

In the following sections, we formalise the structure of single-rate equivalents of MRSDF graphs of increasing complexity. This allows one to reason about redundant channels, paths and subgraphs. We start with the simplest structure, which is the single-rate equivalent of an MRSDF channel, and conclude with single-rate equivalents of strongly connected MRSDF graphs.

6.2.1 Structure: parallel and crossing channels

Consider a consistent MRSDF graph $\mathcal{G}$, with repetition vector q and its single-rate equivalent, $\mathcal{H}$. The latter may be obtained from the former using Algorithm 5, presented earlier in Chapter 4. We restrict our attention to the subgraph of $\mathcal{H}$ that corresponds to a single channel vw in $\mathcal{G}$, and denote this subgraph by $\mathcal{H}_{vw}$. This subgraph consists of $q_v + q_w$ actors, and q_w channels (see Chapter 4 for details). Each actor in $\mathcal{H}_{vw}$ corresponds to a single firing of actor v or w in $\mathcal{G}$; as before we denote the q_v actors corresponding to firings of v by $v_1, \dots v_{q_v}$, and the q_w actors corresponding to firings of w by $w_1, \dots w_{q_w}$. In the remainder of this section, we use the shorthand notation $\tilde{\pi}_{vw}$, defined by:

$$\tilde{\pi}_{vw}(k) = \pi_{vw}(k) \bmod_1 q_v. \tag{6.1}$$

Note that for each channel $v_i w_j$ in $\mathcal{H}_{vw}$, we have $i = \tilde{\pi}_{vw}(j)$, and $\delta_{v_i w_j} = \left\lfloor \frac{q_v - \pi_{vw}(j)}{q_v} \right\rfloor$. Both equalities follow from Algorithm 5.

The structure of $\mathcal{H}_{vw}$ emerges from the in-order token consumption, as given by the predecessor function π_{vw}, and the balance equations (see Chapter 2). The latter states that the number of tokens produced onto vw by q_v firings of actor v are consumed by q_w firings of actor w. Due to the presence of initial tokens on the channel, these q_w consuming firings may span at most *two* consecutive graph iterations (such is the case if the number of initial tokens on vw is not a multiple of the channel's consumption rate). In other words, the number of tokens on any two channels in $\mathcal{H}_{vw}$ can not differ by more than one.

The in-order token consumption gives rise to a non-decreasing number of tokens on channels leaving actors in $\mathcal{H}_{vw}$, in the following way. Consider actors v_i and v_j in $\mathcal{H}_{vw}$, with $i < j$. The number of tokens on any channel leaving v_j can not be lower than the number of tokens on any channel leaving v_j.

The above means that for channels in $\mathcal{H}_{vw}$ that are *disjoint*, meaning that they share neither source nor sink, their number of initial tokens may be inferred simply by looking at the firing indices of their sources and sinks. We introduce the following terminology to formalise the structure of $\mathcal{H}_{vw}$:

Definition 6.1 (Parallel and crossing channels). *Let $\mathcal{G}$ be a consistent MRSDF graph, vw a channel in $\mathcal{G}$, and $\mathcal{H}_{vw}$ the single-rate equivalent of channel vw. Furthermore, let $e_1 = v_{i_1} w_{j_1}$ and $e_2 = v_{i_2} w_{j_2}$ be two channels (with $i_1 = \tilde{\pi}_{vw}(j_1)$ and $i_2 = \tilde{\pi}_{vw}(j_2)$ in $\mathcal{H}_{vw}$). The relations* parallel *and* crossing *are defined as follows:*

» *e_1 is crossing with e_2, denoted $e_1 \nparallel e_2$, if:*

$$(i_2 > i_1 \wedge j_2 < j_1) \vee (i_2 < i_1 \wedge j_2 > j_1).$$

» *e_1 is parallel with e_2, denoted $e_1 \parallel e_2$, if:*

$$(i_1 > i_2 \wedge j_1 > j_2) \vee (i_1 < i_2 \wedge j_1 < j_2).$$

Note that our definition of *parallel* channels should not be confused with parallel channels found in a multigraph (where they refer to multiple edges connecting the same two nodes).

Two crossing channels can not have the same number of tokens, since in that case tokens would be consumed out of order. Therefore, one of the two crossing channels must have precisely one token more than the other. This is formalised in the following proposition:

Proposition 6.1 (different delays on crossing edges). *Let vw be a channel in a consistent MRSDF graph, and $\mathcal{H}_{vw}$ its single-rate equivalent. Let $v_{i_1}w_{j_1}$ and $v_{i_2}w_{j_2}$ be two* crossing *channels in $\mathcal{H}_{vw}$, with initial tokens k_1 and k_2, respectively, and with $i_2 > i_1$ (and thus $j_2 < j_1$). Then $k_2 = k_1 + 1$.*

Proof. First of all, since $i_1 = \tilde{\pi}_{vw}(j_1)$ and $i_2 = \tilde{\pi}_{vw}(j_2)$, we have $\tilde{\pi}_{vw}(j_2) > \tilde{\pi}_{vw}(j_1)$. Furthermore, since $j_2 < j_1$, we have $\pi_{vw}(j_2) < \pi_{vw}(j_1)$. From the definition of the predecessor function (see Equation 3.3 in Chapter 3) it then follows that $\delta_{v_{i_2}w_{j_2}} < \delta_{v_{i_1}w_{j_1}}$. Channel $v_{i_2}j_{j_2}$ thus has more tokens than channel $v_{i_1}w_{j_1}$. The fact that the number of tokens on the two channels can not differ by more than one completes the proof. □

Following a similar reasoning, we may infer that two parallel channels in $\mathcal{H}_{vw}$ have the *same* number of tokens:

Proposition 6.2 (same delays on parallel edges). *Let vw be a channel in a consistent MRSDF graph, and $\mathcal{H}_{vw}$ its single-rate equivalent. Let $v_{i_1}w_{j_1}$ and $v_{i_2}w_{j_2}$ be two* parallel *edges in $\mathcal{H}_{vw}$, with initial tokens k_1 and k_2, respectively, and with $i_2 > i_1$ (and thus $j_2 > j_1$). Then $k_2 = k_1$.*

Proof. $j_2 > j_1$ gives $\pi_{vw}(j_2) > \pi_{vw}(j_1)$ and $i_2 > i_1$ gives $\tilde{\pi}_{vw}(j_2) > \tilde{\pi}_{vw}(j_1)$. It then follows that $\delta_{v_{i_2}w_{j_2}} = \delta_{v_{i_1}w_{j_1}}$. □

The relationship between parallel (and crossing) channels and their number of initial tokens can be extended to *disjoint paths* (two paths are disjoint if no actor is shared between the paths) in the single-rate equivalent of a path in a consistent MRSDF graph. Instead of a single channel between actors v and w, we now consider the situation of having n actors a_1, $a_2, \ldots, a_n$, and a path consisting of MRSDF channels a_1a_2, $a_2a_3, \ldots, a_{n-1}a_n$. In the single-rate equivalent of this path, this sequence of channels is represented by several paths, the indices of which are now denoted in superscript, so $(a_1^{i_1}a_2^{i_2} \ldots a_{n-1}^{i_{n-1}}a_n^{i_n})$ denotes such a path in which channel a_ka_{k+1} is represented by channel $a_k^{i_k}a_{k+1}^{i_{k+1}}$.

We refer to a path by the sequences of its actors and denote the number of tokens on a path P by $|P|_d$. Paths are assumed to be *simple*, i.e., no actors is repeated in a path. Similar to the definitions for channels in the single-rate equivalent, we introduce the following terminology:

Definition 6.2 (Parallel and crossing paths). *Let $\mathcal{H}_P$ be the single-rate equivalent of a path $P = (a_1a_2 \ldots a_n)$ in a consistent MRSDF graph $\mathcal{G}$. Furthermore, let $P_i = (a_1^{i_1}a_2^{i_2} \ldots a_n^{i_n})$ and $P_j = (a_1^{i_1}a_2^{i_2} \ldots a_n^{i_n})$ be two disjoint paths in $\mathcal{H}_P$. Then P_i and P_j are:*

» parallel, *denoted* $P_i \parallel_p P_j$, *if* $(j_1 > i_1 \wedge j_n > i_n) \vee (j_1 < i_1 \wedge j_n < i_n)$.

» crossing, *denoted* $P_i \nparallel_p P_j$, *if* $(j_1 > i_1 \wedge j_n < i_n) \vee (j_1 < i_1 \wedge j_n > i_n)$.

Analogous to the case of disjoint channels, the relative number of tokens on parallel and crossing paths depends only on the indices of the first and last actors of the paths. This property can be derived from Propositions 6.1 and 6.2 in a straightforward way, using induction on the number of actors represented by the paths:

Lemma 6.1 (relative tokens on disjoint paths). *Let $\mathcal{H}_P$ be the single-rate equivalent of a path $P = (a_1 a_2 \ldots a_n)$ in a consistent MRSDF graph $\mathcal{G}$, and let $P_i = (a_1^{i_1} \ldots a_n^{i_n})$ and $P_j = (a_1^{j_1} \ldots a_n^{j_n})$ be two paths in $\mathcal{H}_P$, with $i_1 > j_1$. Then:*

(1) $|P_i|_d = |P_j|_d$ if P_i and P_j are parallel;

(2) $|P_i|_d = |P_j|_d + 1$ if P_i and P_j are crossing.

Proof. We prove this by induction on the number of actors n of the paths. First of all, note that $a_k^{i_k} \neq a_k^{j_k}$ for all $k = 1, 2, \ldots, n$ since two disjoint edges do not have the same sink (and by assumption the inequality holds for $k = 1$). In case $n = 2$ both paths are in fact channels and we obtain the result from Propositions 6.1 and 6.2. Next, suppose the result holds for all paths with at most n actors, and consider two paths P_i and P_j with $n > 2$ actors and with final edges $e_i = a_n^{i_n} a_{n+1}^{i_{n+1}}$ and $e_j = a_n^{j_n} a_{n+1}^{j_{n+1}}$, respectively. We assume again that $i_1 > j_1$. There are four cases to consider, depending on the relative orders of i_n, j_n and i_{n+1}, j_{n+1}. If $j_n < i_n$ ($j_n > i_n$) and $j_{n+1} < i_{n+1}$, then the paths $P_i - a_{n+1}^{i_{n+1}}$ and $P_j - a_{n+1}^{j_{n+1}}$ are parallel (crossing) and $a_n^{i_n} a_{n+1}^{i_{n+1}}$ and $a_n^{j_n} a_{n+1}^{j_{n+1}}$ are parallel (crossing), and the claims follow by induction and by Propositions 6.1 and 6.2. The other two cases are similar. □

An important consequence of the above lemma is the following: If we have three pairwise disjoint paths and each crosses at least one of the other two paths, then two of these three paths must be parallel. Furthermore, if a path crosses two other disjoint paths, then these two paths must be parallel. These two implications are captured in the following corollary, which follows directly from the above lemma.

Corollary 6.1 (restrictions on disjoint paths). *Let P_i, P_j and P_k be three* disjoint *paths in the single-rate equivalent of a path P in a consistent MRSDF graph, with $k_1 > j_1 > i_1$. Furthermore let P_i cross P_j. Then:*

(1) if P_j and P_k cross, then P_i and P_k do not cross;

(2) if P_j and P_k are parallel, then P_i and P_k cross.

Proof. Both claims may be easily proven by contradiction using Lemma 6.1. As the proofs for both claims is similar, it suffices to prove claim (1). Assume paths P_i and P_k do cross. Then by Lemma 6.1, P_i, P_j and P_k must all have different numbers of tokens, in particular $|P_i|_d \neq |P_j|_d$. But since $k_1 > j_1$ and $k_1 > i_1$, by the same lemma we have $|P_k|_d = |P_i|_d + 1$ and $|P_k|_d = |P_j|_d + 1$, which implies the contradiction $|P_i|_d = |P_j|_d$. □

Note that the results of the above lemma and its corollary also hold if we consider walks instead of paths (so actors may be repeated) in the MRSDF graph. In particular, the result also holds for *closed walks*, i.e., walks for which the first and last actor are the same. In the following two sections, we describe how this leads to an efficient analysis of MRSDF graphs.

6.2.2 The throughput of closed walks

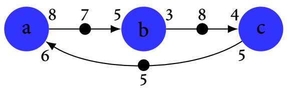
Consider again the single-rate equivalent $\mathcal{H}$ of a consistent MRSDF graph $\mathcal{G}$ with repetition vector q. Let C be a cycle in $\mathcal{G}$, consisting of actors $a_1, a_2, \ldots, a_n$ and channels $a_1a_2, a_2a_3, \ldots, a_{n-1}a_n, a_na_1$.

As before, we denote by $\mathcal{H}_C$ the subgraph of $\mathcal{H}$ that corresponds to C. This subgraph has a sequence of $q_k = q_{a_k}$ actors $a_k^1, a_k^2, \ldots, a_k^{q_k}$ for every actor a_k, and channels $a_k^i a_{k+1}^j$ (and $a_n^i a_1^j$) representing the channels in C, as defined before. For convenience, we repeat the sequence of q_1 nodes for actor a_1 at the end and think of $\mathcal{H}_C$ as an array of $n+1$ columns, where the sequences of actors are ordered from left to right, and where the leftmost and rightmost sequences are identical, representing the actor $a = a_1$ (see Figure 6.3(b)). To distinguish these two sequences, we denote the leftmost sequence by $L = L(a)$ and the rightmost by $R = R(a)$.

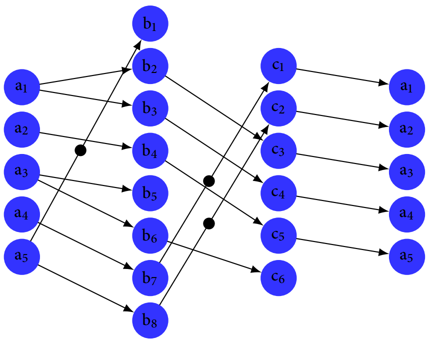
Now consider the *cycle-induced* subgraph of the single-rate equivalent. This graph is obtained by removing all actors and channels that do not lie on a cycle (see Figure 6.3(c)) and consists of a number of disjoint (since each actor has an indegree of one) paths, each of which starts in L and ends in R. Furthermore, if there are n paths in the subgraph, each column contains precisely n actors. Because each path has the same length and the (relative) number of tokens on a path is (by Lemma 6.1), fully determined by its start and end vertices, we choose to compactly represent the cycle-induced subgraph by a *permutation* $\rho : \{1, \ldots, n\} \to \{1, \ldots, n\}$. This permutation maps the index of an actor in L to the index of an actor in R, where the index of an actor is based on the superscripts of the actors (i.e., $a_k^i < a_k^j$ if $i < j$).

In the remainder of this section, we shall refer to actors (in L and R) by their said index; paths in the cycle-induced subgraph then start in an actor $i \in \{1, \ldots, n\}$ and terminate in $\rho(i) \in \{1, \ldots, n\}$.

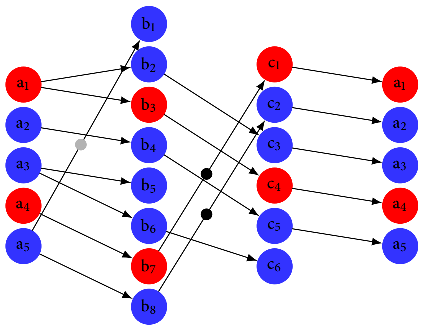
Representing the cycle-induced subgraph as a permutation on a set of integers reveals a clear structure in the single-rate equivalent of an MRSDF cycle. Consider the case where two parallel paths start in subsequent actors, indexed i and $i+1$. Using the lemma stated above and its corollary we may derive that these paths also terminate in subsequent vertices, or $\rho(i+1) = \rho(i) + 1$ (see Figure 6.4(a)), which leads to the following proposition:



(a) MRSDF graph

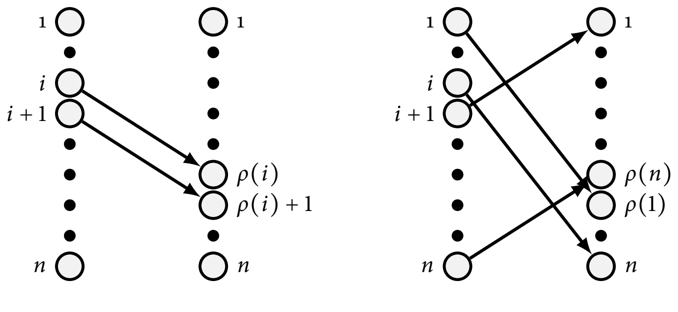


(b) Equivalent HSDF graph



(c) Cycle-induced subgraph with two (disjoint) cycles

Figure 6.3 – An MRSDF graph, its single-rate equivalent in column representation (with actor a duplicated), and the cycle-induced subgraph of the single-rate equivalent. The two cycles of the latter have the same cycle ratio.



(a) parallel paths. (b) crossing paths.

Figure 6.4 – Structure of the permutation ρ.

Proposition 6.3. *Consider the single-rate equivalent, $\mathcal{H}_C$ of a cycle C in a consistent MRSDF graph. Let P_i and P_{i+k} be two parallel paths in $\mathcal{H}_C$, which start in actors indexed i and $i + k$, respectively, with $k > 0$. Then $\rho(i + k) = \rho(i) + k$.*

Proof. Let $k = 1$. Note that since P_i and P_{i+1} are parallel, we have $\rho(i + 1) > \rho(i)$. Assume $\rho(i + 1) > \rho(i) + 1$. There must exist j such that $\rho(j) = \rho(i) + 1$. Let P_j be the path that connects j with $\rho(i) + 1$. In case $j < i$, we have $P_j \nparallel_p P_i$ and $P_j \parallel_p P_{i+1}$. By Corollary 6.1 however, we have $P_j \nparallel_p P_{i+1}$, which is a contradiction. The assumption that $j > i + 1$ leads to a contradiction in a similar way. We thus have $\rho(i + 1) = \rho(i) + 1$, and by straightforward induction on k it follows that $\rho(i + k) = \rho(i) + k$. □

For crossing paths, a similar relation in terms of ρ exists. This is illustrated in Figure 6.4(b) and may be understood by considering two crossing paths P_i and P_{i+1} that start in subsequent vertices i and $i + 1$, respectively. We may divide the set of paths in two subsets: the first subset contains all paths starting in actors indexed $1, 2, \ldots, i$, and the second contains all paths starting in actors $i + 1, \ldots, n$. By Corollary 6.1, both subsets contain pairwise parallel paths. Furthermore, each path in one subset crosses all other paths in the other subset. As a consequence, we must have $\rho(i) = n$ and $\rho(i+1) = 1$. The following proposition formally generalises this conclusion:

Proposition 6.4. *Let $\mathcal{H}_C$ be the single-rate equivalent of a cycle C in a consistent MRSDF graph. Furthermore, let P_i and P_{i+1} be two crossing paths in $\mathcal{H}_C$, which start in actors indexed i and $i + 1$, respectively. Then $\rho(i + k) = k$ for $k > 0$ and $\rho(i - k) = n - k$ for $k \geq 0$.*

Proof. Assume $\rho(i + 1) > 1$. There must exist j such that $\rho(j) = 1$. Let P_j be the path that connects j with $\rho(j)$. In case $j < i$, we have $P_j \parallel_p P_{i+1}$ and $P_j \parallel_p P_i$.

By Corollary 6.1 however, we have $P_i \nparallel_p P_j$, which is a contradiction. In case $j > i + 1$, we have $P_j \nparallel_p P_i$ and $P_j \nparallel_p P_{i+1}$, which again contradicts Corollary 6.1, thus $\rho(i+1) = 1$. Following a similar reasoning it follows that $\rho(i) = n$. By straightforward induction on k it follows that $\rho(i+k) = k$ and $\rho(i-k) = n-k$. □

We are now ready to move from paths in $\mathcal{H}_C$ to cycles. A simple cycle in $\mathcal{H}_C$ may be constructed by repeatedly applying the permutation ρ until the start actor is reached again. For this, let $\rho^{k+1}(i) = \rho(\rho^k(i))$ and $\rho^0(i) = i$. Due to the structure of the single-rate equivalent, the cycle ratio (i.e., the ratio between execution time and tokens, see Chapter 2) from a cycle in $\mathcal{H}_C$ may be deduced from the *number of cycles* in $\mathcal{H}_C$, and the number of tokens on a path in $\mathcal{H}_C$. The following theorem formally states and proves this, by exploiting the definition of ρ.

Lemma 6.2 (Paths infer cycle ratio). *Let $\mathcal{H}_C$ be the* cycle-induced *subgraph of the single-rate equivalent of a cycle C in a consistent MRSDF graph. Let $\mathcal{H}_C$ consist of N paths. Furthermore, let $C_i = \{i, \rho(i), \ldots, \rho^{n_i}(i)\}$ with $\rho^{n_i}(i) = i$ be a simple cycle in $\mathcal{H}_C$, with $\rho(i) \geq i$. As before, let P_j denote the path that starts in an actor indexed j. The cycle ratio of C_i, denoted $\lambda(C_i)$, satisfies*

$$\lambda(C_i) = \frac{N\,|C|_\tau}{N\,|P_i|_d + \rho(i) - i}.$$

Proof. Let $\mathcal{H}_C$ consist of N paths. We may define ρ as follows:

$$\rho(a) = (a + k) \bmod_1 N,$$

for some $k \in \{1, \ldots, N\}$. Using this definition of ρ, the length l of a cycle starting at an actor indexed a may be calculated by finding the minimum positive (i.e., greater than zero) value of l that satisfies the linear congruence: $a + l \cdot k \equiv a \pmod{N}$. This solution l is given by:

$$l = \frac{N}{\gcd(k, N)}.$$

Cycle C_i thus comprises l paths. We may divide these l paths into two subsets, S_0 and S_1: S_0 contains paths P_j with $\rho(j) \geq j$, and S_1 contains those paths P_j for which $\rho(j) < j$. Both sets contain pairwise parallel paths and each of the paths in S_1 crosses every path in S_0. By Lemma 6.1, each path in S_0 has $|P_i|_d$ tokens, and each path in S_1 has $|P_i|_d + 1$ tokens. Furthermore, set S_1 contains precisely $\frac{k}{\gcd(N,k)}$ paths. The total number of tokens on cycle C_i is thus given by:

$$|C_i|_d = \frac{N\,|P_i|_d + \rho(i) - i}{\gcd(k, N)}.$$

The total execution time of actors on cycle C_i follows from the l paths in $\mathcal{H}_C$ that compose C_i. Cycle C_i thus has a total execution time $|C_i|_\tau$ of:

$$|C_i|_\tau = \frac{N\,|C|_\tau}{\gcd(k, N)}.$$

The quotient of $|C_i|_\tau$ and $|C_i|_d$, with k substituted for $\rho(i) - i$, gives the cycle ratio stated by the lemma. □

A straightforward result from this lemma is that any two (simple) cycles in $\mathcal{H}_C$ must have the same cycle ratio. The following theorem formally states and proves this. Note that the theorem is not restricted to *simple* MRSDF cycles, but applies to any closed walk in the MRSDF graph.

Theorem 6.1 (Two cycles have the same cycle ratio). *Let $\mathcal{H}_C$ be the cycle-induced subgraph of the single-rate equivalent of a cycle C in a consistent MRSDF graph, and let $C_i = \{i, \rho(i), \ldots, \rho^{n_i}(i)\}$ with $\rho^{n_i}(i) = i$ and $C_j = \{j, \rho(j), \ldots, \rho^{n_j}(j) = j\}$ be two disjoint simple cycles in $\mathcal{H}_C$. Then $\lambda(C_i) = \lambda(C_j)$.*

Proof. This follows directly from Lemma 6.2: without loss of generality, assume $\rho(i) \geq i$ and $\rho(j) \geq j$. Consider the paths P_i and P_j, which start in actors indexed i and j, respectively. By assumption, these paths are parallel, thus by Lemma 6.1, they have the same number of tokens. By Lemma 6.2, C_i and C_j then have the same cycle ratio. □

6.2.3 Efficient subgraph analysis

Theorem 6.1 provides an efficient approach to compute the throughput of an MRSDF cycle $\mathcal{G}$. In the single-rate equivalent, $\mathcal{H}$ of $\mathcal{G}$, each actor has precisely one incoming channel. An actor may either lie on a cycle, or on a path that is reachable from actors that lie on a cycle. In order to find a cycle in $\mathcal{H}$, we may thus, rather than fully constructing $\mathcal{H}$, pick a random actor in $\mathcal{H}$, and follow channels in *reverse* direction until a cycle is detected. By Theorem 6.1, this cycle must then be (one of) the graph's critical cycle(s).

