Factorisation of discrete dynamical systems

Sara Riva

UNIVERSITÉ CÔTE D'AZUR

ÉCOLE DOCTORALE
SCIENCES ET TECHNOLOGIES DE L'INFORMATION ET DE LA COMMUNICATION

UNIVERSITÀ DEGLI STUDI DI MILANO BICOCCA

THÈSE DE DOCTORAT

Factorisation de Systèmes Dynamiques Discrets

Sara RIVA

Laboratoire d'Informatique, de Signaux et Systèmes de Sophia Antipolis (I3S)
UMR7271 UCA CNRS

Présentée en vue de l'obtention du grade de docteur en INFORMATIQUE **d'**Université Côte d'Azur **et de** Università degli Studi di Milano-Bicocca

Dirigée par : Enrico FORMENTI, Professeur, Université Côte d'Azur
Co-dirigée par : Alberto DENNUNZIO, Maître de conférences, Università degli Studi di Milano-Bicocca
Soutenue le : 21 novembre 2022

Devant le jury, composé de :
Julien CERVELLE, Professeur, Université Paris-Est Créteil
Stefan HAAR, Directeur de recherche, INRIA Saclay-Île de France
Sébastien VEREL, Professeur, Université du Littoral Côte d'Opale
Sylvain SENÉ, Professeur, Aix-Marseille Université
Pedro Paulo BALBI DE OLIVEIRA, Professeur, Universidade Presbiteriana Mackenzie
Sergiu IVANOV, Maître de conférences, Université d'Évry, Université Paris-Saclay

FACTORISATION DE SYSTÈMES DYNAMIQUES DISCRETS

Factorisation of Discrete Dynamical Systems

Sara RIVA

Jury :

Président du jury

Julien CERVELLE, Professeur, Université Paris-Est Créteil

Rapporteurs

Stefan HAAR, Directeur de recherche, INRIA Saclay-Île de France
Sébastien VEREL, Professeur, Université du Littoral Côte d'Opale

Examinateurs

Sylvain SENÉ, Professeur, Aix-Marseille Université
Pedro Paulo BALBI DE OLIVEIRA, Professeur, Universidade Presbiteriana Mackenzie
Sergiu IVANOV, Maître de conférences, Université d'Évry, Université Paris-Saclay

Directeur de thèse

Enrico FORMENTI, Professeur, Université Côte d'Azur

Codirecteur de thèse

Alberto DENNUNZIO, Maître de conférences, Università degli Studi di Milano-Bicocca

---

Université Côte d'Azur

Sara RIVA

Factorisation de Systèmes Dynamiques Discrets

xii+222 p.

Ce document a été préparé avec LaTeX2e et la classe `these-ISSS` version v. 2.10.

Impression : sommaire.tex – 22/12/2022 – 15:23

Révision pour la classe : these-ISSS.cls,v 2.10 2020/06/24 14:16:37 mpelleau

Factorisation de Systèmes Dynamiques Discrets

Résumé

Un Système Dynamique Fini à temps Discret (SDD) est constitué d'un ensemble fini $\mathcal{X}$, dit espace des états, et d'une fonction f, dite fonction de mise à jour (associant à un état v l'état $f(v)$). Les SDD sont un outil formel pour modéliser de nombreux phénomènes en physique, en mathématique, en biologie, et, bien sûr en informatique. Si la formalisation mathématique et les résultats qui en découlent sont élégants et parlants, souvent, ces résultats sont peu applicables en pratique à cause de leur coût computationnel élevé. Dans la littérature, il est connu que les SDD équipés d'opérations de somme et de produit appropriées forment un semi-anneau commutatif. Cette structure algébrique nous permet d'écrire des équations polynomiales dans lesquelles les coefficients et les inconnues sont des SDD. En particulier, si nous sommes intéressés par une certaine dynamique dérivée de données expérimentales, nous pouvons écrire une équation avec celle-ci comme terme de droite constant et modéliser des hypothèses sur la fonction f (ou ses propriétés) dans un terme de gauche polynomial. Trouver des solutions à cette équation permet de mieux comprendre le phénomène et ses propriétés. Cette approche est intéressante mais pose des limites computationnelles importantes. En effet, résoudre une équation polynomiale (à plusieurs variables) est, en général, indécidable et même en se concentrant sur le cas de la validation des hypothèses, le coût computationnel reste élevé. L'idée est alors de chercher des approximations donnant des informations pertinentes sur les solutions de l'équation originale. Trois abstractions (équations plus simples) sont introduites afin d'identifier : le nombre d'états des variables, le comportement asymptotique ou le comportement transient (comportement avant que le système se stabilise). Chaque abstraction est construite d'un point de vue théorique et algorithmique dans le but d'introduire une méthode pour effectuer la validation d'hypothèses sur SDD. Dans cette thèse, il est montré qu'au moyen de transformations algébriques, il est possible d'énumérer les solutions d'une équation polynomiale avec un terme droit constant par l'énumération d'un nombre fini d'équations plus simples. Enfin, le lien entre la résolution ces équations simples et le problème de la cancellation connu en théorie des graphes est exploré. Cela a permis de trouver une borne supérieure linéaire sur le nombre de solutions.