A straightforward question is whether the same approach works for an arbitrary MRSDF graph: Can we choose a random actor in the single-rate equivalent and restrict the search for a critical cycle to the subgraph reachable (by following channels in reverse direction) from it? In this section we show that this is indeed the case for strongly connected MRSDF graphs (note that the throughput of an MRSDF graph that is not strongly connected may be calculated from the throughputs of its strongly connected components, as is described in [40]).

An important property in understanding why this approach works, concerns the reachability of vertices in an MRSDF. This is formalised in Proposition 6.5, which immediately follows from the fact that in a strongly connected MRSDF graph each actor has at least one incoming channel.

Proposition 6.5 (reachability). *Let a and b be actors in a consistent and strongly connected MRSDF graph $\mathcal{G}$, and let $\mathcal{H}$ be the single-rate equivalent of $\mathcal{G}$. Then for each vertex b_j in $\mathcal{H}$ there exists an i such that $\mathcal{H}$ contains a path from a_i to b_j.*

In words, Proposition 6.5 states that if, in an MRSDF graph $\mathcal{G}$, by following channels in reverse, actor a is reachable from actor b, then, in the single-rate equivalent of $\mathcal{G}$, from any actor that represents a firing of actor b we may reach a vertex that represents a firing of actor a. We use this fact together with Theorem 6.1 to derive the important result that only a subgraph of the single-rate equivalent of an MRSDF graph needs to be explored for its critical cycle:

Theorem 6.2 (Subgraph analysis). *Let $\mathcal{H}$ be the single-rate equivalent of a consistent and strongly connected MRSDF graph $\mathcal{G}$, and s an arbitrary vertex in $\mathcal{H}$. Furthermore, let $\mathcal{H}_s$ be the induced subgraph of $\mathcal{H}$ that consists of those vertices from which a path to s exists. The maximum cycle ratio of $\mathcal{H}_s$ is equal to the maximum cycle ratio of $\mathcal{H}$.*

Proof. Let the cycle attaining maximum cycle ratio in $\mathcal{H}$ be $C = (a_1^{i_1} \dots a_n^{i_n} = a_1^{i_1})$. Cycle C is contained in the single-rate equivalent of a cycle W in $\mathcal{G}$, with $W = (a_1, a_2, \dots, a_n = a_1)$ (Note that this cycle may be a walk in $\mathcal{G}$, i.e., vertices may be repeated).

We prove the theorem by contradiction. Let $\mathcal{H}_s$ not contain C. Then s obviously does not lie on C. Furthermore, there is no path from a vertex on C to s, since if this were the case, C would be in $\mathcal{H}_s$.

Now choose an actor $v = a_1^{i_m}$ that is not reachable from C, but from which there is a path to s (i.e., v is in $\mathcal{H}_s$). By Proposition 6.5, such an actor can always be found. If in the single-rate equivalent of W we follow channels in reverse direction from v, then eventually a cycle C' will be found. By Theorem 6.1, C' has the same cycle ratio as C. Since C' is in $\mathcal{H}_s$ and $\mathcal{H}_s$ is a subgraph of $\mathcal{H}$, the maximum cycle ratio of $\mathcal{H}_s$ is equal to the maximum cycle ratio of $\mathcal{H}$. We may thus restrict our search to $\mathcal{H}_s$. □

Theorem 6.2 implies that it is not necessary to explore the entire single-rate equivalent for its critical cycle. More specifically, it does not matter whether the single-rate equivalent is strongly connected or not. We may thus, in a similar way to the approach for MRSDF cycles proposed in the previous section, choose an arbitrary root actor in the single-rate equivalent and search the subgraph that is induced by the actor and channels from which the root actor is reachable, for its critical cycle.

As an example, consider the MRSDF graph and its single-rate equivalent depicted in Figure 6.5. The latter (Figure 6.5(b)) consists of nine HSDF actors. We choose HSDF actor a_4 as a root actor, and follow channels in reverse to construct the subgraph induced by those actors and channels from which a_4 can be reached. This results in a subgraph of six HSDF actors and eight channels. Note that the choice of root actor matters; had we chosen actor a_2, then the subgraph would have consisted of only three actors: a_2, b_1 and c_3.

The example just discussed raises the point that the choice of root actor has an important impact on the size of the subgraph that is eventually analysed for its maximum cycle ratio. We now turn to the final result of this section, which allows

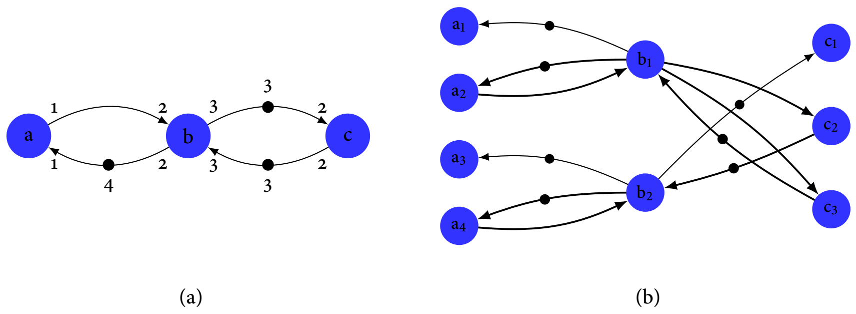


Figure 6.5 – The throughput of an MRSDF graph may be computed from a subgraph of its single-rate equivalent. This subgraph consists of those actors and channels from which an arbitrary, fixed, root actor can be reached.

one to decrease the size of the analysed subgraph even further. For this, we consider again the single-rate equivalent $\mathcal{H}$ of a consistent and strongly connected MRSDF graph $\mathcal{G}$. Furthermore, we choose an arbitrary actor s in $\mathcal{H}$, and let $\mathcal{H}_s$ denote the subgraph induced by paths that terminate in s.

Recall that $\mathcal{H}$ was obtained by Algorithm 5, which has omitted from $\mathcal{H}$ those channels that do not impose a binding constraint on the schedules of $\mathcal{H}$ (see Section 4.1.4). This means that any admissible schedule for $\mathcal{H}$ can be mapped to an admissible schedule for $\mathcal{G}$. Now consider the strongly connected components of $\mathcal{H}$. Each strongly connected component has its own maximum cycle ratio; this means that some of these components may process tokens faster than other ones. In particular, the component with maximum cycle ratio is in this sense the "slowest" component. It processes tokens at most as fast as upstream components do. As a result, if we let actors in $\mathcal{H}$ fire as soon as they are enabled, tokens will accumulate on channels leading into the slowest component. If we map this self-timed schedule to a schedule for $\mathcal{G}$, then in $\mathcal{G}$ there must also be some channel on which channels keep on accumulating. This is a contradiction: since $\mathcal{G}$ is strongly connected, the number of tokens that accumulate on any channel is bounded [40].

From the above we may conclude that the computation of the maximum cycle ratio of $\mathcal{H}_s$ can be restricted to a specific strongly connected component. This is the component that has no upstream components, because if it would have had one, the reasoning detailed above leads to a contradiction. As a result, we may restrict the analysis of $\mathcal{H}$ even further, by topologically sorting the strongly connected components, and choosing the component that topologically precedes all other components. We refer to this particular strongly connected component as a *feeding component*. The feeding component corresponds to a source node in the *condensation* of $\mathcal{H}$, which is the graph obtained by contracting each strongly connected component to a single node [92]. The following theorem formalises this reasoning.

Theorem 6.3 (Feeding components are bottlenecks). *Let $\mathcal{G}$ be a strongly connected MRSDF graph, and $\mathcal{H}$ its single-rate equivalent. Furthermore, let s be an arbitrary actor in $\mathcal{H}$, and $\mathcal{H}_s$ the subgraph of $\mathcal{H}$ induced by paths that terminate in s.*

The throughput of $\mathcal{G}$ is equal to the throughput of each of the strongly connected components that correspond to source nodes in the condensation of $\mathcal{H}_s$.

Proof. The proof follows the same lines as the reasoning in text above. Let $\mathcal{S}^*$ be the strongly connected component in $\mathcal{H}_s$ that has maximum cycle ratio, and let $\mathcal{S}_0$ be a strongly connected component in $\mathcal{H}_s$ that precedes $\mathcal{S}^*$ in the topological ordering of the strongly connected components of $\mathcal{H}_s$. Note that since $\mathcal{H}_s \subseteq \mathcal{H}$, these components are also present in $\mathcal{H}$.

Now consider a self-timed schedule of $\mathcal{H}$. Since the throughput of $\mathcal{S}_0$ is higher than the throughput of $\mathcal{S}^*$, and there is a path from $\mathcal{S}_0$ to $\mathcal{S}^*$, the rate at which tokens are produced onto this path is higher than the rate at which they are consumed from the path by actors in $\mathcal{S}^*$. Since firings of actors in $\mathcal{H}$ correspond to firings of actors in $\mathcal{G}$, the accumulation of tokens onto a channel in $\mathcal{H}$ corresponds to the accumulation of tokens on some channel in $\mathcal{G}$. This accumulation is unbounded, which contradicts the fact that the capacity of each channel in a strongly connected MRSDF graph can be upper-bounded [40]. □

Theorem 6.3 translates to an efficient algorithm for computing the throughput of an MRSDF graph by computing the maximum cycle ratio of its single-rate equivalent. The algorithm is listed in Algorithm 8. To illustrate the algorithm, consider again the single-rate equivalent depicted in Figure 6.5(b). The subgraph that is eventually analysed for its maximum cycle ratio consists of the following three strongly connected components, in topological order: $\{a_2, b_1, c_3\}$, $\{c_2\}$ and $\{a_4, b_2\}$. The analysis may thus be restricted to the subgraph induced by actors a_2, b_1 and c_3.

Note that computation of strongly connected components may be done in time linear in the size of the graph, using either of the well-known algorithms of Tarjan [92] or Kosaraju [85]. These algorithms both essentially perform a depth-first search on the graph (Kosaraju's algorithm performs two). Furthermore, the construction of the subgraph, denoted $\mathcal{H}_s$ in Algorithm 8, induced by paths that terminate in a specific actor may be done with a single depth-first search as well, and may be combined with the computation of the strongly connected components. Obviously, the construction of the subgraph $\mathcal{H}_s$ does not require the full single-rate equivalent to be constructed first.

In Chapter 7 we compare the performance of Algorithm 8 with state-of-the art throughput analysis methods, on a set of MRSDF graphs.

Algorithm 8: Computes the throughput of an MRSDF graph by computing the maximum cycle ratio of the graph's single-rate equivalent.

```
1 input  : A strongly connected MRSDF graph $\mathcal{G}$.
2 output : The throughput of $\mathcal{G}$.
3 Let $\mathcal{H}$ be the single-rate equivalent of $\mathcal{G}$
4 Let $s$ be an arbitrary actor in $\mathcal{H}$
5 Let $\mathcal{H}_s$ be the subgraph of $\mathcal{H}$ induced by paths terminating in $s$
6 Compute a list $L_{\text{scc}}$ of strongly connected components of $\mathcal{H}_s$
7 Topologically sort $L$
8 Compute $\lambda^*$, the maximum cycle ratio of the first element of $L$
9 return $\frac{1}{\lambda^*}$
```

6.3 An incremental approach

In an HSDF graph, composition of the throughput of cycles is straightforward: the throughput of the composition of two cycles is the minimum of the throughputs of the individual cycles. For non-homogeneous SDF graphs this is generally not true, as was illustrated in the beginning of this chapter with the graph of Figure 6.2: in MRSDF and CSDF graphs, *parallelism* must be taken into account.

Computing the throughput of an SDF graph by analysing the single-rate equivalent for its maximum cycle ratio is a potentially costly approach, as the size of the single-rate equivalent is exponential in the size of the original graph [77]. Although the approach outlined in the previous section improves the efficiency of such an analysis by exploring only a part of the single-rate equivalent, it avoids, rather than targets, the nature of the problem, which is the incorporation of parallelism in composing the throughput of cycles. In this section, we further elaborate on the interplay between throughput and parallelism in the composition of cycles.

Section 6.3.2 develops this concept into a transformation that compresses single-rate equivalents into smaller graphs by enforcing the parallel execution of groups actor firings.

We show how, by appropriately grouping firings into parallel executing groups, and unfolding actor firings, an MRSDF graph is transformed into a graph in which the throughput is attained by a single cycle. This gives rise to an approach to the computation of throughput by incrementally transforming the graph, while keeping its size to a minimum. We present this approach in Section 6.3.3.

6.3.1 Estimated throughput

The single-rate approximations, which we presented in the previous chapter, provide a means to compute lower and upper bounds on the throughput of an SDF graph. By Theorem 5.1, the maximum cycle ratio of the *optimistic* approximation gives an *upper* bound on the throughput, and the *pessimistic* approximation gives a lower bound. The difference between these bounds gives an indication of the error made by the approximation. If the bounds are equal, then we say that the approximation is *exact*.

In the remainder of this section, we restrict our attention to graphs that are (consistent) SDF *cycles*, in which actors have a non-varying execution time. The throughput of such a graph is thus bounded by the cycle ratios of its single-rate approximations. The approximation is exact, if every actor in the single-rate equivalent of the cycle lies on a cycle. This is a consequence of the structural properties of the single-rate equivalent, which we described in detail in Section 6.2.

Theorem 6.4 (Exact cycle approximations). *Let $\mathcal{G}$ be a consistent CSDF cycle, where each actor has a non-varying execution time, and with single-rate equivalent $\mathcal{H}$. Furthermore, let $\hat{\lambda}$ and $\check{\lambda}$ be the cycle ratios of the pessimistic and optimistic single-rate approximation of $\mathcal{G}$. If every actor in $\mathcal{H}$ lies on a cycle, then the approximations are exact, that is: $\hat{\lambda} = \check{\lambda}$.*

Proof. By propositions 6.3 and 6.4, every pair of cycles in $\mathcal{H}$ is *disjoint*. That is, every actor in $\mathcal{H}$ lies on precisely on cycle; the in-degree and out-degree of every actor is equal to one. This implies that every actor v in $\mathcal{G}$ is represented by the same number of HSDF actors, $v_1, \ldots, v_{q_v}$, in $\mathcal{H}$. In other words, all entries in the repetition vector of $\mathcal{G}$ are equal.

Now, consider a channel vw in $\mathcal{G}$. Observe that, due to the above, the predecessor function associated with vw, π_{vw}, is a *bijection*. Again by propositions 6.3 and 6.4, this bijection must be a *translation*, of the form $\pi_{vw}(k) = k - c$, for some constant $c \in \mathbb{Z}$. From the construction of the single-rate approximations of $\mathcal{G}$ (see Chapter 5), it follows that the predecessor functions of the associated channels, $\hat{v}\hat{w}$ and $\check{v}\check{w}$, are then given by $\hat{\pi}_{vw}(k) = \check{\pi}_{vw}(k) = k - \beta$ (for some constant $\beta \in \mathbb{Z}$). Hence, $\hat{v}\hat{w}$ and $\check{v}\check{w}$ have the same number of *tokens*. Since, also, actors $\hat{v}$ and $\check{v}$ have the same execution time, the cycle ratios of the single-rate approximations are the same, as well. □

The above theorem gives a relation, between the accuracy of the single-rate approximations of a cycle, and the structure of its single-rate equivalent. The contrapositive of the theorem implies that, if the approximations are not exact, there is at least one actor in the single-rate equivalent that does not lie on a cycle. These actors may be *pruned*. As they do not lie on a cycle, pruning these actors does not change the maximum cycle ratio of the graph. Rather than removing the actors from the graph, we prune them by contracting several actors into a new actor. This transformation is defined as follows.

Definition 6.3 (Actor contraction). *Let $\mathcal{G}$ be an HSDF graph, and let the set $S = \{v_1, \ldots, v_n\}$ be a subset of the actors in $\mathcal{G}$. The* contraction *of the actors in S produces a graph, $\mathcal{H}$, in which actors $v_1, \ldots, v_n$ are replaced with a single actor, z, such that z is adjacent to the union of actors to which actors in S were originally adjacent. That* *is, for every channel ab in $\mathcal{G}$, where $a \in S$, there is precisely one channel zb in $\mathcal{H}$, with $\delta_{zb} = \delta_{ab}$, and, likewise, for every channel ab with $b \in S$, there is precisely one channel az in $\mathcal{H}$, with $\delta_{az} = \delta_{ab}$. Furthermore, the execution time of z is given by $\tau_z = \max_{v \in S} \tau_v$.*

The contraction of actors in an HSDF graph may yield a multigraph: if v_1 and v_2 are contracted into a single actor z, then a pair of channels v_1w and v_2w is mapped to a pair of parallel channels, both connecting actor z with actor w. These two parallel channels both impose a constraint on the firing times of w: if the channels have the same number of tokens, then the constraints they impose are strong, and we may safely prune either of the channels without changing the temporal dynamics of the graph. If the number of tokens on these two parallel channels differs, then the one with the fewest tokens imposes the stronger constraint on the firing times of w. Consequently, we may safely prune the channel that has the most tokens, without changing the temporal dynamics of the graph. By repeating this for as long as the graph has parallel edges, we obtain a simple graph (see also [88] and [52], which describe a similar procedure). The following definition formalises this transformation.

Definition 6.4 (Simplified graph). *Let $\mathcal{G}$ be an HSDF multigraph. The simplification of $\mathcal{G}$ is an HSDF graph, $\mathcal{H}$, which has the same actors as $\mathcal{G}$. Graph $\mathcal{H}$ has a channel e from actor v to w, if vw is a channel in $\mathcal{G}$. The number of tokens on e are given by: $\delta_e = \min\{\delta_c | c \in \mathcal{G} \wedge target(c) = w \wedge source(c) = v\}$.*

In the remainder of this section, we shall assume that an actor contraction is always applied to the simplification of a graph, and that it is always followed by a simplification of the resulting graph. That is, we assume that a contraction always transforms a simple graph into a simple graph.

An actor contraction changes the structure of the graph. Valid contractions are those that leave the maximum cycle ratio of the graph unaffected. The following theorem relates valid contractions, of actors in the single-rate equivalent of a graph $\mathcal{G}$, to the predecessor function of corresponding channels in $\mathcal{G}$.

Theorem 6.5 (Admissible firing contractions). *Let $\mathcal{G}$ be a consistent SDF cycle, with single-rate equivalent $\mathcal{H}$, and let vw be a channel in $\mathcal{G}$. Furthermore, let all actors in $\mathcal{G}$ have non-varying execution times.*

Let w_j and w_{j+1} be actors in $\mathcal{H}$, with j such that $\pi_{vw}(j) = \pi_{vw}(j+1)$. We may contract w_j and w_{j+1} into a new actor, $\mathring{w}$, which has the same execution time as the two contracted actors. This contraction does not affect the maximum cycle ratio of $\mathcal{H}$.

Proof. Every actor in $\mathcal{H}$ has precisely one incoming, and one outgoing channel. The connectivity of $\mathcal{H}$ is determined by the predecessor function: if $\pi_{vw}(k) = m$, then $\mathcal{H}$ has a channel from $v_{m \bmod_1 q_v}$ to $w_{k \bmod_1 q_v}$. Thus, the fact that $\pi_{vw}(j) = \pi_{vw}(j+1)$, implies that actors w_j and w_{j+1} do *not* lie on two *different* cycles. This means that the contraction of the two actors leaves the number of cycles in the graph intact. Without loss of generality, let w_j lie on a cycle C in $\mathcal{H}$. As $\mathring{w}$ has the same execution time as w_j, the contraction does not increase the execution time along C. As a result, the cycle ratio of C is unaltered by the contraction, and thus the maximum cycle ratio of $\mathcal{H}$ is unaffected. □

6.3.2 Cycle analysis by iterative vectorisation

There is a natural correspondence between admissible contraction of actors, in the single-rate equivalent of an SDF cycle, and the manipulation of production and consumption rates associated with actors in that SDF cycle. Actors in the single-rate equivalent correspond to individual firings, and their contraction thus essentially joins them together in a single firing. From a functional perspective, this new firing thus performs a composition of the functions of the contracted firings. Consequently, production and consumption rates associated with the new firing are obtained by taking the *sums* of the rates associated with the contracted firings.

A similar transformation of an actor's rates has been applied in the context of *vector processing* in digital signal processing tasks, where it is called *vectorisation* [80]. The vectorisation transformation replaces the rates associated with an MRSDF actor with an integer multiple, thereby modelling a processing of vectors of samples, rather than individual samples. In [80], vectorisation is used to decrease the number of task activations (by letting tasks process more data), which reduces the overhead of context-switching. One may thus regard actors in these MRSDF graphs as having *parametric* rates, the number of firings that compose a single graph iteration are controlled by varying the parameters. These graphs are referred to as *scalable synchronous dataflow* [79]. In this context, the parameters that control the number of firings that are grouped together are called *blocking factors*, since grouping together firings exploits opportunities to process *blocks* of data.

We generalise the notion of vectorisation, by defining its application to CSDF actors. This means that, rather than replacing non-varying rates with an integer multiple, the transformation operates on *vectors*. Following the naming convention of [80], blocking factors thus become blocking vectors. We define this transformation as follows.

Definition 6.5 (Vectorisation). *Let $\mathcal{G}$ be a CSDF graph, and v an actor in $\mathcal{G}$, with a non-varying execution time, τ_v. Furthermore, let $B = (b_1, \ldots, b_n)$ be an integer vector, such that the sum of the elements in B is an integer multiple of φ_v. The vectorisation of v with* blocking vector *B results in a new graph $\mathcal{H}$, which is obtained from $\mathcal{G}$ by replacing v with v', with $\tau_{v'} = \tau_v$. Furthermore, the consumption rate vector associated with each incoming channel uv' of v' in $\mathcal{H}$, and the production rate vector associated with each outgoing channel $v'w$, are given by:*

$$\rho^-_{uv'}(i) = \sum_{l=1}^{b_i} \rho^-_{uv}(l) \qquad\qquad \rho^+_{v'w}(i) = \sum_{l=1}^{b_i} \rho^+_{vw}(l).$$

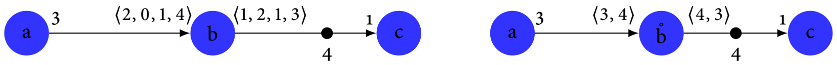


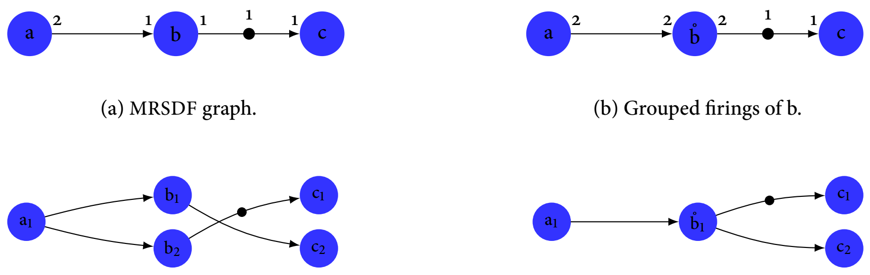
Figure 6.6 – Applying the vectorisation transformation to an actor groups together several consecutive firings into a single firing, by replacing the rates of these firings with their sums. In the graph above, actor b is vectorised by applying a blocking vector of $(3,1)$.

An example of a vectorisation is given in Figure 6.6. Actor b has cyclically varying production and consumption rates. The period of this varying pattern is four, i.e., $\varphi_b = 4$. A blocking vector of $(3,1)$ indicates that, out of every four firings, the first three are grouped together into a single firing. Applying this blocking vector in the vectorisation of b results in a graph in which firings of actor $\mathring{b}$ have cyclically varying rates, with a period of two: in the original graph, the first three firings of actor b consume a total of three tokens from channel ab, and produce four tokens onto channel bc. Consequently, in the vectorised graph, the first firing of $\mathring{b}$ consumes three tokens, and produces four tokens. The second firing of $\mathring{b}$ corresponds to the fourth firing of b.

The vectorisation transformation groups together several consecutive firings into a single firing. Firings that originally depended on one of these merged firings, now depend on this new firing. For careful choices of the blocking vector, the vectorisation transformation corresponds to the contraction of actors (followed by a simplification of the graph), given by definitions 6.3 and 6.4. We give an example to illustrate this correspondence.

Consider the graph in Figure 6.7(a). In a single iteration of this graph, actor b fires twice. The single-rate equivalent of the graph, which is shown in Figure 6.7(c), thus has two actors corresponding to the two firings of b. As the single-rate equivalent shows, both firings of b depend on the first firing of a that occurs in that same graph iteration. This fact may also be obtained from the predecessor function associated with channel ab: each odd firing has the same predecessor as the following even firing, i.e., $\pi_{ab}(2k+1) = \pi_{ab}(2k+2)$. By Theorem 6.5, this means that the contraction of the two firings of b is safe, i.e. it does not change the maximum cycle ratio of the graph. The HSDF graph obtained by the contraction of b_1 and b_2 is shown in Figure 6.7(d).

If we apply a vectorisation to the graph, by grouping together every two consecutive firings of actor b, then we obtain the graph shown in Figure 6.7(b). The single-rate equivalent of the transformed graph, which is shown in Figure 6.7(d), is equivalent to the graph that is obtained by the contraction of b_1 and b_2 in Figure 6.7(c). Observe that if, instead, we would have grouped two firings of c together, by replacing the consumption rate, associated with c, with two, then the single-rate equivalent we would obtain does not correspond to the graph obtained by contracting actors



(a) MRSDF graph.

(b) Grouped firings of b.

(c) Single-rate equivalent of (a).

(d) Single-rate equivalent of (b).

Figure 6.7 – Under specific circumstances, grouping together consecutive firings of an actor corresponds to a contraction, of the corresponding actors in the graph's single-rate equivalent, into a single actor.

c_1 and c_2.

The grouping of firings, by applying a vectorisation transformation, corresponds to a contraction, if the grouped firings depend (in terms of the predecessor function) on the same producing firing. This means that contractions that are *safe*, in the sense that they leave a graph's maximum cycle ratio unaffected, on the one hand, and the vectorisation transformation, on the other hand, are equivalent. The following lemma states this equivalence in a formal manner.

Lemma 6.3 (Contraction and grouping firings). *Let $\mathcal{G}$ be a CSDF graph, in which actors have non-varying execution times, and let $\mathcal{H}$ be the single-rate equivalent of $\mathcal{G}$. Furthermore, let $\mathcal{G}^B$ be the graph obtained by vectorising actor v in $\mathcal{G}$, with blocking vector B of length n. Also, let $S = (s_1 \dots s_n)$ be the vector obtained from B, with $s_i = \sum_{l=1}^{i} b_l$, and let graph $\mathcal{H}^C$ be the graph obtained from $\mathcal{H}$, by iteratively contracting the set of actors $\{v_j | s_k - b_k < j \leq s_k\}$ into actor $\mathring{v}_k$, for $k = 1, \dots, n$.*

If, for $1 \leq i \leq n$, we have:

$$\forall m \in \{1, \dots, b_i\} : \pi_{uv}(s_i - b_i + m) = \pi_{uv}(s_i),$$

then the single-rate equivalent of $\mathcal{G}^B$ and $\mathcal{H}^C$ are equal.

Proof. Let $\mathring{v}$ be the actor in $\mathcal{G}^B$ that corresponds to actor v in $\mathcal{G}$. Observe that the following relations, between the predecessor functions associated with incoming and outgoing channels of v and $\mathring{v}$, hold:

$$\Delta_{\mathring{v}w}(m,k) = \Delta_{vw}(s_m,k), \qquad \Delta_{u\mathring{v}}(m,k) = \Delta_{uv}(m,s_k).$$

Let $v_i w_j$ be a channel in $\mathcal{H}$, and m such that $s_{m-1} < i \leq s_m$. There is a channel $\mathring{v}_m w_j$ in $\mathcal{H}^B$.

$$\pi_{vw}(j) = i \Rightarrow s_{m-1} < \min\{k \in \mathbb{Z} | \Delta_{vw}(k, j) \geq 0\} \leq s_m.$$

This implies that $\Delta_{vw}(s_{m-1}, j) < 0$, and $\Delta_{vw}(s_m, j) \geq 0$, and thus:

$$\min\{k \in \mathbb{Z} | \Delta_{vw}(s_k, j) \geq 0\} = m \Rightarrow \pi_{\mathring{v}w}(j) = m,$$

which implies that $\mathcal{H}^B$ has a channel $\mathring{v}_m w_j$.

Next, let $u_i v_j$ be a channel in $\mathcal{H}$, and m such that $s_{m-1} < j \leq s_m$. Note that $\pi_{uv}(j) = \pi_{uv}(s_m)$.

$$\pi_{uv}(s_m) = i \Rightarrow \min\{k \in \mathbb{Z} | \Delta_{uv}(k, s_m) \geq 0\} = i \Rightarrow \pi_{u\mathring{v}}(m) = i,$$

which implies that $\mathcal{H}^B$ has a channel $u_i \mathring{v}_m$.

Finally, let $v_i w_j$ and $v_k w_j$ be two channels, and k such that $s_{k-1} < i < j \leq s_k$. Both channels are mapped to channel $\mathring{v}_k w_j$. The number of tokens on this channel is equal to the maximum of the number of tokens on v_i and w_j. □

The above implies that admissible actor contractions may be performed implicitly, by an appropriate adaptation of the rates of actors. An admissible contraction may be performed if the *gain* of a channel (see Section 2.1.3) is higher than one. This implies that for that channel, on average, every single producing firing requires more than one consuming firing, and thus some consuming firings may be grouped together into a single one. We may thus apply the vectorisation transformation iteratively, grouping firings of different actors at each step, until the gain of each channel is equal to one.

A gain that is higher than one is by itself not sufficient to allow for a safe vectorisation of the channel's consumer. Consider the example in Figure 6.8. In the depicted graph, actor b has a repetition vector entry of five. This means that, in the graph's single-rate equivalent, there are five actors that correspond to actor b. The gain of channel ab is equal to $5/2$, which is higher than one, and indeed we have $\pi_{ab}(2) = \pi_{ab}(3) = \pi_{ab}(4)$ and $\pi_{ab}(5) = \pi_{ab}(6)$. This implies that actors b_2, b_3, and b_4 can be safely contracted, and also b_5, and b_6. The latter, however, is not an actor in the single-rate equivalent. If we first simulate a single firing of b, which consumes two out of the three tokens from ab, producing one token onto bc, then we obtain $\pi_{ab}(1) = \pi_{ab}(2) = \pi_{ab}(3)$ and $\pi_{ab}(4) = \pi_{ab}(5)$. This gives rise to an admissible contraction, which can be represented by a vectorisation of b with a blocking vector of $(3, 2)$.

Definition 6.6 (Retiming). *A* retiming *of $\mathcal{G}$, by simulating n firings of v, yields a graph $\mathcal{H}$, which is obtained from $\mathcal{G}$ by changing the number of initial tokens on channels incident to v. For every incoming channel of uv in $\mathcal{H}$, let $u\mathring{v}$ be the corresponding channel in $\mathcal{H}$. The number of tokens on $u\mathring{v}$ is given by:*

$$\delta_{u\mathring{v}} = \Delta_{uv}(n, 0).$$

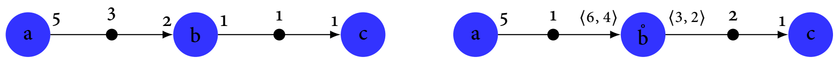


Figure 6.8 – A safe vectorisation transformation may require a retiming of the graph, by simulating a number of firings. In the graph above, actor b is vectorised by applying the blocking vector $\langle 3, 2 \rangle$, after simulating a single firing of b.

Similarly, the number on tokens on a channel $\mathring{v}w$ in $\mathcal{H}$, which corresponds to an outgoing channel, vw of v, in $\mathcal{G}$, is given by:

$$\delta_{\mathring{v}w} = \Delta_{vw}(0, n).$$

Algorithm 9 lists a procedure to perform the retiming and vectorisation, of the consumer of a particular channel, in a systematic manner.

Algorithm 9: Vectorises the consumer of a channel.

```
input  : A consistent MRSDF graph $\mathcal{G}$, and a channel $vw$ in $\mathcal{G}$.
output : A *blocking vector* for actor $w$, which indicates which firings of $w$ may be
         grouped together into a single firing.
Let $q$ be the repetition vector of $\mathcal{G}$
$B_w \longleftarrow$ empty vector
$j \longleftarrow 1$
$idx \longleftarrow 0$
while $\Delta_{vw}(0,1) \geq 0$ do
    Simulate a firing of $w$ in $\mathcal{G}$
for $i = 1$ to $\frac{q_v}{\gcd(q_v, q_w)}$ do
    $k \longleftarrow j$
    while $\Delta_{vw}(i, k+1) \geq 0$ do
        $k \longleftarrow k + 1$
    if $k > j$ then
        $idx \longleftarrow idx + 1$
        $B_w[idx] \longleftarrow k - j$
        $j \longleftarrow k$
return $B_w$
```

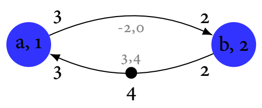
If all channels in a cycle have a gain equal to one, then the optimistic and pessimistic approximations imply whether the graph's single-rate equivalent allows for more admissible actor contractions: if the optimistic and pessimistic approximations differ, then, by the contrapositive of Theorem 6.4, there must be an actor in the single-rate equivalent that does not lie on a cycle. This actor must be the consumer of a channel for which the number of tokens on the corresponding channels in the optimistic and pessimistic approximations differ. For this channel, uv, there exists an integer j, such that $\pi_{uv}(j) = \pi_{uv}(j+1)$, which, by Theorem 6.5, implies that actors, v_j and v_k, with $k = (j+1) \bmod_1 q_v$, may be safely contracted. Consequently, after a suitable retiming of v, we may safely apply a vectorisation transformation to v, which prunes v_j from the single-rate equivalent, by contraction. We may repeat this process until the cycle ratios of the optimistic and pessimistic approximations are equal, at which point we have computed the exact throughput of the MRSDF cycle.

Figure 6.9 illustrates this approach, by applying it to two different graphs. As a first example, consider the graph in Figure 6.9(a). Channel ab has a gain of $3/2$: the rates of the channel are such, that the first firing of a enables a single firing of b, and the second firing of a enables two more firings of b. Consequently, actor b may execute the second and third, out of every three firings, simultaneously. This is enforced by vectorising the actor with a blocking vector of $(1,2)$, which results in the graph CSDF depicted in Figure 6.9(b). Note that, in the CSDF graph, actor b has a constant execution time, which is why we do not explicitly depict an execution time vector. The optimistic and pessimistic approximations for the graph, obtained by applying Algorithm 6, are embedded in the figure, in the form of a pair of integers, associated with every channel. These integers give the number of initial tokens, on the corresponding channel, in the pessimistic and optimistic approximation, respectively. Both approximations of the graph of Figure 6.9(b) thus have the same cycle ratio, which is one. Multiplying this by the scalar invariant (which is six) of the MRSDF graph, thus gives the maximum cycle ratio, of the graph's single-rate equivalent, which is equal to six. The throughput of the graph is thus $1/6$.

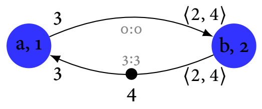
A second example is given in figures 6.9(c) through (e): Following the same reasoning as we did in the previous example, actor d in Figure 6.9(c) can be vectorised, with a blocking vector of $(1,2)$. This gives the graph shown in Figure 6.9(d). The optimistic and pessimistic single-rate approximations of this graph differ. This is due the channel dc: in the optimistic approximation, this channel has six tokens, whereas in the pessimistic approximation it has only three. By the contrapositive of Theorem 6.4, this implies that actor c can be vectorised without affecting the graph's throughput. Indeed, after simulating two firings of c, we have $\pi_{dc}(1) = \pi_{dc}(2)$, and, by Theorem 6.5 we may thus safely group every two consecutive firings of c into a single firing. That is, we may vectorise actor c with a blocking vector of (2), which results in the graph shown in Figure 6.9(e). The optimistic and pessimistic approximations for this graph are identical, and have a cycle ratio of $1/2$. Again, by multiplying this by $\mathcal{N}$ we learn that the maximum cycle ratio of the single-rate equivalent of Figure 6.9(c) is three (and thus the throughput is $1/3$).

6.3.3 Incremental throughput analysis of graphs

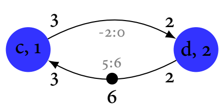
We now move from the incremental analysis of cycles to the incremental analysis of SDF graphs. In the previous section, we have seen how certain vectorisations



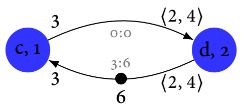
(a) MRSDF graph.



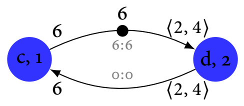
(b) Shift-invariant CSDF graph.



(c) MRSDF graph.

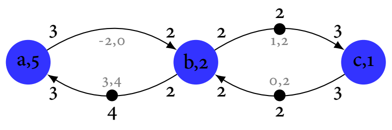


(d) Vectorised graph.

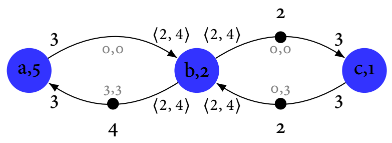


(e) Shift-invariant graph.

Figure 6.9 – By repeatedly applying the vectorisation transformation, a graph is obtained for which the pessimistic and optimistic approximations are exact. The throughput of an MRSDF cycle may thus be analysed without transforming it into its single-rate equivalent.



(a) MRSDF graph.



(b) Vectorised b.

Figure 6.10 – An example of a graph for which the incremental analysis of its cycles is not independent.

of an actor, and the corresponding contractions in the single-rate equivalent, are *safe* in the sense that they do not affect the throughput of the given cycle. These transformation may however affect the throughput of other cycles. Consider, for example, the graph shown in Figure 6.10. Again, we represent the single-rate approximations of the graphs by annotating the tokens on the corresponding channels in the respective approximations: the annotation x, y means that in the pessimistic and optimistic single-rate approximations, the corresponding channels have x and y initial tokens, respectively.

Maximum cycle ratio of the pessimistic approximation of the graph of Figure 6.10(a) is attained by cycle aba. If we treat this cycle in isolation, then the vectorisation of b with blocking vector $(1, 2)$ is safe (following the same reasoning as the example we described in the previous section). In the resulting graph (shown in Figure 6.10(b)), cycle bcb is now deadlocked: because the second firing of b now requires that four tokens are available on channel cb, it can never become enabled. This is also expressed in the single-rate approximations of the graph: the number of tokens in cycle bcb, of the pessimistic approximation, is negative, which indicates a possible deadlock.

Given a cycle C_1 in an MRSDF graph, the fact that a vectorisation of an actor is safe is based on the assumption that the throughput of the MRSDF graph is fully determined by C_1. As the example has shown, this assumption may not hold. The throughput of the MRSDF graph may be fully determined by *another* cycle. This cycle may be simple, or may be a closed walk, which is a cycle in which an actor is traversed multiple times. If the throughput is attained by another *simple* cycle, C_2, then, at some point during the incremental analysis of C_1, maximum cycle ratio of the pessimistic approximation of the graph will be attained by the cycle corresponding to C_1. Furthermore, if cycles C_1 and C_2 do not share an actor, then the incremental analysis of C_1 and C_2 is *independent*: vectorisation of an actor in C_1 does not change the throughput of C_2, and vice versa.

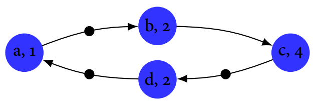
In case two cycles share one or more actors, then their incremental analysis is not independent. This was shown in the example above: vectorising an actor may leave one cycle's throughput unaffected, while decreasing the throughput of another. In the remainder of this section, we present an approach that copes with this problem by *unfolding* those actors for which a vectorisation is *not* safe.

If an actor is shared by two (or more) cycles of an SDF graph, then, in the single-rate equivalent of that graph, HSDF actors corresponding to firings of that actor may lie on several cycles. When treating one of the cycles in isolation, and transforming it in the way described in the previous section, we consider only a subset of these cycles. The transformation may correspond to a contraction of two actors, which both lie on the same simple cycle, into a single actor. As a result, the cycle C is transformed into a closed walk, W, in the single-rate equivalent of the vectorised graph. The cycle ratio, of each of the simple cycles that compose W, may be higher or lower than the cycle ratio of C. However, one of the composing cycles must have a cycle ratio that is at least the cycle ratio of C. This is illustrated by Figure 6.11, and stated formally by the following lemma.

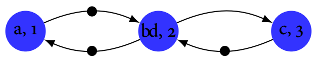
Lemma 6.4 (Contraction and maximum cycle ratio). *Let $\mathcal{H}$ be an HSDF graph, and let v and w be two actors in $\mathcal{H}$. Also, let $\mathcal{H}^c$ be the graph obtained from $\mathcal{H}$ by* contracting *v and w into a single actor z.*

The maximum cycle ratio of $\mathcal{H}^c$ is at least the maximum cycle ratio of $\mathcal{H}$.

Proof. Note that the lemma holds if the contraction of any two actors v and w in a cycle C introduces a cycle C^c with $\lambda(C^*) \geq \lambda(C)$. We prove this sufficient condition as follows.



(a) HSDF graph.



(b) Contraction of b and d.

Figure 6.11 – Contracting two actors, which lie on the same simple cycle, into a single actor, transforms the cycle into a closed walk that is composed of two simple cycles. The cycle ratio of one these simple cycles must have a cycle ratio that is at least the cycle ratio of the original cycle.

Let P_1 be a (simple) path in $\mathcal{H}$ that starts in v and terminates in w. Furthermore, let P_2 be a (simple) path from w to v in $\mathcal{H}$. The path that is formed by concatenating P_1 and P_2 is a (simple) cycle, which we denote by C. Denote by $|P|_w$ the weight of a path P, and by $|P|_\delta$ the number of tokens on P. The lemma may now be stated as:

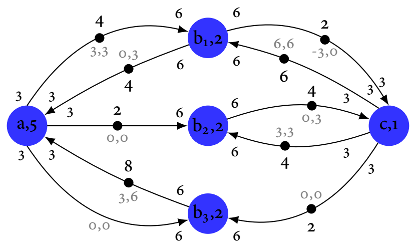
$$\max\left\{\frac{|P_1|_w}{|P_1|_\delta}, \frac{|P_2|_w}{|P_2|_\delta}\right\} \geq \frac{|P_1|_w + |P_2|_w}{|P_1|_\delta + |P_2|_\delta} = \frac{|C|_w}{|C|_\delta}, \tag{6.2}$$

which we prove by contradiction. Assume that (6.2) is false. Then we must have both $|P_1|_w\,|C|_\delta < |P_1|_\delta\,|C|_w$ and $|P_2|_w\,|C|_\delta < |P_2|_\delta\,|C|_w$. These two inequalities imply the following two inequalities:

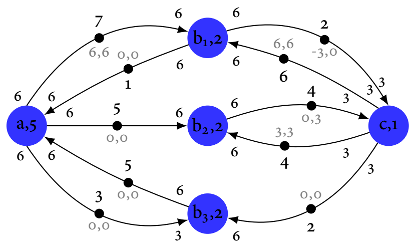
$$\frac{|P_1|_w}{|P_1|_\delta} < \frac{|C|_w}{|C|_\delta} \Rightarrow \frac{|P_1|_w}{|P_1|_\delta} < \frac{|P_2|_w}{|P_2|_\delta} \qquad \wedge \qquad \frac{|P_2|_w}{|P_2|_\delta} < \frac{|C|_w}{|C|_\delta} \Rightarrow \frac{|P_2|_w}{|P_2|_\delta} < \frac{|P_1|_w}{|P_1|_\delta},$$

which contradict each other. This proves the lemma. □

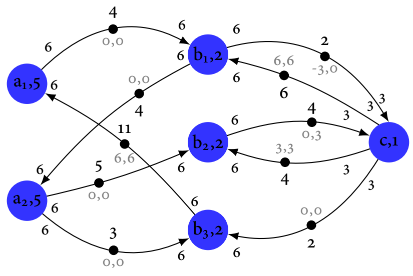
A result of the above lemma is that, if the throughput of an SDF graph is attained by a closed walk in the graph, the incremental analysis of one of the cycles that compose the walk will, at some point, cause another cycle in the walk to attain a higher pessimistic approximation. This was illustrated earlier in this section, in Figure 6.10. In the figure, there is a conflict in the firings of actor b that must be executed in parallel, in order to let each of the cycles attain its throughput. In cycle aba, actor b may be vectorised with blocking vector $(1, 2)$, but in cycle bcb, actor b must fire once before it may be vectorised with the same blocking vector. To resolve this conflict, we *unfold* the actor for which there is a vectorisation conflict, using the transformation presented in Chapter 4. The number of firings that we unfold is given by the sum of the elements of the blocking vector that was initially applied to the actor. For the example of Figure 6.10, we thus unfold b three times, using Algorithm 3. This results in the graph shown in Figure 6.12(a). The graph is again an MRSDF graph, the throughput of which we may analyse again by finding the



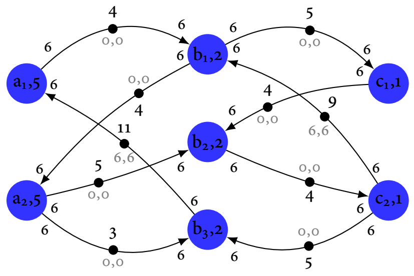
(a) Unfolded actor b, 7 ≤ λ ≤ 20.



(b) Retimed and vectorised a, ∞ ≤ λ ≤ ∞.



(c) Unfolded actor a, 14 ≤ λ ≤ 20.



(d) Fully unfolded, λ = 17.

Figure 6.12 – Resolving the conflict in safe vectorisations of actor b in Figure 6.10, by unfolding three firings of the actor.

cycle that is estimated to be critical, and perform an incremental analysis until we either find a conflict, or obtain the graph's exact throughput.

The above example gives rise to an incremental approach, where we use the optimistic and pessimistic single-rate approximations in the selection of cycles that are estimated to form bottlenecks, and their incremental analysis, and use transformations to unfold actors for which there is a vectorisation conflict. This approach is listed in Algorithm 10.

If we apply Algorithm 10 to the graph shown in Figure 6.12(a) (which was obtained from Figure 6.10 by unfolding actor b), then we find that its pessmistic and optimistic single-rate approximations have maximum cycle ratios of respectively $^{10}/_{3}$ and $^{7}/_{6}$. In the pessimistic approximation, the corresponding critical cycles are ab_1cb_3a and ab_2cb_3a, and, in the optimistic approximation, these are ab_1a and ab_3a. Multiplying the maximum cycle ratios of the two approximations by the scalar invariant of the MRSDF graph, which is six, gives that the maximum cycle ratio of the single-rate equivalent must lie between 7 and 20.

Algorithm 10: Computes the throughput of an MRSDF graph, by incrementally vectorising and unfolding actors.

```
input  : A consistent MRSDF graph $\mathcal{G}$, with scalar invariant $\mathcal{N}_{\mathcal{G}}$.
output : The throughput of $\mathcal{G}$.
repeat
    $\mathcal{H}_{\text{pess}}, \mathcal{H}_{\text{opt}} \longleftarrow$ pessimistic and optimistic single-rate approximation of $\mathcal{G}$
    $\check{\lambda} \longleftarrow$ Maximum cycle ratio of $\mathcal{H}_{\text{opt}}$
    $\hat{\lambda} \longleftarrow$ Maximum cycle ratio of $\mathcal{H}_{\text{pess}}$
    $C_{\text{pess}} \longleftarrow$ critical cycle of $\mathcal{H}_{\text{pess}}$
    Let $C$ be the cycle in $\mathcal{G}$ that corresponds to $C_{\text{pess}}$ in $\mathcal{H}_{\text{pess}}$
    if there is an actor $v$ in $C$ that was vectorised then
        Let $B_v$ be the blocking vector of $v$
        Undo the retiming of $v$
        Undo the vectorisation of $v$
        Unfold actor $v$ into $\sum_i B_v[i]$ actors using Algorithm 3
    else if $C$ has a channel $vw$ with $\frac{p^{\Sigma+}_{vw}}{\varphi_v} > \frac{p^{\Sigma-}_{vw}}{\varphi_w}$ then
        Retime and vectorise $w$ using Algorithm 9
    else if $C$ has a channel $vw$ with $\delta_{\hat{v}\hat{w}} > \delta_{\check{v}\check{w}}$ then
        Retime and vectorise $w$ using Algorithm 9
until $\hat{\lambda} = \check{\lambda}$
return $(\mathcal{N}_{\mathcal{G}}\hat{\lambda})^{-1}$
```

We may improve the throughput estimation of one of the two cycles that are critical in the pessimistic approximation. We may either vectorise a, or c. Let's choose a. It may be vectorised by applying blocking factor 2, which results in the graph shown in Figure 6.12(b). Due to the vectorisation, the two channels in cycle ab_3c now have insufficient tokens to enable their consumer. As a result, the cycle is deadlocked. This is reflected in the single-rate approximations: in both the optimistic and the pessimistic single-rate approximation, the corresponding cycle has zero tokens, which gives a maximum cycle ratio of ∞.