Mots-clés : Systèmes Dynamiques Discrets, Complexité, Vérification.

Factorisation of Discrete Dynamical Systems

Abstract

A Finite Discrete-time Dynamical System (DDS) consists of a finite set $\mathcal{X}$, called state space, and a function f, called next-state map (which associates to a state v the state $f(v)$). DDS are a formal tool for modelling phenomena that appear in Physics, Mathematics, Biology, and, of course, in Computer Science. While the mathematical formalisation and the results that have been found up to nowadays are elegant and meaningful, often they are not very suitable in practice because of their high computational cost. In the literature, it is known that DDS equipped with appropriate sum and product operations form a commutative semiring. This algebraic structure allows us to write polynomial equations in which the coefficients and unknowns are DDS. In particular, if we are interested in some dynamics derived from experimental data, we can write an equation with this as a constant right-hand term and model assumptions about the function f (or its properties) in a polynomial left-hand term. Finding solutions to this equation allow us to better understand the phenomenon and its properties. This approach is interesting but it has important limitations from a computational point of view. Solving a polynomial equation (with several variables) is, in general, undecidable, and even if we focus on the case of hypothesis validation, the computational cost remains high. The idea is then to look for approximations that give relevant information about the solutions of the original equation. It is possible to introduce three abstractions (simpler equations) to identify: the number of states of the variables, the asymptotic behaviour, or the transient behaviour (what happens before the system stabilises). Each one is built from a theoretical and algorithmic point of view to introduce a method to perform hypothesis validation on DDS. In this thesis, it is shown that through algebraic transformations, it is possible to enumerate the solutions of a polynomial equation with a constant term by enumerating a finite number of simpler equations. Finally, the connection between the solution of these simple equations and the cancellation problem known in graph theory is explored. This allowed us to find a linear upper bound on the number of solutions.

Keywords: Discrete Dynamical Systems, Complexity, Formal Verification.

Fattorizzazione di Sistemi Dinamici Discreti

Riassunto

Un Sistema Dinamico finito a tempo Discreto (SDD) è costituito da un insieme finito $\mathcal{X}$, chiamato insieme degli stati, e da una funzione f che associa a uno stato v lo stato $f(v)$. I SDD sono uno modello formale per rappresentare fenomeni che compaiono in Fisica, Matematica, Biologia e, naturalmente, in Informatica. Sebbene la formalizzazione matematica e i risultati ottenuti fino ad oggi siano eleganti e significativi, spesso non sono molto adatti nella pratica a causa del loro elevato costo computazionale. In letteratura è noto che i SDD, dotati di opportune operazioni di somma e prodotto, formano un semianello commutativo. Questa struttura algebrica permette di scrivere equazioni polinomiali in cui i coefficienti e le incognite sono SDD. In particolare, se siamo interessati a una dinamica derivata da dati sperimentali, possiamo scrivere un'equazione con la dinamica come termine destro costante e le ipotesi sulla funzione f (o sulle sue proprietà) in un termine polinomiale a sinistra. La ricerca di soluzioni a questa equazione permette di comprendere meglio il fenomeno e le sue proprietà. Questo approccio è interessante, ma presenta importanti limitazioni dal punto di vista computazionale. La soluzione di un'equazione polinomiale (con diverse variabili) è, in generale, indecidibile e, anche se ci concentriamo sul caso della validazione delle ipotesi, il costo computazionale rimane elevato. L'idea è quindi quella di cercare approssimazioni che diano informazioni rilevanti sulle soluzioni dell'equazione originale. È possibile introdurre tre astrazioni (equazioni più semplici) per identificare: il numero di stati delle variabili, il comportamento asintotico o il comportamento transitorio (ciò che accade prima che il sistema si stabilizzi). Ognuna di esse è costruita da un punto di vista teorico e algoritmico per introdurre un metodo per eseguire la validazione delle ipotesi sui SDD. In questa tesi si dimostra che, attraverso trasformazioni algebriche, è possibile enumerare le soluzioni di un'equazione polinomiale con un termine costante enumerando le soluzioni di un numero finito di equazioni più semplici. Inoltre, viene esplorata la connessione tra queste equazioni semplici e il problema della cancellazione (noto nella teoria dei grafi). Infine, questo permette di trovare un limite superiore lineare al numero di soluzioni.

Parole chiave: Sistemi dinamici discreti, Complessità, Verifica Formale, Diagrammi di decisione, Grafi funzionali.

Acknowledgements

Reading the acknowledgements in friends' theses is always a nice moment, but now that it is my turn, I understand that it is not easy to put everything into a few words.

This has been a personal and professional journey that I have been looking forward to. I consider myself lucky because, despite the ups and downs, I have had the chance to meet unique people. I thank you all. Thank you for being part of this "home" 396 km away from home.

I start by thanking Enrico. From day one of TER, you have shown me the passion for what you do and for that I, hence, thank you very much! I thank you for the advice, the corrections, and the trust you have shown me. I take away with me what you have taught me, I know it will be useful for the future.

Thank you, Alberto, for the trust and the freedom you left me. Thanks to both for a double degree programme that has brought me this experience.

I would like to thank all those who have given me the opportunity to collaborate and grow. Starting from Pedro and Eurico. The experience in São Paulo has been an important opportunity for personal and scientific growth. I deeply thank you for your trust and the time spent together. The advice you have given me has been very useful throughout these three years, and it will continue to be so. *Obrigada*.

A warm thank you goes to the CANA team in Marseille. Thank you, Sylvain, for your support since the beginning, thank you for your suggestions along this path and on the future. Thank you, Kévin. Working with you has taught me a lot, thank you for everything. Thank you, Antonio Enrico P., for these last months of work, for the hours spent behind "our" conjecture and every attempt to prove it. I would like to thank you for your support and the advice on this manuscript as well. *Grazie, Merci*.

Thank you, Jean-Charles. Thank you for leading me towards the magical world of MDD, for your scientific and personal advices even in difficult times, but also for the support in the search for a tomorrow in academia. Thank you really, it means a lot.

All of you co-authors have allowed me to enrich my academic background and have given me the necessary drive to try and dream of an academic future. I thank you for the time, so precious nowadays, that you have given me both, professionally and personally.

I want to thank all the MDSC team permanents at I3S. Over the years, our *alvéole* has been a special place to get useful suggestions over coffees. Special thanks to Cinzia for the chats (also on Sanremo Festival and Eurovision), advice and laughs over the years. Thanks, Bruno, for your support and the chance given to me to help you with EJCIM 2022. It was a precious opportunity.

Thanks also to Jerome for the great support. Together with Sylvain you always had useful advice, thanks to you all the CSIs have been opportunities for personal and professional growth.

Une partie essentielle de ce voyage a été tous les doctorants, amis et les membres actifs de l'association ADSTIC. Adam, Alessio, Alexandre, Aymeric, Amélie, Andrea, Arthur, Assia, Carlotta, Diana, Florian, Foivos, Francesco, Igor, Ingrid, Johann, Jules, Michelangelo, Moudy, Nicolas, Nina, Ophélie, Piergiorgio, Piotr, Romain, Samvel, Steve, Victor, Vladimir, je vous remercie. *Grazie mille*.