Since, due to the vectorisation of a, cycle ab_3c became critical, there is conflict in the assigned grouping of firings of a. We thus undo the retiming and vectorisation of a, and unfold the actor into two actors. This gives the graph shown in Figure 6.12(c), the single-rate approximations of which have maximum cycle ratios of 73 and 103: maximum cycle ratio of the pessimistic approximation equals $^{10}/_3$, and is attained by cycle $a_1b_1cb_3a_1$. The optimistic pessimistic approximation has a maximum cycle ratio of $^7/_3$. Consequently, the graph's maximum cycle ratio lies between 14 and 20.

The only actor that we may vectorise in the graph of Figure 6.12(c), is actor c, with a blocking factor of two. However, doing so gives a graph in which cycle b_1cb_1 is deadlocked (not shown). We must therefore unfold actor c into two actors. The graph that we then obtain is *fully unfolded*: it represents a single iteration of the original graph, shown in Figure 6.10(a). For each channel in the graph, corresponding channels in the optimistic and pessimistic approximation have an equal number of initial tokens. This means that the throughput estimated from these graphs is exact. Maximum cycle ratio of the single-rate equivalent is equal to 17, and is attained by the critical cycle $a_1 b_1 a_2 b_2 c_2 b_3 a_1$.

6.4 Discussion

The classical approach to computing the throughput of an SDF graph, by analysing the maximum cycle ratio of its single-rate equivalent, is an approach that has been declared impractical in several papers [28, 42, 77, 102]. The argument used in these papers is the potentially huge increase in graph size, when one transforms an SDF graph into its single-rate equivalent. A positive result of this conviction is that it has lead to the development of several alternative approaches to the analysis of throughput, such as the approach based on a state-space exploration of a self-timed execution [42], or approximate methods such as those based on the assumption of periodic execution [13, 98]. An unfortunate effect of these alternatives, though, is that their apparent success has distracted research from a careful revision of apparently unsuccessful approaches.

In the chapter, we have restricted the analysis of throughput to MRSDF graphs, or to CSDF graphs in which actors have non-varying execution times. This does not restrict the scope of the presented methods, as CSDF graphs may be subject to these methods, by applying them to their multi-rate equivalents (see Chapter 4).

The single-rate equivalents, of MRSDF graphs, that we analyse are smaller than those analysed in the classical approach: they are obtained from the original graph by unfolding each actor as many times as it fires in a single graph iteration, following the transformation presented in Chapter 4. In the classical approach, an actor w_j, in the single-rate equivalent of graph $\mathcal{G}$, has a single incoming channel for every *token* that is consumed by the j^{th} firing of actor w in $\mathcal{G}$. In the single-rate equivalent that are subjected to maximum cycle ratio analysis, in our approach, each actor w_j has a single incoming channel for every incoming channel of w in $\mathcal{G}$. As algorithms, for computing the maximum cycle ratio of a graph, scale linearly in the number of edges in the graph, this decrease in the number of channels contributes to a more efficient throughput analysis.

The relatively small number of channels in single-rate equivalents reveals a specific *structure*. This chapter shows the potential of a thorough understanding of this structure. Section 6.2 introduces a novel approach to throughput analysis, which exploits this regular structure. Our method shows how the analysis of the single-rate equivalent may be restricted to one of its strongly connected components, which we refer to as the graph's *feeding* component.

In spite of efficient methods for computing a graph's throughput by analysing a graph's single-rate equivalent, scalability of these methods is limited, due to the fact that the transformation of a graph into its single-rate equivalent gives an exponential increase in size. Approximate methods, on the other hand, provide upper and lower bounds on a graph's throughput. This offers a *trade-off*, between the computational complexity and accuracy of the analysis. Existing methods, however, offer no ways to control this trade-off: if the computation of an exact result is computationally too demanding, then the only alternative is a, potentially too inaccurate, conservative approximation. Section 6.3 presents a novel *incremental* approach that addresses this problem. It is the first method that combines both exact and approximate analyses, and gives a practical application of the theory that we put forward in chapters 3, 4, and 5. The method allows for the analysis of graphs for which the single-rate equivalent is too large to be represented explicitly, by unfolding only relevant subgraphs of the single-rate equivalent.

In the following chapter, we compare the efficiency of our first approach to throughput analysis, by computing the maximum cycle ratio of the feeding component of the single-rate equivalent, with existing methods, on different classes of graphs. For the incremental approach, we evaluate its effectiveness in controlling the trade-off between approximation accuracy on the one hand, and keeping the analysed graph small, on the other hand.

7 Case studies

Abstract – *The proof of the pudding is in the eating. In this chapter, we therefore apply the theory presented in the previous chapters to two common problems in the design-space exploration of real-time systems. The first of these problems is* throughput analysis, *for which we introduced two approaches in Chapter 6. The second is the* optimisation of buffer capacities *under a throughput constraint. We compare our two approaches to throughput analysis with current state-of-the-art approaches by applying them to a number of benchmark sets. In particular, we include a particular benchmark set that has been used in literature, in a previous study, but in our evaluation leads to a conclusion that opposes results reported in literature. The experimental evaluation shows that our approach to the computation of throughput is, in terms of runtime, one to three orders of magnitude faster than state-of-the-art methods.*

We furthermore show how throughput analysis and buffer capacity optimisation may be approached in an incremental fashion, by applying the single-rate approximations of Chapter 5 to graphs that are iteratively expanded using the transformations of Chapter 4. The evaluation shows how unfolding specific actors in a graph increases the accuracy of the estimations derived from their single-rate approximations.

There are two common problems that are addressed by analysis techniques for SDF graphs. The first of these two is the computation of throughput, for which we described an approach in Chapter 6. A second problem, which we did not cover thus far in this thesis, is the determination of sufficient buffer capacities, such that a given throughput is attained. This problem occurs in the design-space exploration of stream-processing systems, where communication links have a bounded buffer capacity, as opposed to the unbounded buffer capacities of channels in an SDF graph.

We address these two related problems in different sections: Section 7.1 evaluates the efficiency of our two approaches to throughput analysis, and Section 7.2 describes how sufficient buffer capacities may be conservatively estimated using the theory presented in Chapter 4 and Chapter 5.

7.1 Throughput analysis

The throughput of a system is an important performance characteristic. It expresses the average amount of work that the system can perform, per unit of time. For HSDF graphs, throughput is equal to the inverse of the graph's *maximum cycle ratio*, which may be computed in polynomial time, using well-established algorithms [19, 29, 49, 54, 78, 101]. For non-homogeneous SDF graphs, the *classical approach* to throughput analysis is to compute the maximum cycle ratio of the *single-rate equivalent* [39, 51]. The efficiency of this approach depends on the size of the latter. Since the transformation of a graph into its single-rate equivalent has an exponential complexity [77], scalability of the classical approach is limited. This has lead to the introduction of several new approaches to throughput analysis. The most notable of these approaches is the *state-space exploration* (SSE) method presented in [42] and [41]. This method operates by simulating a self-timed execution of the SDF graph, which eventually settles in a periodic phase. The method identifies this periodic phase from the firing times of the actors in the graph, and the distribution of tokens over the channels. A potential problem of the state-space exploration method is that the *transient phase*, which is the phase that precedes the periodic phase, may be very long. In an experimental study, presented in [42], state-space exploration was compared with the classical approach, and the former was found to convincingly outperform the latter. The conclusion of the evaluation was clear: in the classical approach, the construction of the single-rate equivalent formed the bottleneck in the analysis: for some graphs, the construction of their single-rate equivalents took more than half an hour.

Since the approach to throughput analysis that we presented in Chapter 6 is similar to the classical approach, in the sense that we analyse the single-rate equivalent, we repeat the study of [42], and include our approach in the evaluation. In line with the experiment reported on in [42], we must thus expect that our method will perform less well on graphs for which the single-rate equivalent is relatively large. We furthermore include the approach introduced in [35] and later described in more detail in [36]. This method, which we shall refer to as the *timestamped tokens* (TT) approach, simulates the self-timed execution of only a *single* graph iteration. During this iteration, the temporal dependencies between tokens are recorded (see Section 2.3.1) as a max-plus linear system, which can subsequently be analysed by standard tools. Especially for graphs that contain relatively few tokens, this method may be very effective, although currently, literature lacks an experimental evaluation. We compare our approach with the state-space exploration (SSE) and the timestamped tokens (TT) method in Section 7.1.2.

Our second approach to throughput analysis, presented in Section 6.3, avoids the costly transformation of the graph into its single-rate equivalent, by vectorising actors and performing only *partial* transformations (see Section 6.3 for more details). As this is a new approach, a comparison with state-of-the-art methods can not be performed. Instead, we evaluate its effectiveness on increasing the approximation accuracy, while keeping the graph as small as possible. We report on this experiment in Section 7.1.3.

Before we describe the experiments and their results, we describe the benchmark sets that we used in the experiments, in the following section.

7.1.1 Benchmark sets

The sets of graphs that we use to evaluate our approaches to throughput analysis, consist of four benchmark sets used in the previous study of [42], which was used to compare the classical and state-space exploration approaches. Of these four benchmark sets, the first test set consists of five models of stream-processing applications: an *h.263 decoder* [42, 89], *modem* [10], *MP3 decoder* [42, 89], *sample rate converter* [10], and *satellite receiver* [81].

Each of the other three sets used in the previous study consists of one hundred randomly generated graphs. The generation of these graphs is parameterised in such a way that each of the three sets represents a distinct class of MRSDF graphs. The three sets are the following:

Long transient contains HSDF graphs, which may have a long transient phase and thus pose a difficulty for state-space exploration techniques. Standard combinatorial algorithms for computing the maximum cycle ratio perform efficiently on these graphs.

DSP-like contains MRSDF graphs that mimic digital signal processing applications. Actors in these graphs have small rates, as well as small execution times, which makes the graphs representative for SDF graphs of DSP applications.

Large HSDF contains MRSDF graphs in which channels have relatively high production and consumption rates. Consequently, their single-rate equivalents tend to be several orders of magnitudes larger. In [42], the classical approach showed poorest performance when applied to graphs from this set.

All of the three test sets were generated using the state-of-the-art tool SDF^3, described in [90], and available from [1].

Because the generation of random MRSDF graphs, using SDF^3, is rather slow, especially for larger graphs, we have used the theory put forward in Chapter 5 to generate another 5000 random MRSDF graphs, of different sizes and with different characteristics. The structure of these graphs, and the properties of their actors and channels, are controlled by four parameters. The number of actors in a graph is given by the parameter n, and the *expected* number of channels by $E(m)$. Each actor has a repetition vector entry with the same expected value, given by $E(q)$. Finally, the number of tokens in each of the graphs is controlled by parameter p.

The procedure that we follow to generate these graphs, and the role of these four parameters, is as follows: we first generate the *structure* of the MRSDF graph, by generating a directed graph using the Erdős-Rényi model, which means that for each pair of vertices v and w, a directed edge vw is included in the graph with a fixed probability (see [4] for more details). This procedure gives a graph for which the number of vertices and the expected number of edges is fixed. Rather than controlling the probability of including an edge, we control the expected number of edges by fixing $E(m)$. The three combinations that we use are $n = 25, E(m) = 50$, $n = 25, E(m) = 200$, and $n = 50, E(m) = 400$. As a final step in this procedure, the graph is made strongly connected by adding a directed cycle that connects each of the graph's strongly connected components.

After generating a directed graph, we assign repetition vector entries to the graph's vertices, by drawing, for each vertex, an integer from, depending on the choice for the parameter $E(q)$, either the uniform distribution $U(25, 375)$, or $U(25, 975)$. This controls the size of the graph's single-rate equivalent: as a result of the assignment of repetition vector entries, the single-rate equivalents have an expected size (i.e., number of actors) of either $E(q) = 200n$ or $E(q) = 500n$. The directed graph is subsequently transformed into an MRSDF graph, by assigning rates to channels such that they correspond to the assigned repetition vector entries. Furthermore, actors are assigned a random (integer) execution time from the uniform distribution $U(2, 48)$.

The generation of a random MRSDF graph concludes with the distribution of initial tokens in the graph. Rather than controlling the number of tokens that are present in the graph directly, we control the graph's throughput by varying the tokens: by increasing the number of tokens placed on channels in cycles of a graph $\mathcal{G}$, the maximum cycle ratio of the graph's *pessimistic single-rate approximation* (see Chapter 5), $\mathcal{H}$, increases, which corresponds to an increase in the *lower bound* on the throughput of $\mathcal{G}$. We use the parameter p to control the graph's lower bound on throughput. If the sum of the execution times of the actors of a graph $\mathcal{G}$ is equal to W, then we assign as few initial tokens as possible to channels of $\mathcal{G}$, such that the lower bound on the throughput of $\mathcal{G}$ is equal to W/p. For $p = 0$, we add tokens to guarantee that the graph is deadlock free.

The process followed to ensure that the generated MRSDF graph $\mathcal{G}$ has a throughput of at least $\mathcal{N}_{\mathcal{G}}\lambda$, is the following:

1. We compute the maximum cycle ratio, λ^*, and the associated critical cycle, C^*, of the pessimistic approximation, $\mathcal{H}$, of $\mathcal{G}$.
2. If $\lambda^* > \lambda$, we add as few tokens as necessary to a single channel in C^*, such that the cycle ratio of C^* is at least λ, and repeat the process. The channel to which we add tokens is the one that has the highest associated flow normalisation vector (see Section 2.1.3).
3. If $\lambda^* \leq \lambda$, then, by Theorem 5.1, graph $\mathcal{G}$ has a throughput of at least $\mathcal{N}_{\mathcal{G}}\lambda$, and the process terminates.

Using the procedure above, we have generated 20 sets of graphs, using 20 different combinations of the parameter values. These 20 sets consists of the following five different combinations of the parameters n, $E(m)$ and $E(q)$:

25-Sparse-200 $n = 25$, $E(m) = 50$, and $E(q) = 200$.

25-Sparse-500 $n = 25$, $E(m) = 50$, and $E(q) = 500$.

25-Dense-200 $n = 25$, $E(m) = 200$, and $E(q) = 200$.

50-Sparse-200 $n = 50$, $E(m) = 200$, and $E(q) = 200$.

10-Large-HSDF $n = 10$, $E(m) = 50$, and $E(q) = 10,000$.

From each of the above five parameter combinations, we obtain four sets of 250 graphs each, by combining the three parameters with $p = 0$, $p = 1$, $p = 4$ and $p = 16$. This gives a total of 5000 graphs. Note that, especially for the "sparse" graphs, the average number of channels in the graphs in the set may be higher than $E(m)$, due to the fact that extra channels are added in order to make the graph strongly connected.

7.1.2 Analysis of the single-rate equivalent

Our first approach to throughput analysis, which we described in Section 6.2, analyses the single-rate equivalent of a graph, for its maximum cycle ratio. There are two differences between our approach and the classical approach that was evaluated in [42]. First of all, the single-rate equivalents that we construct (using Algorithm 5, see Chapter 4) typically contain fewer channels than those constructed in the classical approach. In our approach, each HSDF actor in the single-rate equivalent has as many incoming channels as its corresponding SDF actor has in the original graph. In the classical approach, on the other hand, the in-degree of an HSDF actor is equal to the number of *consumed tokens* per firing.

A second difference between our approach and the classical approach is that we restrict the analysis of the single-rate equivalent to a *feeding component* (see Theorem 6.3) of the graph. In the worst case, the single-rate equivalent may be strongly connected, in which case the restriction to its feeding component does not give any benefit.

We have implemented our approach in C, following the procedure listed as Algorithm 8 in Chapter 6. We compute the throughput of an MRSDF graph, $\mathcal{G}$, by analysing the feeding component of the single-rate equivalent $\mathcal{H}$ of $\mathcal{G}$, for its maximum cycle ratio. To compute the feeding component, we use the well known algorithm of Tarjan [92]. Tarjan's algorithm finds strongly connected components in *reversed topological order*: no strongly connected component is identified before any of its successors. Since we perform the algorithm by following edges in reverse, we essentially run Tarjan's algorithm on a *transposed* graph. Consequently, the first strongly connected component that is found by the algorithm is the feeding component.

After identifying the feeding component, S, of $\mathcal{H}$, we compute the throughput of $\mathcal{G}$ by computing the *maximum cycle ratio* of S (recall that the throughput of a graph is the inverse of the maximum cycle ratio of its single-rate equivalent). The algorithm that we use to compute the maximum cycle ratio of S is the method based on longest parametric paths, described by Young, Tarjan and Orlin in [101], and experimentally evaluated in [29]. Note that the method presented in [101] computes a graph's *minimum*, rather than its *maximum* cycle ratio, but the adaptation of the method to compute a maximum cycle ratio is straightforward (see [29] for more details).

An important aspect of our implementation is that we do not store the *channels* of the graph's single-rate equivalent, but rather *compute* them. Each actor w_j in $\mathcal{H}$ has one incoming channel for each incoming channel of the corresponding actor, w in $\mathcal{G}$ (see also Section 4.3). The source actor v_i, of the incoming channel $v_i w_j$ of w_j, is determined from the predecessor function associated with channel vw: $i = \tilde{\pi}_{vw}(j)$ (see Chapter 6 for details). If the single-rate equivalent $\mathcal{H}$ is a dense graph with n actors, then avoiding the storage of the channels of $\mathcal{H}$ reduces the memory used to store the graph data structure by a factor of n. Although this reduction in memory is traded for an extra computational effort, given the fact that the size of the single-rate equivalent is exponential in the size of the SDF graph, we believe that the trade-off is advantageous to the overall running time of the implementation.

Throughout the remainder of this section, we refer to the implementation of our approach as YTO, which is an acronym for Young, Tarjan and Orlin's algorithm [101], which we use to compute the maximum cycle ratio. In the previous study, the same algorithm (and others) was used in the context of the classical approach.

We compare the running time of our algorithm with the following two algorithms offered by SDF^3:

throughput (SSE) Computes the throughput of a graph by exploring the state-space of a simulated self-timed execution, until a periodic phase is found. This is the method that is described and evaluated in [42].

mpthroughput (TT) Executes a single graph iteration, during which the temporal dependencies between the arrival times of tokens is recorded and captured by a max-plus system [35, 36]. The max-plus system has a single recurrence relation for each token in the graph. After constructing the max-plus system, its eigenvalue of the max-plus system is then computed, which gives the inverse of the graph's throughput.

We refer to the first algorithm as SSE (for state-space exploration), and to the second algorithm as TT (for timestamped tokens).

The first test that we evaluated the three methods, YTO, SSE, and TT, on, is the set of five graphs that model stream-processing applications. For each graph, we measured the runtime required to compute the throughput. The measured results are

		h.263 decoder	modem	mp3 decoder	sample rate conv.	satellite
Statistics	actors	4	16	13	6	22
	channels	10	54	37	16	74
	tokens	4757	54	25	38	1564
	HSDF: actors	4754	48	13	612	4515
	SCC: rel. actors	1.0	1.0	1.0	1.0	1.0
	SCC: rel. channels	0.75	0.80	1.0	0.58	0.72
YTO	avg (ms)	1.2	< 0.01	< 0.01	0.18	0.91
	std dev (ms)	0.58	< 0.01	< 0.01	0.39	0.29
SSE	avg (ms)	3.1	0.36	0.091	0.73	0.43
	std dev (ms)	1.0	0.48	0.29	0.62	0.64
TT	avg (ms)	1.4	0.18	0.27	0.18	1.2
	std dev (ms)	0.64	0.39	0.45	0.39	0.39

Table 7.1 – Statistics and running times of the three modes, on the five graphs that model stream-processing applications.

listed in Table 7.1. Each graph was analysed ten times, the table reports the average (avg) and *sample* standard deviation (std dev), of the running time, in milliseconds (ms). Furthermore, the table reports the statistics of the graphs. Numbers of tokens are listed in order to assess the effectiveness of TT in reducing the size of the analysed system. The number of actors in the full single-rate equivalent is given in the row "HSDF: actors". The final two rows that report on statistics give the relative size of the feeding component, with respect to the size of the full single-rate equivalent obtained in the classical approach.

Results for this first test set show that all three algorithms require a runtime of at most a few milliseconds to compute the throughput of each of the five graphs in the set. The hardest of the five graphs seems to be the one that models the h.263 decoder, even though this graph is the smallest. The density of the graph, however, is relatively high, which may cause the state-space exploration method to require a relatively long time to reach the periodic phase.

If we compare the results listed in Table 7.1 to those reported in the previous study of [42], then the runtime of our approach (YTO) forms a notable exception. In the previous study, the classical approach only completed the modem and MP3 decoder within the time limit of half an hour. For the modem, the classical approach required 81 milliseconds, and the MP3 decoder took one millisecond. State-space exploration, on the other hand, required one millisecond for the modem and sample rate converter, four milliseconds for the satellite receiver, eleven for the MP3 decoder, and four seconds for the h.263 decoder. Although our approach improves over the classical approach by limiting the size of the graph that is analysed, this can not fully explain the extreme improvement in running times over those observed in the previous study.

		Long transient	DSP-like	Large HSDF
YTO	N	100	91	46
	avg (ms)	< 1	1	8
	std dev (ms)	< 1	< 1	< 1
SSE	N	100	100	100
	avg (ms)	912	< 1	< 1
	std dev (ms)	277	< 1	< 1

Table 7.2 – Average and standard deviations in the running times of state-space exploration (SSE), and the algorithm of Young, Tarjan and Orlin (YTO), as reported in the previous study of [42]. Running times reported for YTO exclude the time required for the transformation of the graphs into their single-rate equivalents.

To the best of our knowledge, this is the first experimental evaluation of throughput analysis using the timestamped token approach. From Table 7.1, we conclude that it is on par with both our approach and state-space exploration. Note that the measured sample standard deviations are too high to declare any of the three methods a clear winner.

The other three test sets that were used in the previous study, consist of randomly generated graphs. In the previous study, especially the "Large HSDF" set proved to be problematic for the classical approach: out of the 100 graphs, ten could not be transformed, into their single-rate equivalents, within 30 minutes. For the 90 graphs that could be transformed, the transformation took, on average, two seconds. In other words, the transformation from the graph into its single-rate equivalent formed a clear bottleneck in the procedure. State-space exploration, on the other hand, completed the analysis of the graphs in the "Mimic DSP" and "Large HSDF" sets, within a millisecond, on average. For the HSDF graphs in the "Long transient" set, state space exploration required more time, which was due to the longer transient time for these graphs. After transforming the graph into its single-rate equivalent, the classical approach used in the previous study applied three different algorithms, one of which (YTO) is the same method we have used in our implementation. Relevant results of the previous experiment are listed in Table 7.2.

We have repeated the experiment, including our approach and the timestamped token approach of [35, 36]. Statistics for the test sets, and running times for the algorithms, are listed in Table 7.3.

From the table, we observe several differences with the results collected in [42]. While the previous study reported that 10 of the graphs in the "Large HSDF" graph could not be transformed into its single-rate equivalent within the time limit of 30 minutees, our method performed this transformation, *and* the subsequent maximum cycle ratio analysis, for each of the 100 graphs, within a few milliseconds.