Merci Laeti G. de m'avoir accompagné dans l'aventure MT180 et dans les formations les plus étranges.

Remerciements particuliers pour le bureau 222. François et Loïc, vous avez été là dans les bons et les mauvais moments, toujours là pour un mot gentil ou une blague stupide. Ce doctorat n'aurait pas été le même sans vous. Je suis reconnaissante d'avoir eu de tels compagnons d'aventure. François je te remercie tout particulièrement pour les jours où nous avons travaillé ensemble, les discussions que nous avons eues au tableau, tes conseils toujours utiles et ta gentillesse.

Merci Rémy pour ton soutien constant, pour avoir répondu à toutes mes questions, pour avoir écouté et consolé mes inquiétudes. Tu es un ami précieux et un partenaire de voyage unique.

Laeti L. merci pour chaque moment que nous avons passés ensemble, pour m'avoir compris et m'avoir fait sentir spéciale. Nous savons que nous sommes chacune, d'une manière unique, l'équipe de soutien de l'autre et cela vaut plus que mille mots. Merci pour la belle personne que tu es.

Un grazie speciale a Giulia, compagna di isolamento e di risate. Grazie per i tuoi consigli e il tuo supporto.

Grazie Antonia per tutto. Sei una amica, una coinquilina e una collega. Sai capirmi come pochi e hai sempre una parola o una battuta per farmi stare meglio. Questi ultimi anni non sarebbero stati gli stessi senza di te.

Michi e Chiara vi ringrazio per essere sempre state al mio fianco capaci di supportarmi e tifare per me ogni giorno da molti anni. Siete due persone davvero speciali.

Grazie Aurora, Dag, Ilario, Ivan, Nicola e Riccardo per esserci sempre nonostante la distanza.

Ringrazio la mia famiglia (e famiglia acquisita) per avermi supportata durante questo percorso. Un ringraziamento speciale ai miei genitori. Il vostro supporto e il vostro amore sono capaci di coprire qualsiasi distanza.

Voglio ringraziare Emma e Giorgio per sorrisi e risate capaci di risollevarmi e di motivarmi a dare il meglio di me stessa.

Un grazie infinito va alla persona che negli ultimi anni della mia vita è stata il mio riferimento. Grazie Pietro sei la mia forza e la mia motivazione, grazie per avermi seguita in questa mia avventura e in quelle che verranno. Ti sono infinitamente grata.

Grazie a tutti per essere stati parte di questo traguardo. Siete il regalo più bello di questa esperienza. Vi voglio bene.

Contents

Appendix

Introduction

Each of us observes and is surrounded by phenomena that evolve over time. Some are more recognisable than others, but they are everywhere. Let us take a few examples that everyone has experienced: a pandemic, a bank loan, or a simple bouncing ball.

Dynamical systems (DS) are mathematical models whose purpose is to describe a phenomenon and how it evolves. Hence, they can be used, for example, to describe how a disease is transmitted, the characteristics of infective agents, and connections with other social or physiological factors to model the evolution of an infective disease [Ma (2009)]. In the same way, they can be used to describe how a debt situation with a bank changes (month after month) according to the interest rate and the part of the loan already returned [Bournez et al. (2022)], or how a bouncing ball moves through space (instant after instant) according to the rules of physics [Okninski and Radziszewski (2010)]. Through these few examples, one can understand that DS have applications in various fields, such as mathematics, physics, biology, chemistry, economics, medicine, and many others. The *Dynamical System Theory* arose in the 19th century from an interest in phenomena coming from astronomy. For this reason, it is common to find the birth of the study of DS indicated in Newtonian Mechanics. However, it is widely recognised that the modern theory of dynamical systems derives from the work of Jules Henri Poincaré [Poincaré (1892), Poincaré (1893), Poincaré (1899)] on the three-body problem of celestial mechanics *i.e.* the problem of determining (according to an initial position, speed, and masses) the movements of three elements which attract each other according to Newton's laws of motion and universal gravitation.

Another aspect that the previous examples hint is the importance of time in a DS. Indeed, DS can be divided into two large families: *continuous-time* and *discrete-time dynamical systems*. Regardless of how time is defined, the idea of a DS is to associate a state (a "snapshot") of the system with each instant in time. The set of possible states of a system is called *state space*, and the transitions within this set are governed by differential equations, in the continuous-time case, and by relations, in the discrete one [Devaney (2018), Sandefur (1993)].

One of the main goals with these models has always been to study the long-term dynamics of a modelled phenomenon in order to forecast future states and their properties. A fundamental concept, in dynamical system theory, is the notion of *orbit*, namely, given an initial state, the sequence of states that the system will pass through along time. This term also harkens back to the origins of DS.

However, before the advent of computers, studying the evolution of DS required fine mathematical skills and could only be done for a few systems. Now, instead, one can study DS and derive information about their future behaviour and predict, for example, whether the system will enter a cyclic behaviour or whether it will stabilise (*i.e.*, whether it will remain closer to a certain orbit). These are not trivial computations in many application cases, and researchers try to introduce always new techniques to be able to perform increasingly more complex predictions on real-life systems using more and more powerful machines and approximation techniques [Antoulas (2004)]. For example, in 2009, two french researchers have been able to study the positions of some planets of the solar system in 5 billion years [Laskar and Gastineau (2009)].

After modelling a phenomenon, asking what happens in the long-term dynamics is surely the most natural question. Nevertheless, another interesting research goal consists in inferring the

rules governing a DS from a few information about it [Wang et al. (2021), Ahmadi and Khadir (2020)].

The advent of computers has also given greater importance to discrete-time dynamical systems [Jakubczyk and Sontag (1990)]. Indeed, although there are phenomena that are more naturally modelled by continuous-time DS, their discrete version is known to be an excellent approximation as it is computationally simpler and still allows the study of certain aspects of the dynamics (such as stability). Intuitively, the transition from continuous to discrete corresponds to a "sampling", meaning that the state of the system is observed after regular intervals of time.

In this thesis, we will focus on Discrete-time Dynamical Systems with a finite number of states $\mathcal{X}$, which we will call DDS. This type of dynamical systems can be seen as graphs, called *dynamics graphs*, where the set of nodes is the set of states (*i.e.* the state space) and the pairs $(v, f(v))$ with $v \in \mathcal{X}$ are the edges. Then, each node has one and only one outgoing arc (which can be a self loop).

In graph theory, these oriented graphs are called *functional graphs* (FG). They correspond to permutations or map models, from a phase space with a finite number of points to itself (*i.e.* endofunctions). In this case, the set of nodes is finite, hence, each state is *ultimately periodic i.e.* each long enough orbit originating in a state will eventually run into a cyclic behaviour. In the literature, it is well known that these graphs are characterised by a finite number of components with just one cycle. Then, the nodes of each component can be classified as *periodic* if they belong to a cycle, or *transient* otherwise.

From a mathematical point of view, FG (or DDS) are simple combinatorial structures. However, they find important applications to represent the dynamics arising from models such as Boolean automata networks, cellular automata [Sené (2012), Alonso-Sanz (2012), Adamatzky et al. (2020)], as well as real-world phenomena such as genetic regulatory networks, epidemic models, plankton population, interactions between market demand and market supply for a given good or service, and many others [Finkenstädt and Grenfell (2000), Bower and Bolouri (2004), Kartal et al. (2016), Danca and Fečkan (2019)].

An important research direction has been to study the expected number of components of a random map (*i.e.* random FG). This question has been opened by Metropolis and Ulam [Metropolis and Ulam (1952)] and answered by Kruskal one year later [Kruskal (1954)]. Since then, a lot of researchers have studied the number of cycles, trajectories, and the number and the dimension of the components of random generated FG [Romero and Zertuche (2003), Romero and Zertuche (2005), Derrida and Flyvbjerg (1987)] providing exact and asymptotic formulas. The goal of this type of research is to try to answer questions such as "given any state, what is the likely amount of states the system will pass through before exhibiting cyclic behaviour?" or "what is the probability that the system will reach a fixed point?", and similar. In the case of trajectories, typically, one can try to identify the average number of different states that may be involved in the orbit of a state, or the average number of nodes that may reach a state with their orbit. These questions have also been studied directly on DDS corresponding to Boolean functions and explaining the meaning in a biological context [Coste and Henon (1986)]. It is important to emphasise that given a number of states n, the number of n-functional graphs, or the number of (non-isomorphic) DDS with n states follows a known asymptotic formula [The Online Encyclopedia of Integer Sequences (1996)a].