A further striking result, obtained from the statistics reported by our method, is

		Long transient	DSP-like	Large HSDF
Statistics	actors	284	20	13
	channels	359	24	21
	tokens	48	500	41578
	HSDF: actors	284	1008	8166
	SCC: rel. actors	1.0	0.083	0.012
	SCC: rel. channels	1.0	0.54	0.25
YTO	avg (ms)	0.010	< 0.01	0.060
	std dev (ms)	0.10	< 0.01	0.51
SSE	avg (ms)	420	1.1	71
	std dev (ms)	130	4.6	520
TT	avg (ms)	2.2	1.7	730
	std dev (ms)	0.78	11	5200

Table 7.3 – Statistics of the three sets of randomly generated graphs, used in [42], and running times of the three different algorithms.

that for *each of the two hundred* MRSDF graphs (that is, excluding the HSDF graphs in *long-transient*), the size of the feeding component, of the graph's single-rate equivalent, is equal to the size of the original graph. As a consequence, the HSDF graph that is analysed for its maximum cycle ratio is extremely small, compared to the classical approach in the previous study, and thus the running time of our approach is minimal, on each of the three test sets. In particular, the time required to analyse each of the graphs in "DSP-like" fell below the resolution of the timer.

Because of this striking result, we have examined the graphs in the sets "Long transient" and "Large HSDF" in more detail. As it turns out, for each of the cycles in the 200 graphs, the total weighted (by the flow normalisation vector) number of tokens in the cycle, is an integer multiple of the graph's scalar invariant. That is, the number of tokens on each cycle is such that each actor v may fire precisely q_v times in parallel, removing all tokens from its incoming channel onto its outgoing channels. This means that the randomly generated MRSDF graphs behave as HSDF graphs, if (after a suitable retiming) we apply a vectorisation of each actor v, with a blocking factor q_v.

From the above we conclude that the benchmark sets used in [42] are not very suitable for drawing conclusions, and biased towards a positive outcome for the state-space exploration method. To obtain a fair comparison between the three methods, we therefore also applied our approach and the two methods offered by SDF^3 to the 20 benchmark sets we generated, using the procedure described in Section 7.1.1.

Table 7.4 shows the results for the four sets generated from set "25-Sparse-200". For the analysis of graphs in this set, we set a time limit of one minute. Graphs that could not be analysed within this time limit, have been omitted from the results.

		p = 0	p = 1	p = 4	p = 16
Statisics	Channels	61.3	61.5	60.9	60.7
	Tokens	12655	239980	843127	3661108
	HSDF size	12517	12496	12556	12644
	avg HSDF actors	0.80	0.80	0.79	0.80
	avg HSDF channels	0.0028	0.0029	0.0029	0.0028
YTO	N	250	250	250	250
	avg	1.5	2.4	4.3	5.1
	std dev	0.60	0.94	1.4	1.7
SSE	N	250	250	250	249
	avg	72	73	228	967
	std dev	22	175	1.3×10^3	3.7×10^3
TT	N	250	250	249	226
	avg	85	631	2.9×10^3	12×10^3
	std dev	52	2.0×10^3	5.7×10^3	11×10^3

Table 7.4 – Statistics of the graphs, and running times for the three methods, measured on the set “25-Sparse-200”.

For each of the three methods, the row denoted “N” reports the number of graphs that could successfully be analysed within the time limit.

Our method is the only one that was able to complete the analysis of each of the 1000 graphs within the time limit. The table furthermore shows that running times for our algorithm are low, compared with those observed for state-space exploration and the timestamped token approach: on average, our approach (YTO) is between 30 and 189 times faster than state-space exploration, and the latter is faster than the timestamped token approach. Moreover, the standard deviation of the running time of our approach remains very low compared to the other methods.

The running time of all three approaches increases as the graphs contain more tokens: for our approach, this increase is modest: for $p = 16$, running times are about three and a half times as high as for $p = 0$. For the state-space exploration method, running time increases by a factor of more than twelve, and the timestamped token approach suffers most from the increase in tokens, which is to be expected.

If we increase the average repetition vector entries, such that the single-rate equivalent is about two and a half times larger, then we observe that all three methods see an increased running time, of at least that factor: Table 7.5 shows the results obtained for the set “25-Sparse-200”. The table shows that, similar to the set “25-Sparse-200”, as the number of tokens in the graphs increases, running times increase for all three methods, but both the increase in the average and standard deviation, of the running times for the algorithms implemented in SDF^3, is significantly steeper than for our algorithm.

In the third class of generated graphs, set “25-Dense-200”, graphs have the same

		p = 0	p = 1	p = 4	p = 16
	Channels	61.3	61.5	60.9	60.7
	Tokens	12655	239980	843127	3661108
Statisics	HSDF size	12517	12496	12556	12644
	avg HSDF actors	0.80	0.80	0.79	0.80
	avg HSDF channels	0.0028	0.0029	0.0029	0.0028
	N	250	250	250	250
YTO	avg	4.8	8.5	13	15
	std dev	1.3	2.9	4.6	4.9
	N	250	250	249	233
SSE	avg	180	193	805	3003
	std dev	62	618	1960	7117
	N	250	246	196	87
TT	avg	520	4004	20970	37980
	std dev	147	5230	12100	11250

Table 7.5 – Statistics of the graphs, and running times for the three methods, measured on the set “25-Sparse-500”.

number of actors, but a higher number of channels: each of the 25 actors is, on average, connected to 20 other actors. Results for the four benchmark sets generated from this class are shown in Table 7.6. When comparing these results with those reported for the sparse graphs in Table 7.4, we see that all three methods have a higher average running time. Our approach is again the fastest method, and significantly outperforms the other two: our approach is between 29 and 470 times faster than state-space exploration. The timestamped token approach again suffers from an increase in tokens: for $p = 16$, it fails to analyse 112 of the 250 graphs within the time limit of one minute. An interesting observation from Table 7.6 is that, whereas the running times of our algorithm and the timestamped token approach increases as the number of tokens increase, the running time of state-space exploration decreases for $p = 1$ and $p = 4$, and then increases again for $p = 16$. We do not have a clear explanation for this observation.

In order to evaluate the scalability of our approach in the number of actors, the fourth class of graphs that we consider contains of sparse graphs that have 50 actors, and on average a little over 200 channels. Results for this class are shown in Table 7.7. If we compare these results with those for the graphs with fewer actors but (relatively) more channels, then we observe lower running times, which is to be expected considering that we doubled the number of actors, but multiplied the number of channels by 10. Interestingly, we again observe a decrease in the running time of state-space exploration, when moving from $p = 0$ to $p = 1$ and $p = 4$. Also, the timestamped token approach is again significantly faster on the set with $p = 0$. Again, our approach (YTO) is the clear winner for this class of graphs: it is between 24 and 41 times faster than the other two methods.

		p = 0	p = 1	p = 4	p = 16
	Channels	503	503	503	503
	Tokens	114999	1124422	4056866	15995505
Statisics	HSDF size	5023	5034	4982	4986
	rel. HSDF actors	1.0	1.0	1.0	1.0
	rel. HSDF channels	0.0070	0.0070	0.0071	0.0070
	N	250	250	250	250
YTO	avg (ms)	15	19	29	52
	std dev (ms)	1.8	4.1	7.5	29
	N	250	250	250	250
SSE	avg (ms)	7053	1122	860	1126
	std dev (ms)	1178	306	329	489
	N	250	250	250	138
TT	avg (ms)	798	1601	7050	38280
	std dev (ms)	203	627	3306	11170

Table 7.6 – Analysis running times for the three different methods, on MRSDF graphs of 25 actors, (expected) 50 channels, 5000 actors in the single-rate equivalent, and varying numbers of initial tokens.

		p = 0	p = 1	p = 4	p = 16
	Channels	207	204	205	205
	Tokens	34731	175379	597551	2410259
Statisics	HSDF size	9998	10002	10008	10025
	rel. HSDF actors	0.94	0.95	0.94	0.95
	rel. HSDF channels	0.0070	0.0071	0.0070	0.0069
	N	250	250	250	250
YTO	avg (ms)	7.4	9.7	15	23
	std dev (ms)	1.0	2.2	3.6	5.8
	N	250	250	250	250
SSE	avg (ms)	1655	449	363	578
	std dev (ms)	312	218	208	475
	N	250	250	250	244
TT	avg (ms)	306	724	2593	14220
	std dev (ms)	55	391	1605	7400

Table 7.7 – Analysis running times for the three different methods, on MRSDF graphs of 25 actors, (expected) 50 channels, 5000 actors in the single-rate equivalent, and varying numbers of initial tokens.

As a final experiment, we analysed graphs of which the single-rate equivalent is several orders of magnitude larger than the graph itself. This serves as a proper repetition of the study of the performance of state-space exploration and our approach, on a set like the “Large HSDF” set of [42]: these graphs form worst-case instances for our approach. Results are shown in Table 7.8. The table shows that running times of the state-space exploration are in line with those observed in the experiments for the other test sets, but running times for the timestamped token approaches are dramatic: for only 12 of the 1000 graphs, the algorithm completed within the time limit of one minute: the only graphs for which we can report the results of TT are those for which $p = 0$.

Our algorithm, on the other hand, completed all 1000 graphs within the time limit, outperforming the other two methods. However, if we compare the results for this set of graphs with the results reported above for the other sets, then we observe that the gap between our approach and state-space exploration has decreased significantly: for the set with $p = 1$, our algorithm is only about three times faster than SSE. For $p = 4$ and $p = 16$, the gap widens again. Note that, on these two sets, state-space exploration did not complete the analysis of 72 of the graphs within the time limit. This means that the reported sample standard deviation is, in reality, much larger.

We expect that, for graphs that have even larger single-rate equivalents, the gap between the average running time of SSE and our approach will become even smaller, better revealing the potential effectiveness of state-space exploration. However, extrapolating the observed standard deviation, we also expect that SSE will often perform on at least some of the graphs analysed. For graphs that are too large to be analysed efficiently by our approach, though, the *estimated* throughput will likely be quite accurate, and as such, the cost for performing an exact analysis will no longer outweigh the potential error of a rough approximation. This brings us to the next study that we cover in this chapter, which is the relationship between graph size and approximation accuracy.

7.1.3 Incremental analysis

Graphs, for which the single-rate equivalent is too large to be analysed efficiently, may be analysed for their throughput in an *approximate* way: the maximum cycle ratios, of the optimistic and pessimistic single-rate approximations of a graph, give upper and lower bounds on the graph’s throughput (see Theorem 5.1). Furthermore, both the construction and the analysis of the single-rate approximations may be done in time polynomial in the size of the graph, and are thus, computationally, relatively cheap compared to a time consuming exact analysis. The error of an approximation may be significant. If the error is unacceptably large, then we may apply the techniques presented in Chapter 6 to increase the approximation accuracy by *vectorising* actor firings, and *unfolding* actors into multiple actors, using the transformations given in Chapter 4. In this section, we evaluate the relation between the accuracy of the approximation, and the size of the graph. That is, we apply

		p = 0	p = 1	p = 4	p = 16
Statisics	Channels	52	52	52	52
	Tokens	4.4×10^5	4.3×10^8	1.8×10^9	6.7×10^9
	HSDF size	101313	97740	99939	100140
	rel. HSDF actors	0.98	0.97	0.98	0.98
	rel. HSDF channels	1.5×10^{-4}	1.6×10^{-4}	1.5×10^{-4}	1.6×10^{-4}
YTO	N	250	250	250	250
	avg (ms)	96	743	2.9×10^3	6.4×10^3
	std dev (ms)	21	401	4.0×10^3	15×10^3
SSE	N	250	250	247	181
	avg (ms)	1.1×10^3	2.2×10^3	12×10^3	34×10^3
	std dev (ms)	308	3.1×10^3	8.0×10^3	12×10^3
TT	N	12	-	-	-
	avg (ms)	4.8×10^4	-	-	-
	std dev (ms)	7.7×10^3	-	-	-

Table 7.8 – Graph statistics and algorithm running times for the test set that consists of graphs for which the transformation into a single-rate equivalent yields a graph that is around 10,000 times larger.

Algorithm 10 to incrementally analyse graphs for their throughput, and report on the *relative* size of the analysed graph.

We have implemented the procedure listed in Algorithm 10 in Java. In our implementation, we select actors that may be vectorised in the following way: if the critical cycle C^* contains channels with a positive gain, we select from these channels the channel vw for which q_v is minimal, and apply the vectorisation to w. If w was already vectorised during the analysis of another cycle than C^*, then we undo the vectorisation and retiming of w (see lines 10 – 13 of Algorithm 10). For the analysis of the optimistic and pessimistic single-rate approximations for their maximum cycle ratios we again used the algorithm of Young, Tarjan and Orlin, as we did in the previous section. Note that, whereas in the implementation of *exact* throughput analysis, which we evaluated in the previous section, we avoided storing the channels in the single-rate equivalent, we do not apply this optimisation in our implementation of the incremental analysis, due to the dynamic nature of the graph.

We evaluate the relation between accuracy and graph size on two of the test sets used in the previous section. The reason for not including more test sets, is that the accuracy of the approximation for the excluded sets is below one percent: for the sets with $p = 4$ and $p = 16$, the accuracy is even below one permille.

Graph size is expressed as a relative, rather than an absolute number: we report the number of actors of the analysed graph, divided by the number of actors in the graph's single-rate equivalent. Consequently, graph size is a number greater than zero, and at most one. In the worst case the entire graph needs to be unfolded into

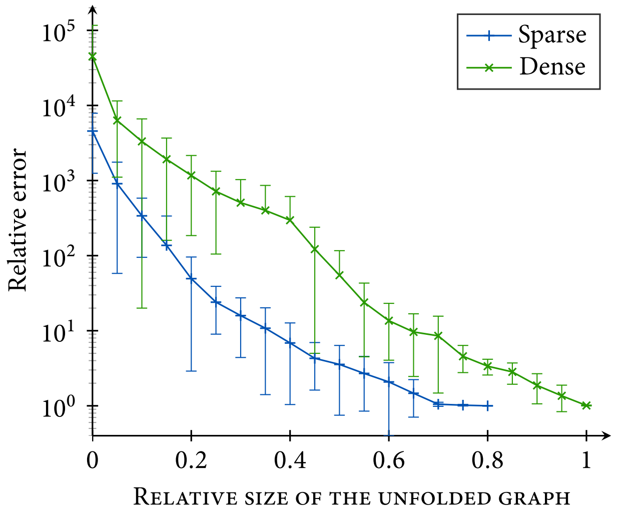


Figure 7.1 – Accuracy of the approximation versus size of the analysed graph.

its single-rate equivalent, in which case the reported graph size is one.

Approximation accuracy is defined as the relative difference between the graph's upper and lower bound on throughput, respectively computed from its optimistic and pessimistic single-rate approximation. This means that an exact approximation is reported as an accuracy of one. Figure 7.1 shows how the approximation accuracy decreases as the graph is unfolded.

The figure shows that the first throughput estimate has an extremely large error: for the sparse graphs, the maximum cycle ratio of the pessimistic single-rate approximation was, on average, around 5000 times as high as the maximum cycle ratio of the optimistic approximation. In order to decrease this error such that it falls below 100% (i.e., a relative error of 2), about 30% of the single-rate equivalent is constructed. For dense graphs, a significantly larger part of the single-rate equivalent must be constructed: almost 60%.

We furthermore observed that vectorisations of actors (lines 15 – 17 of Algorithm 10) was often corrected in a subsequent iteration of the algorithm. This was especially true for the dense graphs. Another observation that we made during the evaluation, is that the running time of our implementation of the incremental analysis is high. Whereas the running time for the exact analysis, reported in the previous section, is in the order of milliseconds, the running time for the incremental analysis is in the order of tens of seconds. We believe that this is primarily due to the fact that, single-rate approximations and their maximum cycle ratios are recomputed at every step in the approach: after vectorising or unfolding a single actor. Although the running time of these two procedures is in the order of tens to hundreds of

		Relative graph size				
	HSDF size	0.01	0.05	0.1	0.5	1.0
h.263 decoder	4757	2.0×10^3	2.0×10^3	2.0×10^3	2.0×10^3	1.0
modem	48	-	-	-	1.0	1.0
mp3 decoder	13	-	-	-	-	1.0
sample rate conv.	612	1.6	1.0	1.0	1.0	1.0
satellite	4515	1.1	1.0	1.0	1.0	1.0

Table 7.9 – Results showing relative errors, obtained from different graph sizes, for the incremental throughput analysis of the five graphs representing stream-processing applications.

milliseconds, their repeated application (the average number of steps required for the incremental analysis of the two test sets is around one hundred) adds up to several seconds. Also, the management of large and evolving graph data structures, as well as the runtime overhead of Java (compared to C), seems to give a further performance penalty.

Finally, the graphs that we have used to evaluate the incremental analysis are simply too difficult to analyse incrementally: because of the procedure used to generate the graphs in the test set, many cycles in the graph have a comparable throughput. As such, in the single-rate approximation, many different cycles are identified as potentially critical, and unfolded. Since the generated graphs we used may not be representative of graphs that occur in practice, as models of actual stream-processing applications, we have applied the incremental analysis to the five applications used in the previous section. Results are listed in Table 7.9.

The table shows varying results. Running times (which we do not show in the table) were in the order of a few hundred milliseconds, for all five graphs. Of the five graphs, only one graph (the h.263 decoder) needed to be unfolded into its full single-rate equivalent (the size of which is shown in the second column), in order to obtain an acceptable throughput estimation: up to the point where the graph is fully unfolded, the error in the estimation remains constant, at a factor of 2000. Note that the h.263 decoder consists of only four actors, which means that the increase in graph size per step is already relatively high.

For the other four graphs, the incremental analysis was more effective. No actors needed to be unfolded in the graphs representing the modem and sample rate converter. Since the modem has 16 actors, and its single-rate equivalent has 48 actors, an exact approximation is obtained for a relative size of $1/3$. The mp3 decoder is an HSDF graph, which means that the initial throughput estimation is exact.

For the graphs representing the sample rate converter and satellite receiver, the necessary unfolding was minimal: the initial rough approximation for these graphs, based on their single-rate approximations, is 60% and 10%. After only a few iterations, which, in both cases, unfold less than five percent of the full single-rate equivalent, an exact approximation was obtained.

To conclude the evaluation of our incremental approach to throughput analysis, we remark that its efficiency, in terms of avoiding the construction of most of the single-rate equivalent, very much depends on the graph that it is applied to: some graphs may have a clear performance bottleneck, in the form of cycles that have a significantly lower throughput than other cycles in the graph. For such graphs, the incremental approach quickly converges, as it only needs to unfold the bottleneck cycle. The graphs representing the modem, sample rate converter, and satellite receiver, are examples for which this is true, while the h.263 decoder has a less prominent bottleneck.

7.2 Buffer capacity optimisation

The second problem that we address as a case study, is the optimisation problem of determining sufficient *buffer capacities*, such that a given throughput is attained. This problem occurs in the design-space exploration of stream-processing systems, where communication links have a bounded buffer capacity, as opposed to the unbounded buffer capacities of channels in an SDF graph.

We take a simple CSDF model and transform it, using Algorithm 1, into an equivalent MRSDF graph, after which we can apply analysis techniques designed for MRSDF graphs. We compare the results obtained from this equivalent MRSDF graph with analysis results computed directly for the CSDF graph using a CSDF-specific algorithm.

We use a model of an MP3 playback application (see Figure 7.2), which has been used as a case study in several other studies [7, 98, 99]. The application consists of three tasks, each of which is modelled by a single CSDF actor: the MP3 decoder (modelled in Figure 7.2 by the actor labelled MP3) processes a 48 kHz variable bitrate MP3 file, and the sample rate converter (modelled by actor SRC) converts this to a 44.1 kHz stream to match the frequency of the digital-to-analog-converter, which is modelled by the actor labelled DAC. Communication channels between the tasks are FIFO buffers with a finite capacity, whereas channels in a synchronous dataflow graph have unbounded capacity. The approach that is often used to model the bounded capacity of a channel vw, is to add a *reverse* channel wv to the graph, the number of initial tokens of which indicates the capacity of the forward channel, vw. To leave these buffer capacities unspecified, the number of tokens on these reversed channels are captured by variables.

The worst-case execution time of the ten different phases of the sample rate converter task are (in order): 136577, 133824, 133760, 133750, 133748, 133863, 133844, 133955, 133882, and 133862 clock cycles. The MP3 decoder task has a worst-case execution of 1603621 clock cycles, and the digital-to-analog-converter samples its input every 5000 clock cycles. The latter gives a minimal required throughput; in order not to stall the digital-to-analog-converter, data should arrive in time.

The throughput of the graph is dependent on the capacity of the two buffers between the three tasks, i.e., the variables d_1 and d_2. If we approximate these buffer sizes

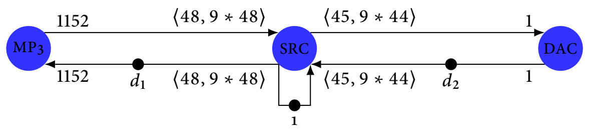


Figure 7.2 – A Cyclo-Static dataflow graph model of an MP3 playback application. The capacities of the buffers between the tasks are captured by (integer) variables d_1 and d_2. The compact notation $\langle 45, 9 * 44\rangle$ specifies the statically varying sequence "45, followed by nine times 44".

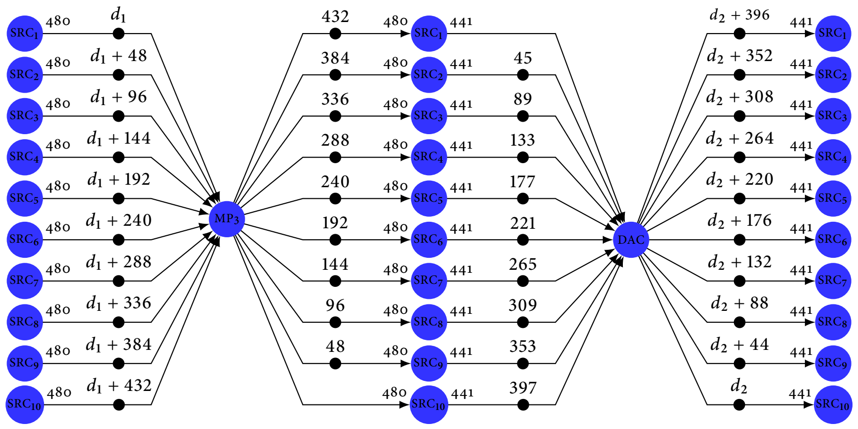

SRC1 … SRC10
480
d_1, $d_1 + 48$, $d_1 + 96$, $d_1 + 144$, $d_1 + 192$, $d_1 + 240$, $d_1 + 288$, $d_1 + 336$, $d_1 + 384$, $d_1 + 432$
MP3
432, 384, 336, 288, 240, 192, 144, 96, 48
441
45, 89, 133, 177, 221, 265, 309, 353, 397
DAC
$d_2 + 396$, $d_2 + 352$, $d_2 + 308$, $d_2 + 264$, $d_2 + 220$, $d_2 + 176$, $d_2 + 132$, $d_2 + 88$, $d_2 + 44$, d_2

Figure 7.3 – The MRSDF graph that is temporally equivalent to the CSDF graph from Figure 7.2.

required to reach the required throughput, using the techniques from [98], we find that d_1 must be at least 1536 and d_2 must be at least 90. Note that this algorithm is applied to the CSDF graph, and does not require a transformation of the graph into an equivalent MRSDF graph.

Transforming the CSDF graph into an equivalent MRSDF graph "unfolds" the sample rate converter into ten actors, each representing one of the ten phases of the CSDF actor. Because the other two actors in the CSDF graph are basically MRSDF actors, they do not need to be unfolded. The equivalent MRSDF graph is not easily represented in a compact way. We have therefore unfolded the cycles of the graph, such that all channels point from left to right. Furthermore, we omitted the self-loops and the production and consumption rates of the MP3 actor (which are 1152) and the DAC actor (which produces and consumes data with a rate of one). The resulting MRSDF graph is depicted in Figure 7.3.

If we analyse the equivalent MRSDF graph, using the techniques described in [97], for the minimum buffer sizes (i.e., d_1 and d_2), the results are the following: For the

Table 7.10 – Minimum buffer capacities (d_1 and d_2) and their sum, computed for different choices of *unfolding factors* for the three actors in the graph of Figure 7.2.