Considering functional graphs, the components are a decomposition of the set of nodes in disjoint minimal non-empty invariant sets[1]. Given this important aspect, Katz studied the probability that a random mapping is *indecomposable* (*i.e.* the graph has just one component) [Katz (1955)].

Cycles of DDS (or FG) represent the asymptotic behaviour of the dynamics and they are particularly important, for example, when the dynamics modelled in an FG arise from biological applications. In fact, they have been linked to biological phenotypes and they can provide important insights for biologists to understand molecular mechanisms underlying many cellular processes such as cellular division, differentiation, and others [Schwab et al. (2020)b]. For this reason, researchers propose algorithmic techniques to be able to extrapolate the cyclic behaviour of a DDS arising from a biological context directly from networks able to model the interaction between genes involved in a process, for example. Some of these techniques are based on the SAT problem [Dubrova and Teslenko (2011)], others on Binary Decision Diagrams [Garg et al. (2008), Zheng et al. (2013), Garg et al. (2007)].

In the state of the art, there are also works investigating the possibility of introducing algebraic structures on DDS [Ginocchio (1998)]. In particular, in 2018, it was shown how DDS (up to isomorphism) equipped with sum and product operations form a commutative semiring [Dennunzio et al. (2018)]. This result derives from the idea of being able to find dynamical systems that can be combined to recreate a certain dynamical system of interest (up to isomorphism).

The most natural and intuitive way to combine two dynamics is to consider them either in parallel, *i.e.* at each time step both systems change state synchronously, or in an alternative execution, *i.e.* one of the original dynamics is observed as time passes according to the initial state. These two options of combining DDS, to create larger and more complex dynamics, correspond exactly to the two operations of the semiring: product and sum. Specifically, the sum is defined as the disjoint union of the components, while the product is defined from a Cartesian product of the states of the original dynamics, and the possible transitions are hence given by the Cartesian product of the original transitions.

Beyond the interesting algebraic result, this commutative semiring structure offers the possibility of investigating decompositions and/or factorisations of DDS. In fact, it naturally brings to define polynomials over DDS in which both coefficients and unknowns terms are DDS. In particular, it becomes possible to use a polynomial equation with a constant right-hand side to investigate whether a certain dynamic (contained in the right-hand term) can be produced by other DDS expressed by the polynomial. Only by finding values for the variables in the polynomial is it possible to understand whether the DDS can be factorised and/or decomposed. As may have already been guessed, these polynomial equations are particularly interesting as they allow one to model even more complex *hypotheses* than simply "is it factorisable?", or "is it decomposable?". This is thanks, for example, to the coefficients of the polynomial. Indeed, coefficients can be interpreted as subdynamics that should cooperate to create the dynamics of interest. This can be particularly useful in all those cases in which the DDS being studied is obtained based on partial knowledge, or when one wishes, for example, to identify whether there are common subsystems between different DDS.

The problem at this point becomes the complexity of solving equations on DDS. In [Dennunzio et al. (2018)], it was indeed shown that polynomial equations without a constant term are undecidable and, in the case there is a constant term, they are in NP (and in some cases even NP-complete).

[1] Given $\mathcal{X}' \subseteq \mathcal{X}$, $\mathcal{X}'$ is an invariant set if and only if $f^{-1}(\mathcal{X}') \subseteq \mathcal{X}'$.

Since we have said that a DDS can be viewed as a functional graph, one might wonder why we are not studying whether a DDS can be factorised using graph factorisation techniques. Indeed, the product as defined in [Dennunzio et al. (2018)] corresponds to the *direct product* of graphs. The problem of factorisation of graphs is widely studied from a theoretical point of view [Hammack et al. (2011)], and some heuristics have been proposed for the factorisation of graphs with respect to the direct product [Calderoni et al. (2021)]. However, this approach is not exactly equivalent to studying factorisations and decompositions using multivariate polynomials and does not allow us to take advantage of the fact that we are dealing with functional graphs, *i.e.* graphs whose important property is that all nodes have outgoing degree one.

This thesis studies multivariate polynomial equations with a constant right-hand term. As a starting point, equations in which there is one variable per monomial are considered, leaving the more general case for future work. Therefore, the equations will be of form

$$A_1 \cdot X_1^{w_1} + A_2 \cdot X_2^{w_2} + \ldots + A_m \cdot X_m^{w_m} = B$$

where coefficients A_z, variables X_z, and the known dynamics B are DDS. Our goal is a solution based on *abstractions*. The term abstraction derives from the fact that the equation on DDS will be studied by means of other equations that will focus on specific properties of the systems involved while ignoring other aspects. This is done to reduce the solution space one property at a time and to give relevant information about the solutions of the original DDS equation. An additional goal is set in this work: the enumeration of solutions. This is because the aim is to enumerate systems capable of recreating a dynamics or validating a hypothesis modelled by an equation. It should be noted that in order to enumerate the solutions of the original equation on DDS, one needs to enumerate the solutions of the abstractions, since the original solutions have to satisfy all of them (*i.e.* solutions can be found in the intersection of the solutions of all the abstractions).

We will study three main aspects of the dynamics involved in an equation: the cardinality of the set of states, the cyclic behaviour, and the transient behaviour. For each of these, a specific equation (c-abstraction, a-abstraction, and t-abstraction) is then introduced to identify the cardinalities of the set of states, the cyclic or the transient behaviour of the different unknowns in the polynomial. Each abstraction is studied, first, from a theoretical point of view to be able, later, to propose methods to solve the equations over the abstraction and to perform hypothesis validation on DDS.

Recall that, in general, an equation models an hypothesis like *"does the dynamics that we observe B result from several independent smaller systems $A_z \cdot X_z^{w_z}$, each of them having its dynamics determined by the joint parallel action of a known part A_z and an unknown part $X_z^{w_z}$ to be computed?"*.

This work leads to the implementation of a pipeline capable of validating hypotheses on DDS. Through algebraic transformations, called *contraction steps*, we enumerate the solutions of a polynomial equation with a constant right-hand term examining a finite number of simpler equations, typically of the form $A \cdot X = B$ where A, X, and B are DDS. For each abstraction, we show how to solve the corresponding basic equations, and how to combine their solutions to find the solutions of the original equation. In the case of the number of states and of the asymptotic behaviour, Multi-valued Decision Diagrams (MDD) are used to identify the solutions. Furthermore, it is also presented how to compute roots on the asymptotic behaviour of DDS. Finally, the connection between the solution of simple equations on DDS and the cancellation problem in graph theory is explored. Thanks to this connection, a linear upper bound on the number of solutions of basic equations is introduced.

In this work, we will therefore explore the connection between MDD and equations over DDS. MDD are a generalisation of Binary Decision Diagrams (BDD) [Akers (1978), Bergman et al. (2016)]. They are a data structure used to obtain efficient representations of functions (with finite domains) or finite sets. In the last years, MDD have been applied in many disparate research domains proving the potential of this structure. MDD are used, for instance, to improve random forest algorithms by replacing the classic binary decision trees [Nakahara et al. (2017)], to represent and analyse automotive product data of valid/invalid product configurations [Berndt et al. (2012)], and to perform trust analysis in social networks [Zhang et al. (2019)]. MDD find applications also in the analysis of discrete dynamical systems. Indeed, in [Naldi et al. (2007)], MDD represents logic rules to analyse some properties and dynamics aspects in the case of regulatory networks (for example, to perform a stable states identification). A crucial aspect of MDD is the exponential compression power of the reduction operation and the fact that many classical operations (intersection, union, *etc.*) can be performed without decompression. These aspects will be exploited in this thesis. In addition, a new algorithmic technique for calculating the intersection on structures containing Cartesian products of sets of ordered solutions will be proposed in this thesis.