MP3	SRC	DAC	d_1	d_2	$d_1 + d_2$
1	10	1	1488	73	1561
1	10	3	1488	72	1560
1	10	21	1488	71	1559
1	10	49	1488	69	1557
1	10	147	1488	59	1547
1	10	441	1488	45	1533
5	10	441	1488	45	1533
5	30	1323	1440	45	1485
5	60	2646	1392	45	1437
5	120	5292	1152	45	1197

size of the buffer between the MP3 decoder and the sample rate converter, we find $d_1 = 1488$. This is an improvement over the buffer size found by the CSDF-specific algorithm by 3.1%. For the size of the buffer between the sample rate converter and the digital-to-audio converter we find $d_2 = 73$, which is an improvement of 18.9%.

An exact computation, using the tool SDF3 [90], yields that the required capacities d_1 and d_2 are respectively 1152 and 45. This means that the minimum buffer sizes, computed from the equivalent MRSDF graph, are 29.2% (d_1) and 62.2% (d_2) higher than the exact capacities. In order to improve the estimated capacities, we apply Algorithm 1 to transform *several periods* of the actors in the graph, rather than a single period. As the error in the buffer capacity estimates depends on the *greatest common divisor* of the channel's production and consumption rate, we choose the number of periods in such a way that these greatest common divisors increase. Table 7.10 lists the resulting buffer capacities, for different unfolding factors. The first six rows show that the capacity d_2 decreases from 73 to 45 (which equals the exact required capacity). Similarly, the last four rows show how the estimate for d_1 decreases from 1488 to the optimum, 1152. The three unfolding factors (5, 120 and 5292) in the last row correspond to the repetition vector entries of the three actors; this means that with these factors, Algorithm 1 produces an HSDF graph. The estimated buffer capacities for this final case match the optimal capacities computed using SDF^3. Note that, by unfolding the MRSDF graph further, the estimated sum of the capacities decreases by 23.3% from 1561 to 1197.

7.3 Discussion

The evaluation of throughput, by analysing the feeding component of a graph's single-rate equivalent for its maximum cycle ratio, clearly outperforms state-of-the-art methods. On the various test sets that we included in our study, our approach is between one to three orders of magnitude faster than the methods offered by the widely used state-of-the-art tool SDF^3. Furthermore, the evaluation of our approach on the benchmark sets that were used in the previous study reported in [42], calls for a revision of the conclusions drawn in that study. In [42], the classical approach was considered inefficient due to the transformation of a graph into its single-rate equivalent, our current study shows that this transformation may be performed in very little time, because we may *compute* the graph's channels, rather than represent them explicitly.

Furthermore, the graphs that we have included in our evaluation proved to be more challenging than those used in [42]. In fact, the graphs used in [42] showed a structure that could easily be exploited by our approach. On the most challenging of the four test sets, our algorithm still outperforms state-space exploration by almost an order of magnitude.

To the best of our knowledge, this experimental evaluation is the first to evaluate the timestamped token approach described in [35, 36]. Results confirm that for graphs with relatively few tokens, the method is competitive with the state-space exploration approach, although, on these graphs, its running time is between 40 and 60 times higher than our approach.

Despite the confirmed efficiency of our approach on challenging MRSDF graphs, the scalability of our approach (and other exact approaches) is limited by the size of the single-rate equivalent. As such, the incremental approach that we introduced in Section 6.3 offers a trade-off between complexity of the analysis on the one hand, and accuracy of the estimated throughput on the other hand.

The sets of graphs on which we evaluated the relation between the accuracy of estimated throughput and the size of the graph that was analysed, show that the error of an initial approximation may be extreme. Increasing the size of the graph by unfolding those actors that are critical gives a steep drop in the estimated throughput. The experiments indicate that the relation between the decrease in approximation error, as the size of the analysed graph increases, is exponential.

A disadvantage of (the current implementation of) our incremental approach to throughput analysis is its high running time. We believe that this may be improved in several ways. First, the approach demands that maximum cycle ratio is computed in a dynamic fashion: a change in the graph should lead to an update in the parametric longest paths tree maintained by the algorithm, rather than recomputing it from scratch. A second improvement in the approach is to combine it with an exact analysis of cycles. Cycle analysis is relatively cheap (see Section 6.2.2), and gives an upper bound on the graph's throughput. This upper bound may be tighter than the bound obtained from the graph's optimistic approximation, and thus gives a faster convergence of the throughput bounds.

8 Conclusions and future work

Abstract – *This chapter gives an overview of the research results presented in this thesis, and gives recommendations for future work.*

This thesis provides a mathematical characterisation of SDF graphs, in the form of linear, periodically shift-varying, discrete event systems, formulated in max-plus algebra. With this representation, the thesis introduces several transformations, which are finally combined to the problem of computing a graph's throughput. This chapter provides an overview of the research results presented in the thesis, and gives recommendations for future work.

8.1 Summary and conclusions

SDF was introduced about 30 years ago, as a programming paradigm for the design and implementation of stream-processing systems. Its main purpose was to aid in the design of DSP applications for concurrent implementation on parallel hardware, by making concurrency explicit. Classically, analysis of the *temporal dynamics* of SDF graphs involves the transformation of the graph into its single-rate equivalent, which is an HSDF graph that shows identical behaviour. The size of this graph depends on the ratios between the production and consumption rates associated with channels in the graph, and is exponential in the size of the SDF graph. This affects the scalability of classical methods, and has caused a separation in literature, creating two main schools of thought.

The first of these two schools is primarily concerned with *exact* analysis, computing the same results as the classical method, but avoiding its computational burden by using clever heuristics. The main analysis method that is applied in this line of research is state-space exploration, which is used both for analysing the throughput of a graph, and for the optimisation of buffer capacities under a throughput constraint. In the influential earlier study of [42], which experimentally evaluated the classical approach and state-space exploration, on the task of throughput analysis, the latter was shown to outperform the former. The perspective taken in this school of thought is an *operational* one: SDF graphs are treated as formalisms defined by a set of rules, which specify the behaviour of the graph in terms of valid transitions between its states. In this view, the state of a graph consists of the distribution of tokens over its channels, and the active firings of its actors.

The focus of the second school is on the development of computationally efficient methods, to analyse, and optimise for, performance characteristics of SDF graphs. Efficiency comes at the expense of a loss in accuracy: analysis results are *conservative* with respect to the performance of the application modelled by the graph, which means that its actual performance can be guaranteed to be no worse than the analysis indicates. Conservatively approximating methods have found many applications in the computation of throughput, the optimisation of buffer capacities, and the analysis of data parallelism. However, any (conservative) approximation is only useful if its *error* can be assessed. Existing work does not provide any such assessment. Consequently, the extent to which systems designed using these conservative methods are over-dimensioned can not be quantified.

The two schools described above form two extremes in the trade-off between the computational complexity and accuracy of the analysis. However, this trade-off can not be chosen freely: if the computation of an exact result is computationally too demanding, then the only alternative is a potentially inaccurate conservative approximation. This is due to the fact that the two schools of thought operate on different and incompatible levels of mathematical abstraction: methods of the first school can not improve the approximation computed by methods of the second. This misalignment of abstractions is caused by the difference in perspective, taken in the different approaches. The conservative approximations of the second school are founded on *abstractions* of the temporal dynamics of an SDF graph, but are not formulated *in terms of these* temporal dynamics: methods either assume *periodic arrival times*, of tokens on channels, or *periodic schedules*, of firings of actors. For the "exact" school, the apparent success of the developed state-space exploration approach has distracted research from further developing the classical view, despite the fact that the classical view has its roots in the related and well-matured field of Petri nets.

This thesis therefore revisits the more classical viewpoint, laying a mathematical basis that allows for the unification of the two schools of thought. Our perspective, which we describe in detail in Chapter 3, regards SDF graphs as *linear systems*. That is, a graph is a mathematical structure that transforms an input sequence, of timestamps, into an output sequence of timestamps. The primary difference with existing systems that are used to describe, for example, mechanical or electric systems, is that SDF graphs are described in *max-plus algebra*, in which the operators max and + respectively correspond to conventional addition and multiplication. In this system-theoretic view, a consistent SDF graph is regarded as a *periodically shift-varying linear system*. We give two dual perspectives on the temporal dynamics of an SDF graph: in the *actor firing* perspective, we capture the interdependencies between the firing times of actors, and the *token transfer* perspective describes the times at which tokens are transferred over channels. These two perspectives correspond to the notions of token arrival curves and periodic schedules used in literature.

The system-theoretic perspective that we introduce is fundamental in the sense that it allows for the unification of the two existing perspectives, by providing a solid level of abstraction from which both exact and approximate analyses may be derived. As such, *graph transformations* form a key result built upon this foundation. They form the body of this thesis, and are described in chapters 4, 5 and 6.

Chapter 4 presents several *exact* transformations, in the sense that graphs are transformed into graphs that provably have the same temporal dynamics. These transformations allow CSDF actors to be *unfolded* into actors that represent a specific sampling of the firings of the original actor. For example, an actor may be unfolded into two actors, which represent the odd and even firings of that actor. In particular, such an unfolding transformation allows CSDF graphs to be transformed in *multi-rate equivalents*: MRSDF graphs that have the same temporal dynamics. Furthermore, in line with the classical view, we show how CSDF graphs may be transformed into their single-rate equivalent.

The presented transformations highlight a difference between our perspective and the classical view. Classically, the single-rate equivalent of an SDF graph is constructed by ensuring that the distribution of data (in the form of tokens), in the former and latter, is identical. The de facto transformation achieves this by leaving the *amount of data* processed by the graph invariant by the transformation. In our view, the transformation *duplicates* the data: actors in the single-rate equivalent process the same stream of data as their corresponding actors in the original graph. In this view, the construction of a single-rate equivalent can be understood as an appropriate unfolding of actors, followed by a pruning of non-binding channels that impose a non-binding constraint on the graph's admissible schedules, and a scaling of production and consumption rates.

Chapter 5 introduces four more transformations. While the transformations presented in Chapter 4 are *exact*, in the sense that they result in *temporally equivalent* graphs, the transformations put forward in Chapter 5 are *approximate*: they result in graphs, of which the performance characteristics either give a lower, or an upper bound on the performance characteristics of the original graph. Using the notion of *temporal abstraction*, we prove the validity of these transformations. Furthermore, we demonstrate and prove that any schedule computed for the pessimistic approximation may be mapped to a valid schedule for the original graph, and any schedule for the original graph may be mapped to a valid schedule for the optimistic approximation.

A further aspect that we treat in Chapter 5 is the difference between the two perspectives taken in existing approaches. The quality of conservative approximations, derived from the token-transfer and actor-firing perspective, may differ. We have illustrated how, when applied to CSDF graphs, neither of the two perspectives is better in the sense that it provides a tighter estimate on a graph's performance. More importantly, we have proven that, for MRSDF graphs, the estimations derived from the actor-firing perspective are never worse than those derived from the token-transfer perspective.

Chapter 6 is dedicated to the problem of computing the throughput of an SDF graph. Classically, the throughput of an SDF graph is computed from the maximum cycle ratio of its single-rate equivalent. The single-rate equivalents that we analyse are smaller than those constructed by the de facto transformation used in the classical approach: our transformation creates a *simple* graph, with significantly *fewer* channels than the *multi-graphs* constructed by the classical transformation. The chapter shows, and thoroughly proves, how the single-rate equivalent has a regular structure, which may be exploited. In fact, in case the single-rate equivalent is not strongly connected, its analysis may be restricted to a single strongly connected component, which we refer to as a *feeding component*.

Since the size of a graph's single-rate equivalent is exponential in the size of the original graph, the computation of a graph's throughput through its single-rate equivalent does not scale. In Chapter 6 we therefore introduce a *second* approach to throughput analysis, which combines the transformations of Chapter 4, and the approximations of Chapter 5, into an *incremental* approach. This approach starts with an initial throughput estimation, computed from the graph's pessimistic single-rate approximation, and an initial identification of a potentially critical cycle. We show how the accuracy of the estimation of the throughput of a graph depends on the structure of its single-rate equivalent, and how the accuracy may be improved by changing this structure implicitly, through the *vectorisation* of actors. Our incremental approach is the first demonstration of a method that combines exact and approximate analyses, and gives a practical application of the theory presented in chapters 3, 4 and 5. Moreover, it serves as a justification of our system-theoretic perspective, and proves that it gives a sound level of abstraction that unifies the viewpoints of the two existing schools of thought.

Chapter 7 contains an extensive experimental study of the performance of our exact throughput analysis method. It furthermore studies the validity and effectiveness of our incremental approach, to compute increasingly tighter bounds on a graph's throughput, and an application of the graph transformations and approximations to the problem of computing sufficient buffer sizes under a throughput constraint. The evaluation of our exact throughput analysis method is partially a repetition of a previous study, which compared the classical approach to throughput analysis with a method based on state-space exploration. We have used the same benchmark sets of graphs used in the previous study, and included the same method of state-space exploration in our evaluation. Furthermore, we have included a more recent approach, which is based on a max-plus characterisation of the temporal dynamics between the individual tokens in an SDF graph. To the best of our knowledge, this is the first experimental evaluation of this method on the task of throughput analysis.

The results of our experimental evaluation, of the three methods on the benchmark sets used in the previous study, lead to a conclusion that is the *opposite* of the conclusion drawn in the earlier study: while, in the earlier study, the classical method was significantly outperformed by the state-space exploration method, our study shows that the computation of a graph's throughput, by analysing its single-rate equivalent, outperforms state-space exploration. In fact, the running time of our method was often several orders of magnitude lower than the time required by state-space exploration.

Results obtained for the incremental approaches confirm the validity of the approach, on both the throughput analysis and buffer capacity optimisation task: as the graph is partially transformed into a larger graphs, with approximations identifying which of the actors to unfold, the estimated throughput or buffer capacities become more accurate. Especially on SDF graphs that model real applications, from the digital signal processing and multimedia domain, the incremental analysis shows its effectiveness. An observed disadvantage of the incremental analysis is its running time, which is partially due to the fact that the analysis is applied to evolving graphs. For throughput analysis, this means that the relatively naive but simple approach of analysing the full single-rate equivalent is faster than the smarter incremental approach. Several of the problems attributing to the high running time may be overcome by a smarter choice of algorithms, and we believe that there are many opportunities for future research to improve upon the incremental approach, maturing it into a state-of-the-art method that increases the scalability, and consequently the applicability, of SDF graph analysis methods.

8.2 Contributions

This thesis addresses the shortcomings of current approaches to the analysis of SDF graphs. The research results presented in this thesis contribute to addressing the main research question:

> *How can we combine exact and approximate analysis of synchronous dataflow graphs into an approach that offers a trade-off between accuracy and complexity?*

We have approached this question by taking a system-theoretic perspective, in which we have given a mathematical characterisation of the temporal dynamics of an SDF graph, and demonstrated how both exact and approximate analyses can be derived. In particular, the incremental approach to throughput analysis, which we presented in Chapter 6, serves as a justification of our system-theoretic perspective, and proves that it gives a sound basis for approximate and exact analyses. This forms the primary contribution of this thesis.

This first and foremost theoretical contribution is the product of several other contributions of this thesis, both practical and theoretical, listed below:

» The *actor-firing* and *token-transfer* perspective, introduced in Chapter 3, give an explicit formal characterisation of the two predominant existing approaches: state-of-the-art approximate methods either assume periodic arrival times of tokens on channels, or periodic firing times of actors, without assessing the quality of the estimations that are derived from them. We contribute to the field by demonstrating how approximations derived from these two viewpoints relate to each other in terms of accuracy: we show that for CSDF graphs, neither of the two perspectives is consistently better than the other, but for MRSDF graphs, the actor-firing perspective is to be preferred. This means that the existing approaches based on the analysis of *token arrival times* is, in the best case, as accurate than approaches based on the periodic firing of actors.

» Existing approaches to the approximation of performance characteristics of SDF graph lack a quantitative assessment of its accuracy. This due to their focus on computing a *conservative* approximation. The four single-rate approximations that we introduce in Chapter 5 provide a conservative (*pessimistic*), as well as an *optimistic* approximation. Tight bounds on a graph's throughput may be computed from these approximations, and the difference between these bounds provides an upper bound on the error made by the conservative approximation.

» The concept of the *multi-rate* equivalent of CSDF graph is a novel one: currently, the only transformation from CSDF into ("lumped") MRSDF that is currently known is temporally conservative, and may introduce deadlock. We prove that our transformation gives a graph that is temporally equivalent to the original graph, and show how schedules for the CSDF graph map to the multi-rate equivalent, and vice versa. This contributes to the field by generalising the scope of existing analysis techniques for MRSDF graphs to CSDF graphs.

» An important property of the transformations presented in Chapter 4 is that they can deal with SDF graphs in which the rates and initial tokens of channels are *parameterised*, whereas existing transformations can not. This contributes to the field by putting the current conviction, that the structure of a graph's single-rate equivalent depends on the number of initial tokens placed on its channels, into a new perspective.

» The construction of the single-rate equivalent of an SDF graph allows approximate analyses to be applied within the same mathematical abstraction. Currently, approximate methods typically operate by formulating the analysis as a *linear program*, which is subsequently solved by a state-of-the art solver. Representing the approximated temporal dynamics of an SDF graph as an HSDF graph maintains the structure of the analysis or optimisation problem, which aids in the interpretation of the approximation, in terms of the properties of the original graph.

» Our perspective on SDF graphs allows the study of CSDF graphs where actors with varying execution times are free to start several consecutive executions in parallel. Currently, this scheduling freedom is either disallowed, which means that CSDF does not, taxonomically speaking, generalise MRSDF, or it is allowed, in which case functional determinacy is no longer guaranteed. We show how firings with differing execution times may be executed in parallel, while maintaining functional determinacy.

» The incremental approach to throughput analysis, which combines the transformations of Chapter 4 and Chapter 5, bridges the gap between current approximate and exact approaches. We describe how the error of an approximation depends on the structure of the graph's single-rate equivalent, and how, by *vectorisation* of actors, the throughput of an MRSDF cycle may be analysed without unfolding it into its single-rate equivalent. We further show how, for MRSDF graphs, an initial rough estimate may be improved upon, by unfolding critical actors in the graph. This contributes to the field by giving a solid foundation, based upon which the two currently separated lines of research may be unified.

» Our experimental evaluation of *throughput analysis* confirms the validity of methods based on the analysis of the single-rate equivalent. The results of this study, as presented in Chapter 7, clearly reveal the efficiency of our method, compared to the state-of-the-art. This contributes to the field by improving the scalability of throughput analysis.

» We have implemented all SDF graph transformations described in chapters 4, 5 and 6, in a Python library. The library is available at `http://caes.ewi.utwente.nl/sdf`. The throughput analysis algorithms and benchmark sets used in the evaluation are available from the same link.

8.3 Recommendations for future work

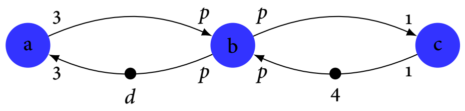
The theoretical foundation put forward in this thesis provides a basis for future research directions. We believe that the most promising line of research is opened up by our novel incremental analysis techniques, to both throughput analysis and the optimisation of buffer capacities. Although the experimental evaluation confirmed the validity of these incremental techniques, the high running times leave many opportunities for improvements. Future work may investigate different combinations of exact analyses, of the single-rate equivalent of an MRSDF subgraph, on the one hand, and the heuristic identification of critical cycles and associated throughput, provided by the single-rate approximations, on the other hand. We foresee several improvements over the ideas presented in this thesis, such that the scalability of SDF graph analysis is increased, and therewith its applicability.

The mathematical characterisation that we have introduced in this thesis, and for which we have developed approximation and transformation methods, is not limited to CSDF graphs. Several models that extend SDF may be described using the same approach. A model that is close to CSDF, and for which the theory set forth in this thesis may be applied without too much effort, is the *phased computation* *graph* [96]. Phased computation graphs extend CSDF graphs with an initial set of phases, which an actor executes prior to entering its cyclically varying behaviour, and allows a *threshold* to be associated with each phase. The threshold gives the number of tokens that must be present on the particular incoming channel, before tokens may be consumed from it, similar to the *computation graphs* introduced by Karp and Miller [55]. By generalising the theory of this thesis to phased computation graphs, they may be transformed into multi-rate equivalents, and consequently be analysed using available methods.

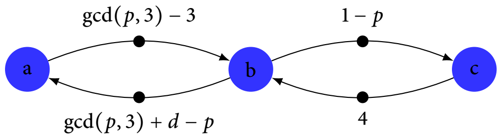
Another example of a model that may be attractive to model in a system-theoretic way is multi-dimensional synchronous dataflow (MDDF) [59, 70], which forms an extension of MRSDF graphs, by generalising scalar production and consumption rates to tuples that specify multidimensional index spaces. Multi-dimensional dataflow graphs find their application in the exploitation of data parallelism in image processing. In order to apply the approximating and unfolding transformations of Chapter 5 and Chapter 4 to these graphs, one must specify a predecessor function for channels in an MDDF graph, in the same spirit as we did, in Chapter 3, for CSDF graph. Although finding a suitable mathematical characterisation of an MDDF channel may be far from trivial, it may enable a transformation from MDDF graphs into multi-rate equivalents, and therewith bring analysis techniques to MDDF.

Extending the theory of this thesis to other models can be regarded as a *widening* of its scope. We believe that there is also much to gain in a *deepening* of the scope: extending the range of problems that we may approach using the theory. The incremental nature of the approach we presented for throughput analysis (Chapter 6), and buffer capacity optimisation (Chapter 7), may be applied to several other (optimisation) problems. For example, a natural generalisation of the buffer optimisation problem is the problem of optimising the (production and consumption) rates associated with an actor: the transformations presented in chapters 4 and 5 may safely be applied in case these rates are parameterised. Rate parameters will appear in the form of tokens in the graphs produced by the transformations. An example of this idea is shown in Figure 8.1: Figure 8.1(b) depicts the pessimistic single-rate approximation of the parameterised MRSDF graph of Figure 8.1(a). Note that the problem of optimising these rates, for example, under a throughput constraint, is more involved than the buffer capacity optimisation problem. This is due to the fact that increasing the rate, p, associated with actor b, has a *non-linear* effect on the throughput of cycle aba, as the term $\gcd(p, 3)$ is equal to 3 in case p is a multiple of three, and 1 otherwise.

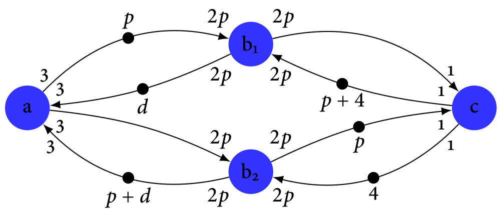
Another interesting problem for which applications of the theory of this thesis may be worthwhile to explore, is the computation of static order schedules, given that the number of processors is limited. This can be approached by adding cycles made up of HSDF channels to the graph, the role of which is to enforce a strictly sequential execution of consecutive firings of actors, essentially partitioning the graphs into subgraphs that are each associated with a particular processor. Again, this



(a) Parameterised MRSDF graph.



(b) Pessimistic single-rate approximation.



(c) Unfolded b into its odd and even firings.

Figure 8.1 – Both a single-rate approximation, and the transformation of a graph by unfolding an actor, allow one to transform SDF graphs where rates and actors are parameterised. This allows one to explore the design space spanned by the parameters, in an iterative fashion.

problem can be approached in an incremental fashion: in the coarsest approximation, the obtained static order schedule schedules each of an actor's firing on the *same* processing element. In an unfolded graph, for example, odd and even firings could be scheduled onto different processors, in an alternating fashion. Costs (i.e., time) of inter processor communication may be modelled by adding actors with an appropriate execution time.

In conclusion, we believe that the work presented in this thesis, in particular the system-theoretic characterisation of SDF graphs and the transformations and approximation built upon it, serves as a fertile basis for future research.

A Integer Arithmetic

The *predecessor functions* defined in Chapter 3 capture the dependency between the number of producing and consuming firings, associated with a channel in an SDF graph. When manipulating these functions, integers play an important role. This chapter gives several relations and properties that hold for integer functions, which find their application throughout the thesis. Most of the relations are explained in [44]. For those that are not, we give a short proof or a brief explanation.

The *floor* and *ceiling* functions map the reals onto the integers. The floor of x, denoted $\lfloor x \rfloor$, and the ceiling of x, denoted $\lceil x \rceil$, are defined as:

$$\lfloor x \rfloor = \max \{ n \in \mathbb{Z} | n \leq x \}, \tag{A.1a}$$

$$\lceil x \rceil = \min \{ n \in \mathbb{Z} | n \geq x \}. \tag{A.1b}$$

They may be interchanged using ($n, m \in \mathbb{Z}$, n positive):

$$\left\lfloor \frac{m}{n} \right\rfloor = \left\lceil \frac{m-n+1}{n} \right\rceil \qquad \left\lceil \frac{m}{n} \right\rceil = \left\lfloor \frac{m+n-1}{n} \right\rfloor. \tag{A.2}$$

The *modulo* function gives the remainder after division of an integer by another integer. It is defined in terms of the floor function as:

$$a \bmod b = a - b \left\lfloor \frac{a}{b} \right\rfloor. \tag{A.3}$$

Note that the modulo function always yields a non-negative integer.

The floor and ceiling function may be used to derive an equivalent inequality between integers, from an inequality where one of the involved terms is non-integer:

$$\lfloor x \rfloor < n \Leftrightarrow x < n, \tag{A.4a}$$

$$\lceil x \rceil \leq n \Leftrightarrow x \leq n, \tag{A.4b}$$

$$\lceil x \rceil > n \Leftrightarrow x > n, \tag{A.4c}$$

$$\lfloor x \rfloor \geq n \Leftrightarrow x \geq n. \tag{A.4d}$$

The negation of the floor function equals the ceiling of the negated argument:

$$-\lfloor x \rfloor = \lceil -x \rceil. \tag{A.5}$$

From (A.1) and (A.4), the following equalities may be derived, where n is a positive integer, $m \in \mathbb{Z}$, and $x \in \mathbb{R}$:

$$\left\lfloor \frac{m+x}{n} \right\rfloor = \left\lfloor \frac{m+\lfloor x \rfloor}{n} \right\rfloor \qquad \left\lceil \frac{m+x}{n} \right\rceil = \left\lceil \frac{m+\lceil x \rceil}{n} \right\rceil, \tag{A.6}$$

Of particular interest is the composition of the floor and ceiling functions with *affine functions*. An affine function f is a function that can be written as:

$$f(x) = mx + b,$$

where m and b are constants. If we compose the floor (or ceiling) functions with an affine function $f : \mathbb{Q} \to \mathbb{Q}$, we obtain functions of the following form:

$$f(k) = \left\lfloor \frac{ak+b}{m} \right\rfloor, \tag{A.7}$$

When composed in this way, the resulting function is no longer (guaranteed to be) affine. However, if we substitute $k'n + r$ for k, where n and r are integers, chosen in such a way that m divides the product of a and n, we obtain the function ($k' \in \mathbb{Z}$):

$$g_r(k') \equiv f(k'n+r) = \left\lfloor \frac{ak'n + ar + b}{m} \right\rfloor = \frac{ak'}{m} + \left\lfloor \frac{ar+b}{m} \right\rfloor, \tag{A.8}$$

which is again affine. In fact, function $g_r(k')$ as defined by (A.8) is affine for every $r \in \{0, \ldots, n-1\}$. In other words, a function of the form (A.7) is affine when restricted to the so-called *residue classes* of the integers modulo n, where the residue class m is the set $\{m + kn | k \in \mathbb{Z}\}$. This gives rise to the following definition, which is taken from [56].

Definition A.1 (Residue-class-wise-affine mapping). *A mapping $f : \mathbb{Z} \to \mathbb{Z}$ is* residue-class-wise-affine *if there is a positive integer m such that all m restrictions of f to the residue classes of $\mathbb{Z}/m\mathbb{Z}$ are affine. That is, f is residue-class-wise-affine modulo m if:*

$$\exists b, n \in \mathbb{Z}. \forall a \in \{0, \ldots, m-1\} : f(a+km) = b + jn.$$

The graph of a residue-class-wise-affine mapping follows a staircase pattern. A residue-class-wise-affine mapping $f : \mathbb{Z}/m\mathbb{Z} \to \mathbb{Z}/n\mathbb{Z}$ has a slope of $\frac{n}{m}$.

The graphs of two affine functions $f(x)$ and $g(x)$ that have the same slope are parallel; the difference $f(x) - g(x)$ is constant for all x in the domain of f. If f and g are residue-class-wise affine mappings, then their difference, $f(k) - g(k)$, may not be constant for all $k \in \mathbb{Z}$, even if they have equal slopes. The maximum and minimum difference between two such mappings depends on the greatest common divisor, of the numerator and denominator, of their slopes. The following identities give the difference between two equally-sloped residue-class-wise affine mappings depends:

$$\max_{k\in\mathbb{Z}}\left\{\left\lceil\frac{km-c_1}{n}\right\rceil-\left\lceil\frac{km-c_2}{n}\right\rceil\right\}=\left\lceil\frac{g_{m,n}\left(\left\lfloor\frac{c_2}{g_{m,n}}\right\rfloor-\left\lfloor\frac{c_1}{g_{m,n}}\right\rfloor\right)}{n}\right\rceil, \tag{A.9}$$

where $g_{m,n} = \gcd(m, n)$.

Proof. Let $m' = \frac{m}{g_{m,n}}$ and $n' = \frac{n}{g_{m,n}}$. We rewrite the left-hand side in the above identity into the following:

$$\max_{k\in\mathbb{Z}}\left\{\left\lceil\frac{k'-g_{m,n}\left\lfloor\frac{c_1}{g_{m,n}}\right\rfloor}{n'}\right\rceil-\left\lceil\frac{k'-\left\lfloor\frac{c_2}{g_{m,n}}\right\rfloor}{n'}\right\rceil\right\}$$

Now, choose functions $q, r : \mathbb{Z} \to \mathbb{Z}$, with $0 \leq r(k) < n'$ for all k, such that $k' - g_{m,n}\left\lfloor\frac{c_2}{g_{m,n}}\right\rfloor = q(k')n' + r(k')$. Then $k' - g_{m,n}\left\lfloor\frac{c_1}{g_{m,n}}\right\rfloor = q(k)n' + r(k') - g_{m,n}\left\lfloor\frac{c_2}{g_{m,n}}\right\rfloor + g_{m,n}\left\lfloor\frac{c_1}{g_{m,n}}\right\rfloor$. Substituting this into the above gives:

$$\max_{k\in\mathbb{Z}}\left\{\left\lceil\frac{r(k')-g_{m,n}\left\lfloor\frac{c_1}{g_{m,n}}\right\rfloor-g_{m,n}\left\lfloor\frac{c_2}{g_{m,n}}\right\rfloor}{n'}\right\rceil-\left\lceil\frac{r(k')}{n'}\right\rceil\right\}.$$

Since k' and n' are relatively prime, $r(k')$ can attain the values $0, \ldots, n'-1$. The two possible values for $r(k')$ at which the above maximum is attained, are $r(k') = 0$ and $r(k') = n' - 1$. The former of these two is easily verified to give the desired maximum. □

In a similar fashion, the following identities may be derived:

$$\max_{k\in\mathbb{Z}}\left\{\left\lceil\frac{km-c_1}{n}\right\rceil-\left\lceil\frac{km-c_2}{n}\right\rceil\right\}=\left\lceil\frac{g_{m,n}\left(\left\lfloor\frac{c_2}{g_{m,n}}\right\rfloor-\left\lfloor\frac{c_1}{g_{m,n}}\right\rfloor\right)}{n}\right\rceil \tag{A.10}$$

$$\min_{k\in\mathbb{Z}}\left\{\left\lfloor\frac{km-c_1}{n}\right\rfloor-\left\lfloor\frac{km-c_2}{n}\right\rfloor\right\}=\left\lfloor\frac{g_{m,n}\left(\left\lceil\frac{c_2}{g_{m,n}}\right\rceil-\left\lceil\frac{c_1}{g_{m,n}}\right\rceil\right)}{n}\right\rfloor \tag{A.11}$$

$$\max_{k\in\mathbb{Z}}\left\{\left\lfloor\frac{km-c_1}{n}\right\rfloor-\left\lfloor\frac{km-c_2}{n}\right\rfloor\right\}=\left\lceil\frac{g_{m,n}\left(\left\lceil\frac{c_2}{g_{m,n}}\right\rceil-\left\lceil\frac{c_1}{g_{m,n}}\right\rceil\right)}{n}\right\rceil, \tag{A.12}$$

with again $g_{m,n} = \mathrm{gcd}(m,n)$.

Two identities that are related to those above, involve the minimum and maximum of the difference between an affine function, and its composition with the ceiling (or floor) function. These are:

$$\min_{k\in\mathbb{Z}}\left\{\frac{km}{n} - \left\lceil\frac{km-d}{n}\right\rceil \middle| k\in\mathbb{Z}\right\} = \frac{\mathrm{gcd}(m,n)\left\lceil\frac{d+1}{\mathrm{gcd}(m,n)}\right\rceil}{n} - 1, \tag{A.13a}$$

$$\max_{k\in\mathbb{Z}}\left\{\frac{km}{n} - \left\lceil\frac{km-d}{n}\right\rceil \middle| k\in\mathbb{Z}\right\} = \frac{\mathrm{gcd}(m,n)\left\lfloor\frac{d}{\mathrm{gcd}(m,n)}\right\rfloor}{n}, \tag{A.13b}$$

with $m, n, d \in \mathbb{Z}$ and n positive.

Proof. Using the definition of the modulo operator (A.3), in terms of the floor function, and the negation of the ceiling function (A.5), we may reformulate the set that appears on the left-hand sides of the two identities as:

$$\left\{\frac{d-(d-km) \bmod n}{n} \middle| k\in\mathbb{Z}\right\}. \tag{A.14}$$

Minimum and maximum values of this set are attained at the respective maximum and minimum possible value of $(d-km) \bmod n$. We have [44]:

$$\max_{k\in\mathbb{Z}}\{(d-km) \bmod n\} = n - \mathrm{gcd}(m,n) + d \bmod \mathrm{gcd}(m,n),$$
$$\min_{k\in\mathbb{Z}}\{(d-km) \bmod n\} = d \bmod \mathrm{gcd}(m,n).$$

By writing the mod operator in terms of the floor or ceiling operator, the two stated identities now readily follow. This completes the proof. □

Bibliography

[1] SDF3. `http://www.es.ele.tue.nl/sdf3/`. Accessed: 2015-05-30. (Cited on page 175).

[2] P. Aubry, M. Benazouz, and R. Sirdey. An Approximate Method for Throughput Evaluation of Cyclo-static Dataflow Programs. In *Proceedings of the eighth International Conference on Complex, Intelligent and Software Intensive Systems*, pages 433–438. IEEE, July 2014. ISBN 978-1-4799-4325-8. doi: 10.1109/CISIS.2014.61. (Cited on pages 46 and 47).

[3] N. Bambha, V. Kianzad, M. Khandelia, and S. S. Bhattacharyya. Intermediate Representations for Design Automation of Multiprocessor DSP Systems. *Design Automation for Embedded Systems*, 7(4):307–323, 2002. ISSN 1572-8080. doi: 10.1023/A:1020307222052. (Cited on page 38).

[4] V. Batagelj and U. Brandes. Efficient generation of large random networks. *Physical Review E*, 71(3):036113, mar 2005. ISSN 1539-3755. doi: 10.1103/PhysRevE.71.036113. (Cited on page 176).

[5] A. Benabid-Najjar, C. Hanen, O. Marchetti, and A. Munier-Kordon. Periodic Schedules for Bounded Timed Weighted Event Graphs. *IEEE Transactions on Automatic Control*, 57(5):1222–1232, May 2012. ISSN 0018-9286. doi: 10.1109/TAC.2012.2191871. (Cited on pages 21 and 138).

[6] M. Benazouz and A. Munier-Kordon. Cyclo-static DataFlow phases scheduling optimization for buffer sizes minimization. In *Proceedings of the 16th International Workshop on Software and Compilers for Embedded Systems (M-SCOPES)*, pages 3–12, New York, NY, USA, June 2013. ACM Press. ISBN 9781450321426. doi: 10.1145/2463596.2463602. (Cited on page 71).

[7] M. Benazouz, O. Marchetti, A. Munier-Kordon, and T. Michel. A new method for minimizing buffer sizes for Cyclo-Static Dataflow graphs. In *8th IEEE Workshop on Embedded Systems for Real-Time Multimedia*, pages 11–20. IEEE, oct 2010. ISBN 978-1-4244-9084-4. doi: 10.1109/ESTMED.2010.5666980. (Cited on page 189).

[8] M. Benazouz, A. Munier-Kordon, T. Hujsa, and B. Bodin. Liveness evaluation of a cyclo-static DataFlow graph. In *Proceedings of the 50th Annual Design Automation Conference on - DAC '13*, page 1, New York, New York, USA, May 2013. ACM Press. ISBN 9781450320719. doi: 10.1145/2463209.2488736. (Cited on pages 41, 42, and 99).

[9] E. Best and R. Devillers. State space axioms for T-systems. *Acta Informatica*, Feb. 2015. ISSN 0001-5903. doi: 10.1007/s00236-015-0219-0. (Cited on page 21).

[10] S. S. Bhattacharyya, P. K. Murthy, and E. A. Lee. Synthesis of Embedded Software from Synchronous Dataflow Specifications. *Journal of VLSI signal processing systems for signal, image and video technology*, 21(2):151–166, 1999. ISSN 1573-109X. doi: 10.1023/A:1008052406396. (Cited on page 175).

[11] G. Bilsen, M. Engels, R. Lauwereins, and J. Peperstraete. Cyclo-Static dataflow. *IEEE Transactions on Signal Processing*, 44(2):397–408, 1996. ISSN 1053587X. doi: 10.1109/78.485935. (Cited on pages 4, 12, 13, 14, 15, 17, 18, 39, 72, 75, 76, 77, 94, 101, and 104).

[12] B. Bodin. *Analyse d'Applications Flot de Données pour la Compilation Multiprocesseur*. PhD thesis, Université Pierre et Marie Curie - Paris VI, Dec. 2013. (Cited on pages 19 and 20).

[13] B. Bodin, A. Munier-Kordon, and B. de Dinechin. K-periodic schedules for evaluating the maximum throughput of a synchronous dataflow graph. In *Proceedings of the International Conference on Embedded Computer Systems (SAMOS)*, pages 152–159, July 2012. doi: 10.1109/SAMOS.2012.6404169. (Cited on pages 139 and 170).

[14] B. Bodin, A. Munier-Kordon, and B. D. de Dinechin. Periodic schedules for Cyclo-Static Dataflow. In *Proceedings of the 11th IEEE Symposium on Embedded Systems for Real-time Multimedia*, pages 105–114. IEEE, Oct. 2013. ISBN 978-1-4799-1284-1. doi: 10.1109/ESTIMedia.2013.6704509. (Cited on pages 45, 46, and 138).

[15] B. Bodin, Y. Lesparre, J.-M. Delosme, and A. Munier-Kordon. Fast and efficient dataflow graph generation. In *Proceedings of the 17th International Workshop on Software and Compilers for Embedded Systems - SCOPES '14*, pages 40–49, New York, New York, USA, June 2014. ACM Press. ISBN 9781450329415. doi: 10.1145/2609248.2609258. (Cited on page 42).

[16] A. Bouakaz, J.-P. Talpin, and J. Vitek. Affine Data-Flow Graphs for the Synthesis of Hard Real-Time Applications. In *Proceedings of the 12th International Conference on Application of Concurrency to System Design*, pages 183–192. IEEE, June 2012. ISBN 978-0-7695-4709-1. doi: 10.1109/ACSD.2012.16. (Cited on page 22).

[17] C. Cassandras. *Introduction to discrete event systems*. Springer Science+Business Media, New York, N.Y, 2008. ISBN 0387333320. (Cited on pages 12 and 22).

[18] D. Chao, M. Zhou, and D. Wang. Multiple weighted marked graphs. In *Proceedings of the 12th IFAC World Congress, Sydney, Australia*, volume 4, pages 259–263, 1993. (Cited on page 21).

[19] J. Cochet-terrasson, G. Cohen, S. Gaubert, M. M. Gettrick, and J.-P. Quadrat. Numerical Computation of Spectral Elements in Max-Plus Algebra, 1998. (Cited on pages 30 and 174).

[20] G. Cohen and P. Moller. Algebraic Tools for the Performance Evaluation of Discrete Event Systems, 1989. (Cited on page 12).

[21] G. Cohen, D. Dubois, J.-P. Quadrat, and M. Viot. A linear-system-theoretic view of discrete-event processes and its use for performance evaluation in manufacturing. *IEEE Transactions on Automatic Control*, 30(3):210–220, Mar. 1985. ISSN 0018-9286. doi: 10.1109/TAC.1985.1103925. (Cited on page 19).

[22] G. Cohen, G. J. Olsder, and J.-P. Quadrat. *Synchronization and linearity: an algebra for discrete event systems*. Wiley New York, 1992. (Cited on pages 12, 13, 22, 23, 25, 27, 28, 30, 31, and 85).

[23] G. Cohen, S. Gaubert, and J.-P. Quadrat. Timed-event graphs with multipliers and homogeneous min-plus systems. *IEEE Transactions on Automatic Control*, 43(9): 1296–1302, 1998. ISSN 00189286. doi: 10.1109/9.718621. (Cited on pages 21, 41, and 113).

[24] G. Cohen, S. Gaubert, and J.-P. Quadrat. Max-plus algebra and system theory: Where we are and where to go now. *Annual Reviews in Control*, 23:207–219, Jan. 1999. ISSN 13675788. doi: 10.1016/S1367-5788(99)90091-3. (Cited on pages 12 and 29).

[25] F. Commoner, A. W. Holt, S. Even, and A. Pnueli. Marked directed graphs. *Journal of Computer and System Sciences*, 5(5):511–523, 1971. ISSN 0022-0000. (Cited on page 29).

[26] B. Cottenceau, L. Hardouin, and J.-L. Boimond. Modeling and Control of Weight-Balanced Timed Event Graphs in Dioids. *IEEE Transactions on Automatic Control*, 59(5):1219–1231, May 2014. ISSN 0018-9286. doi: 10.1109/TAC.2013.2294822. (Cited on page 21).

[27] B. Cottenceau, S. Lahaye, and L. Hardouin. Modeling of Time-Varying (max, +) Systems by Means of Weighted Timed Event Graphs. In *Discrete Event Systems*, volume 12, pages 465–470, May 2014. ISBN 978-3-902823-61-8. (Cited on pages 21 and 31).

[28] M. Damavandpeyma, S. Stuijk, T. Basten, M. Geilen, and H. Corporaal. Modeling static-order schedules in synchronous dataflow graphs. In *Proceedings of the Conference on Design, Automation & Test in Europe (DATE)*, pages 775–780. IEEE, Mar. 2012. ISBN 978-1-4577-2145-8. doi: 10.1109/DATE.2012.6176588. (Cited on page 170).

[29] A. Dasdan. Experimental analysis of the fastest optimum cycle ratio and mean algorithms. *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, 9(4):385–418, 2004. ISSN 1084-4309. (Cited on pages 30, 144, 174, and 178).

[30] A. Dasdan, S. S. Irani, and R. K. Gupta. Efficient algorithms for optimum cycle mean and optimum cost to time ratio problems. *Proceedings of the 36th annual ACM/IEEE Design Automation Conference (DAC'99)*, pages 37–42, 1999. (Cited on page 30).

[31] A. Davis and R. Keller. Data Flow Program Graphs. *Computer*, 15(2):26–41, Feb. 1982. ISSN 0018-9162. doi: 10.1109/MC.1982.1653939. (Cited on page 11).

[32] J. B. Dennis. First version of a data flow procedure language. In B. Robinet, editor, *Programming Symposium*, volume 19 of *Lecture Notes in Computer Science*, pages 362–376. Springer Berlin Heidelberg, 1974. ISBN 978-3-540-06859-4. doi: 10.1007/3-540-06859-7_145. (Cited on page 52).

[33] M. Engels, G. Bilson, R. Lauwereins, and J. Peperstraete. Cycle-static dataflow: model and implementation. In *Proceedings of the 28th Asilomar Conference on Signals, Systems and Computers*, volume 1, pages 503–507. IEEE Comput. Soc. Press, 1994. ISBN 0-8186-6405-3. doi: 10.1109/ACSSC.1994.471504. (Cited on page 17).

[34] Z. Ésik, P. Chrzastowski-Wachtel, and M. Raczunas. Liveness of weighted circuits and the diophantine problem of Frobenius - Fundamentals of Computation Theory - Lecture Notes in Computer Science, 1993. (Cited on page 21).

[35] M. Geilen. Reduction techniques for synchronous dataflow graphs. In *Proceedings of the 46th ACM/IEEE Design Automation Conference (DAC)*, pages 911–916. IEEE, 2009. (Cited on pages 35, 36, 37, 38, 39, 45, 174, 178, 180, and 192).

[36] M. Geilen. Synchronous dataflow scenarios. *ACM Transactions on Embedded Computing Systems*, 10(2):1–31, Dec. 2010. ISSN 15399087. doi: 10.1145/1880050.1880052. (Cited on pages 21, 37, 45, 174, 178, 180, and 192).

[37] M. Geilen and S. Stuijk. Worst-case performance analysis of synchronous dataflow scenarios. In *Proceedings of the eighth IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis - CODES/ISSS '10*, page 125, New York, New York, USA, Oct. 2010. ACM Press. ISBN 9781605589053. doi: 10.1145/1878961.1878985. (Cited on pages 21 and 35).

[38] M. Geilen, S. Tripakis, and M. Wiggers. The earlier the better. In *Proceedings of the 14th international conference on Hybrid systems: computation and control - HSCC '11*, page 23, New York, New York, USA, Apr. 2011. ACM Press. ISBN 9781450306294. doi: 10.1145/1967701.1967707. (Cited on page 138).

[39] S. Gerez, S. Heemstra de Groot, and O. Herrmann. A polynomial time algorithm for the computation of the iteration-period bound in recursive data flow graphs. *IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications*, 39(1): 49–52, 1992. ISSN 10577122. doi: 10.1109/81.109243. (Cited on pages 30 and 174).

[40] A. Ghamarian, M. Geilen, T. Basten, B. Theelen, M. Mousavi, and S. Stuijk. Liveness and boundedness of synchronous data flow graphs. In *Proceedings of the 6th conference on Formal Methods in Computer Aided Design (FMCAD)*, pages 68–75. IEEE, Nov. 2006. (Cited on pages 14, 19, 28, 70, 71, 142, 152, 154, and 155).

[41] A. H. Ghamarian. *Timing Analysis of Synchronous Data Flow Graphs*. PhD thesis, Technische Universiteit Eindhoven, July 2008. (Cited on pages 28, 142, and 174).

[42] A. H. Ghamarian, M. C. W. Geilen, S. Stuijk, T. Basten, B. D. Theelen, M. R. Mousavi, A. J. M. Moonen, and M. J. G. Bekooij. Throughput Analysis of Synchronous Data Flow Graphs. In *Proceedings of the 6th International Conference on Application of Concurrency to System Design (ACSD)*, pages 25–36. IEEE, 2006. (Cited on pages 18, 33, 43, 44, 46, 47, 71, 170, 174, 175, 177, 178, 179, 180, 181, 185, 192, and 195).

[43] S. V. Gheorghita, T. Basten, and H. Corporaal. Scenario Selection and Prediction for DVS-Aware Scheduling of Multimedia Applications. *Journal of Signal Processing Systems*, 50(2):137–161, July 2007. ISSN 1939-8018. doi: 10.1007/s11265-007-0086-1. (Cited on page 22).

[44] R. L. Graham, D. E. Knuth, and O. Patashnik. *Concrete Mathematics: A Foundation for Computer Science*. Addison Wesley, Jan. 1994. ISBN 0201558025. (Cited on pages 205 and 208).

[45] S. Ha and H. Oh. Decidable Dataflow Models for Signal Processing: Synchronous Dataflow and its Extensions. In S. S. Bhattacharyya, E. F. Deprettere, R. Leupers, and J. Takal, editors, *Handbook of Signal Processing Systems*, pages 1083–1109. Springer, 2013. ISBN 978-1-4614-6858-5. doi: 10.1007/978-1-4614-6859-2. (Cited on pages 38, 40, 104, and 138).

[46] S. Hamaci, J.-L. Boimond, and S. Lahaye. Modeling and Control of Discrete Timed Event Graphs with Multipliers using (Min,+) Algebra. In *ICINCO 2004 - International Conference on Informatics in Control, Automation and Robotics*, volume 3, pages 32–37, Setubal, Portugal, Aug. 2004. (Cited on page 21).

[47] M. Hartmann and J. B. Orlin. Finding minimum cost to time ratio cycles with small integral transit times. *Networks*, 23(6):567–574, 1993. ISSN 1097-0037. doi: 10.1002/net.3230230607. (Cited on page 30).

[48] J. P. Hausmans, S. J. Geuns, M. H. Wiggers, and M. J. Bekooij. Compositional temporal analysis model for incremental hard real-time system design. In *Proceedings of the 10th ACM International Conference on Embedded Software (EMSOFT)*, pages 185–194, New York, NY, USA, Oct. 2012. ACM Press. ISBN 9781450314251. doi: 10.1145/2380356.2380390. (Cited on pages 22, 42, 66, 113, 138, and 139).

[49] B. Heidergott, G. J. Olsder, and J. van der Woude. *Max Plus at Work: modeling and analysis of synchronized systems*. Princeton University Press, 2006. ISBN 9780691117638. (Cited on pages 13, 18, 20, 22, 23, 25, 27, 28, 43, 142, and 174).

[50] H. P. Hillion and A. H. Levis. Timed event-graph and performance evaluation of systems. Technical Report LIDS-P; 1639, Laboratory for Information and Decision Systems, Massachusetts Institute of Technology, 1987. (Cited on page 19).

[51] K. Ito and K. Parhi. Determining the iteration bounds of single-rate and multi-rate data-flow graphs. In *Proceedings of the 1994 Asia Pacific Conference on Circuits and Systems*, pages 163–168. IEEE, 1994. ISBN 0-7803-2440-4. doi: 10.1109/APCCAS.1994.514543. (Cited on pages 12, 34, 35, and 174).

[52] K. Ito and K. K. Parhi. Determining the minimum iteration period of an algorithm. *Journal of VLSI Signal Processing*, 11(3):229–244, Dec. 1995. ISSN 0922-5773. doi: 10.1007/BF02107055. (Cited on pages 30, 34, 35, and 158).

[53] G. Kahn. The semantics of a simple language for parallel programming. In J. L. Rosenfeld, editor, *Information Processing*, pages 471–475. North Holland, Amsterdam, Aug 1974. (Cited on pages 3 and 16).

[54] R. Karp. Parametric shortest path algorithms with an application to cyclic staffing. *Discrete Applied Mathematics*, 3(1):37–45, Feb. 1981. ISSN 0166218X. doi: 10.1016/0166-218X(81)90026-3. (Cited on pages 30 and 174).

[55] R. M. Karp and R. E. Miller. Properties of a Model for Parallel Computations: Determinancy, Termination, Queueing. *SIAM Journal on Applied Mathematics*, 14(6), Sept. 1966. (Cited on pages 3, 14, 19, and 202).

[56] S. Kohl. A simple group generated by involutions interchanging residue classes of the integers. *Mathematische Zeitschrift*, 264(4):927–938, Mar. 2009. ISSN 0025-5874. doi: 10.1007/s00209-009-0497-8. (Cited on page 206).

[57] S. Lahaye, J.-L. Boimond, and J.-L. Ferrier. Just-in-time control of time-varying discrete event dynamic systems in (max, +) algebra. *International Journal of Production Research*, 46(19):5337–5348, Oct. 2008. ISSN 0020-7543. doi: 10.1080/00207540802273777. (Cited on page 31).

[58] E. L. Lawler. *Combinatorial optimization: networks and matroids*. Courier Corporation, 1976. (Cited on page 30).

[59] E. Lee. Representing and exploiting data parallelism using multidimensional dataflow diagrams. In *Proceedings of the IEEE International Conference on Acoustics Speech and Signal Processing*, volume 1, pages 453–456 vol.1. IEEE, 1993. ISBN 0-7803-0946-4. doi: 10.1109/ICASSP.1993.319153. (Cited on page 202).

[60] E. Lee and D. Messerschmitt. Synchronous data flow. *Proceedings of the IEEE*, 75(9): 1235–1245, 1987. (Cited on pages 3, 11, 12, 14, 19, 21, 33, 35, 52, 70, 71, 75, 76, 94, 104, and 106).

[61] E. Lee and T. Parks. Dataflow process networks. *Proceedings of the IEEE*, 83(5): 773–801, May 1995. ISSN 00189219. doi: 10.1109/5.381846. (Cited on pages 3, 16, and 52).

[62] E. A. Lee. *A Coupled Hardware and Software Architecture for Programmable Digital Signal Processors*. PhD thesis, University of California, Berkeley, 1986. (Cited on pages 33, 39, 75, and 76).

[63] E. A. Lee and D. G. Messerschmitt. Static Scheduling of Synchronous Data Flow Programs for Digital Signal Processing. *IEEE Transactions on Computers*, C-36(1): 24–35, Jan. 1987. ISSN 0018-9340. doi: 10.1109/TC.1987.5009446. (Cited on page 11).

[64] O. Marchetti and A. Munier-Kordon. Minimizing Place Capacities of Weighted Event Graphs for Enforcing Liveness. *Discrete Event Dynamic Systems*, 18(1):91, 2008. ISSN 0924-6703. (Cited on page 15).

[65] O. Marchetti and A. Munier-Kordon. A sufficient condition for the liveness of weighted event graphs. *European Journal of Operational Research*, 197(2):532–540, Sept. 2009. ISSN 03772217. doi: 10.1016/j.ejor.2008.07.037. (Cited on pages 21, 41, and 99).

[66] O. Marchetti and A. Munier-Kordon. Complexity results for Weighted Timed Event Graphs. *Discrete Optimization*, 7(3):166–180, Aug. 2010. ISSN 15725286. doi: 10.1016/j.disopt.2010.03.006. (Cited on page 21).

[67] D. Meyer. A new class of shift-varying operators, their shift-invariant equivalents, and multirate digital systems. *IEEE Transactions on Automatic Control*, 35(4):429–433, Apr. 1990. ISSN 00189286. doi: 10.1109/9.52295. (Cited on pages 31 and 32).

[68] T. Miyawaki and C. Barnes. Multirate recursive digital filters–A general approach and block structures. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 31(5):1148–1154, Oct. 1983. ISSN 0096-3518. doi: 10.1109/TASSP.1983.1164200. (Cited on pages 31 and 32).

[69] O. Moreira and H. Corporaal. *Scheduling Real-Time Streaming Applications onto an Embedded Multiprocessor*, volume 24 of *Embedded Systems*. Springer International Publishing, Cham, 2014. ISBN 978-3-319-01245-2. doi: 10.1007/978-3-319-01246-9. (Cited on pages 28, 39, and 113).

[70] P. Murthy and E. Lee. Multidimensional synchronous dataflow. *IEEE Transactions on Signal Processing*, 50(8):2064–2079, aug 2002. ISSN 1053-587X. doi: 10.1109/TSP.2002.800830. (Cited on page 202).

[71] M. Nakamura and M. Silva. Cycle time computation in deterministically timed weighted marked graphs. In *Proceedings of the 7th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA '99)*, volume 2, pages 1037–1046. IEEE, Oct. 1999. ISBN 0-7803-5670-5. doi: 10.1109/ETFA.1999.813105. (Cited on pages 21, 35, and 109).

[72] T. W. O'Neil. Unfolding Synchronous Data-Flow Graphs. In *Proceedings of the International Conference on Parallel and Distributed Computing and Systems*, pages 278–283, Dallas, USA, Dec. 2011. ACTAPRESS. ISBN 978-0-88986-907-3. doi: 10.2316/P.2011.757-098. (Cited on pages 38 and 39).

[73] K. Parhi. Algorithm transformation techniques for concurrent processors. *Proceedings of the IEEE*, 77(12):1879–1895, 1989. ISSN 00189219. doi: 10.1109/5.48830. (Cited on page 19).

[74] K. Parhi and D. Messerschmitt. Static rate-optimal scheduling of iterative data-flow programs via optimum unfolding. *IEEE Transactions on Computers*, 40(2):178–195, 1991. ISSN 00189340. doi: 10.1109/12.73588. (Cited on pages 38 and 39).

[75] T. Parks, J. Pino, and E. Lee. A Comparison of Synchronous and Cyclo-static Dataflow. In *Proceedings of the 29th Asilomar Conference on Signals, Systems and Computers*, volume 1, pages 204–210. IEEE Comput. Soc. Press, 1995. ISBN 1-55937-533-7. doi: 10.1109/ACSSC.1995.540541. (Cited on pages 16, 18, 40, and 104).

[76] T. M. Parks. *Bounded Scheduling of Process Networks*. PhD thesis, University of California, Berkeley, Dec. 1995. (Cited on page 16).

[77] J. Pino, S. Bhattacharyya, and E. Lee. A hierarchical multiprocessor scheduling system for DSP applications. In *Conference Record of The Twenty-Ninth Asilomar Conference on Signals, Systems and Computers*, volume 1, pages 122–126. IEEE Comput. Soc. Press, Oct. 1995. ISBN 1-55937-533-7. doi: 10.1109/ACSSC.1995.540525. (Cited on pages 12, 38, 109, 142, 156, 170, and 174).

[78] R. Reiter. Scheduling Parallel Computations. *Journal of the ACM*, 15(4):590–599, Oct. 1968. ISSN 00045411. doi: 10.1145/321479.321485. (Cited on pages 3, 12, 14, 30, 33, 42, 142, 143, and 174).

[79] S. Ritz, M. Pankert, and H. Meyr. High level software synthesis for signal processing systems. In *Proceedings of the International Conference on Application Specific Array Processors*, pages 679–693. IEEE Comput. Soc. Press, 1992. ISBN 0-8186-2967-3. doi: 10.1109/ASAP.1992.218536. (Cited on page 159).

[80] S. Ritz, M. Pankert, V. Zivojinovic, and H. Meyr. Optimum vectorization of scalable synchronous dataflow graphs. In *Proceedings of International Conference on Application Specific Array Processors (ASAP '93)*, pages 285–296. IEEE Comput. Soc. Press, 1993. ISBN 0-8186-3492-8. doi: 10.1109/ASAP.1993.397152. (Cited on pages 38, 39, 103, and 159).

[81] S. Ritz, M. Willems, and H. Meyr. Scheduling for optimum data memory compaction in block diagram oriented software synthesis. In *1995 International Conference on Acoustics, Speech, and Signal Processing*, volume 4, pages 2651–2654. IEEE, 1995. ISBN 0-7803-2431-5. doi: 10.1109/ICASSP.1995.480106. (Cited on page 175).

[82] S. Saha, S. Puthenpurayil, and S. Bhattacharyya. Dataflow Transformations in High-level DSP System Design. In *Proceedings of the International Symposium on System-on-Chip*, pages 1–6. IEEE, Nov. 2006. ISBN 1-4244-0621-8. doi: 10.1109/ISSOC.2006.321985. (Cited on pages 37 and 38).

[83] N. Sauer. Marking Optimization of Weighted Marked Graphs, 2003. ISSN 0924-6703. (Cited on page 21).

[84] R. Schoenen, V. Zivojnovic, and H. Meyr. An upper bound of the throughput of multirate multiprocessor schedules. In *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing*, volume 1, pages 655–658. IEEE Comput. Soc. Press, 1997. ISBN 0-8186-7919-0. doi: 10.1109/ICASSP.1997.599853. (Cited on page 41).

[85] M. Sharir. A strong-connectivity algorithm and its applications in data flow analysis. *Computers & Mathematics with Applications*, 7(1):67–72, 1981. (Cited on page 155).

[86] R. Shenoy, D. Burnside, and T. Parks. Linear periodic systems and multirate filter design. *IEEE Transactions on Signal Processing*, 42(9):2242–2256, 1994. ISSN 1053587X. doi: 10.1109/78.317847. (Cited on page 31).

[87] F. Siyoum, M. Geilen, O. Moreira, and H. Corporaal. Worst-case throughput analysis of real-time dynamic streaming applications. In *Proceedings of the eighth IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis - CODES+ISSS '12*, page 463, New York, New York, USA, Oct. 2012. ACM Press. ISBN 9781450314268. doi: 10.1145/2380445.2380517. (Cited on pages 21 and 37).

[88] S. Sriram and S. S. Bhattacharyya. *Embedded Multiprocessors: Scheduling and Synchronization*. CRC Press, Feb. 2009. ISBN 9781420048018. (Cited on pages 33, 34, 35, 39, 40, 75, 94, 104, 138, and 158).

[89] S. Stuijk, M. Geilen, and T. Basten. Exploring trade-offs in buffer requirements and throughput constraints for synchronous dataflow graphs. In *Proceedings of the 43rd annual conference on Design automation - DAC '06*, page 899, New York, New York, USA, jul 2006. ACM Press. ISBN 1595933816. doi: 10.1145/1146909.1147138. (Cited on page 175).

[90] S. Stuijk, M. C. W. Geilen, and T. Basten. SDF^3: SDF For Free. In *Proceedings of the 6th International Conference on Application of Concurrency to System Design (ACSD)*, pages 276–278. IEEE Computer Society Press, Los Alamitos, CA, USA, June 2006. doi: 10.1109/ACSD.2006.23. (Cited on pages 42, 175, and 191).

[91] S. Stuijk, M. Geilen, and T. Basten. Throughput-Buffering Trade-Off Exploration for Cyclo-Static and Synchronous Dataflow Graphs. *IEEE Transactions on Computers*, 57(10):1331–1345, Oct. 2008. ISSN 0018-9340. doi: 10.1109/TC.2008.58. (Cited on pages 18 and 71).

[92] R. Tarjan. Depth-first search and linear graph algorithms. *SIAM Journal on Computing*, 1(2):146–160, 1972. doi: 10.1137/0201010. (Cited on pages 154, 155, and 177).

[93] E. Teruel, P. Chrzastowski-Wachtel, J. M. Colom, and M. Silva. On Weighted T-Systems. *Application and Theory of Petri Nets*, pages 348–367, June 1992. (Cited on pages 15 and 21).

[94] B. Theelen, M. Geilen, T. Basten, J. Voeten, S. Gheorghita, and S. Stuijk. A scenario-aware data flow model for combined long-run average and worst-case performance analysis. In *Fourth ACM and IEEE International Conference on Formal Methods and Models for Co-Design (MEMOCODE)*, pages 185–194. IEEE, July 2006. ISBN 1-4244-0421-5. doi: 10.1109/MEMCOD.2006.1695924. (Cited on page 21).

[95] L. Thiele and N. Stoimenov. Modular performance analysis of cyclic dataflow graphs. In *Proceedings of the seventh ACM international conference on Embedded software - EMSOFT '09*, pages 127–136, New York, New York, USA, Oct. 2009. ACM Press. ISBN 9781605586274. doi: 10.1145/1629335.1629353. (Cited on page 66).

[96] W. Thies, J. Lin, and S. Amarasinghe. Phased computation graphs in the polyhedral model. Technical report, Technical Report, MIT Laboratory for Computer Science Cambridge, MA 02139, 2002. (Cited on pages 19 and 202).

[97] M. Wiggers, M. Bekooij, P. Jansen, and G. Smit. Efficient computation of buffer capacities for multi-rate real-time systems with back-pressure. In *Proceedings of the 4th international conference on Hardware/software codesign and system synthesis - CODES+ISSS '06*, page 10, New York, New York, USA, 2006. ACM Press. ISBN 1595933700. doi: 10.1145/1176254.1176260. (Cited on page 190).

[98] M. Wiggers, M. Bekooij, and G. Smit. Efficient Computation of Buffer Capacities for Cyclo-Static Dataflow Graphs. In *Proceedings of the 44th ACM/IEEE Design Automation Conference (DAC)*, pages 658–663. IEEE, 2007. ISBN 978-1-59593-627-1. doi: 10.1109/RTAS.2007.12. (Cited on pages 18, 138, 170, 189, and 190).

[99] M. H. Wiggers. *Aperiodic Multiprocessor Scheduling for Real-Time Stream Processing Applications*. PhD thesis, University of Twente, Enschede, The Netherlands, June 2009. (Cited on page 189).

[100] M. H. Wiggers, M. J. Bekooij, P. G. Jansen, and G. J. Smit. Efficient Computation of Buffer Capacities for Cyclo-Static Real-Time Systems with Back-Pressure. In *Proceedings of the 13th IEEE Symposium on Real Time and Embedded Technology and Applications (RTAS)*, pages 281–292. IEEE, Apr. 2007. ISBN 0-7695-2800-7. doi: 10.1109/RTAS.2007.12. (Cited on page 46).

[101] N. E. Young, R. E. Tarjant, and J. B. Orlin. Faster parametric shortest path and minimum-balance algorithms. *Networks*, 21(2):205–221, mar 1991. ISSN 00283045. doi: 10.1002/net.3230210206. (Cited on pages 30, 174, and 178).

[102] G. F. Zaki, W. Plishker, S. S. Bhattacharyya, and F. Fruth. Partial Expansion Graphs: Exposing Parallelism and Dynamic Scheduling Opportunities for DSP Applications. In *Proceedings of the 23rd IEEE International Conference on Application-Specific Systems, Architectures and Processors*, pages 86–93. IEEE, July 2012. ISBN 978-1-4673-2243-0. doi: 10.1109/ASAP.2012.14. (Cited on page 170).

[103] V. Zivojnovic, S. Ritz, and H. Meyr. High performance DSP software using data-flow graph transformations. In *Proceedings of the 28th Asilomar Conference on Signals, Systems and Computers*, volume 1, pages 492–496. IEEE Comput. Soc. Press, Nov. 1994. ISBN 0-8186-6405-3. doi: 10.1109/ACSSC.1994.471502. (Cited on page 19).

List of Publications

[RdG:1] Daniel Rubio Bonilla, Colin W. Glass, Jan Kuper, and Robert de Groote. Introducing and Exploiting Hierarchical Structural Information. In *Proceedings of the 2015 IEEE International Conference on Cluster Computing (CLUSTER)*, pages 777–784. IEEE, September 2015.

[RdG:2] Robert De Groote, Philip K. F. Hölzenspies, Jan Kuper, and Gerard J. M. Smit. Incremental analysis of cyclo-static synchronous dataflow graphs. *ACM Transactions on Embedded Computing Systems*, 14(4):68:1–68:26, December 2015. ISSN 1539-9087. doi: 10.1145/2792981.

[RdG:3] Robert de Groote, Philip K. F. Hölzenspies, Jan Kuper, and H. J. Broersma. Back to Basics: Homogeneous Representations of Multi-Rate Synchronous Dataflow Graphs. In *Proceedings of the 11th ACM-IEEE International Conference on Formal Methods and Models for Codesign (MEMOCODE)*, pages 35–46, October 2013.

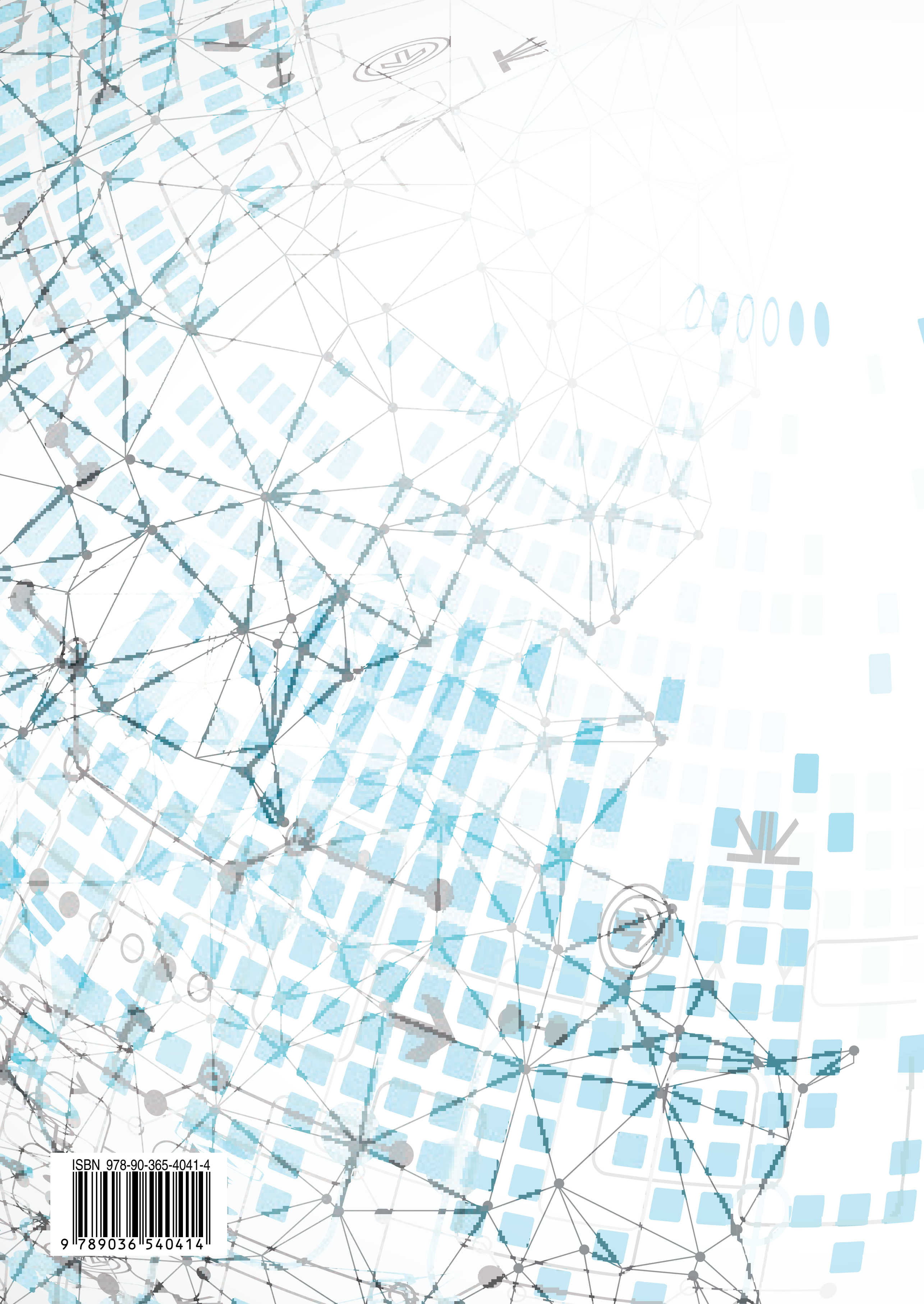
[RdG:4] Robert de Groote, Philip K. F. Hölzenspies, Jan Kuper, and Gerard J. M. Smit. Single-Rate Approximations of Cyclo-Static Synchronous Dataflow Graphs. In *Proceedings of the 17th International Workshop on Software and Compilers for Embedded Systems (SCOPES)*, pages 11–20, June 2014.

[RdG:5] Robert de Groote, Philip K. F. Hölzenspies, Jan Kuper, and Gerard J. M. Smit. Multi-rate Equivalents of Cyclo-Static Synchronous Dataflow Graphs. In *Proceedings of the 14th International Conference on Application of Concurrency to System Design (ACSD)*, pages 62–71. IEEE Computer Society, June 2014.

[RdG:6] Waheed Ahmad, Robert de Groote, Philip K. F. Hölzenspies, Mariëlle I. A. Stoelinga, and Jaco C. van de Pol. Resource-constrained optimal scheduling of synchronous dataflow graphs via timed automata. In *Proceedings of the 14th International Conference on Application of Concurrency to System Design (ACSD)*, pages 72–81, USA, June 2014. IEEE Computer Society.

[RdG:7] Robert de Groote, Jan Kuper, Hajo Broersma, and Gerard J. M. Smit. Max-Plus Algebraic Throughput Analysis of Synchronous Dataflow Graphs. In *Proceedings of the 38th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*, pages 29–38. IEEE, September 2012. ISBN 978-0-7695-4790-9. doi: 10.1109/SEAA.2012.20.

[RdG:8] Robert de Groote, Jan Kuper, and Gerard J. M. Smit. Efficient worst-case timing analysis of synchronous dataflow graphs. In *Proceedings of the Work in Progress Session at the 37th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*, pages 5–6. Johannes Kepler University Linz, 2011. ISBN 978-3-902457-30-1.



ISBN 978-90-365-4041-4

9 789036 540414