This manuscript hence comprises three chapters for the state-of-the-art part. In particular:

- Chapter 1 introduces all the necessary concepts concerning DDS, the sum and product operations, the commutative semiring, and known complexity results about equations on DDS. It also presents two examples of discrete dynamics obtained from Boolean networks and cellular automata.

- Chapter 2 is concerned with useful concepts on the direct product of graphs and, in particular, on the cancellation problem. These notions will be useful to study the number of solutions of an equation $A \cdot X = B$ where A, X, and B are DDS with one component.

- Chapter 3 is a brief introduction to MDD. These structures will be used for the solution enumeration of equations on the cardinality of the set of states and cyclic behaviour. For this reason, useful operations on these structures are also presented such as reduction, intersection, Cartesian product, and others.

The main contributions of this thesis are contained in the next three chapters:

- Chapter 4 introduces the multivariate polynomial equations over DDS with constant right-hand term that will be considered. It also explains the idea behind the introduction of the abstractions and those that will be studied in the thesis work. Finally, it defines the c-abstraction and an MDD-based method to enumerate its solutions.

- Chapter 5 is concerned with the a-abstraction. It introduced a handy notation to express the cyclic part of DDS and studies the relation between these parts and the sum and product operations. The chapter also presents the algebraic transformations (contraction steps) that allow to enumerate the solutions of the abstraction via a finite number of basic equations. Moreover, two algorithmic techniques for solving basic equations are presented. A first technique that explores the connection with the change-making problem, and a second one (based on MDD) that leads to better performances. Finally, Section 5.3 focus on the pipeline required to enumerate the solutions of an a-abstractions equation, and Section 5.4 presents a technique to compute roots on the cyclic part of DDS.

- Chapter 6 is devoted to the t-abstraction on the transient behaviour and analyses the impact of the product operation on the dynamics. It introduces the basic case by showing how the algebraic transformations shown in the previous chapter can also be applied to the transient part. Then, an upper bound on the number of solutions of basic equations over DDS is introduced by exploiting a connection with the cancellation problem over graphs. The chapter ends with the proposal of two algorithmic approaches: a polynomial one to enumerate the solutions of basic equations on t-abstractions, and a second one to enumerate the solutions (up to isomorphism) of a basic equation on DDS (having single components) which has exponential complexity.

Finally, the Appendix A presents the results of research activities carried out during the thesis period concerning Elementary Cellular Automata and their sensitivity to synchronism. In particular, the ratio between the different dynamics obtainable and the number of different feasible asynchronous updates is investigated. For this analysis, asynchronous updates based on deterministic block-sequential update schedules are considered, meaning that the cells are partitioned into disjoint blocks with different priorities of being updated.

Notations

Discrete Dynamical Systems

$S = (\mathcal{X}, f)$	A generic DDS
$\mathcal{X}, \mathcal{Y}$	Set of states
f	Next state map
G_S	Dynamics graph of S
V	Set of nodes
E	Set of edges
$f\|_{\mathcal{Y}}$	Restriction of f to $\mathcal{Y}$
$(\mathcal{Y}, f\|_{\mathcal{Y}})$	Dynamical subsystem induced by $\mathcal{Y}$
v	A generic state
$\mathcal{P}$	Set of periodic states
$\mathcal{T}$	Set of transient states
$\mathcal{C}$	Set of nodes of a cycle
p	Length of a cycle
h	Length of a transient
L	Number of weakly connected components
$\left(\mathcal{X}_j, f\|_{\mathcal{X}_j}\right)$	j-th component of a DDS, with $j \in \{1, \ldots, L\}$
$\mathcal{C}_j$	j-th cycle of a DDS, with $j \in \{1, \ldots, L\}$
$\mathcal{T}_j$	Transient points of the j-th component, with $j \in \{1, \ldots, L\}$
$\mathcal{D}$	Commutative semiring of DDS
$\times$	Cartesian product
$\mathcal{N}$	A semiring isomorphic to the natural numbers

Equations over DDS

$+$	Sum operator
$\sqcup$	Disjoint union of sets
$\cdot$	Product Operator
X	Unknown DDS (a variable in the equation)
$\mathcal{D}[X]$	Semiring of polynomials over the indeterminate X
$P(X_1, \ldots, X_\nu), Q(X_1, \ldots, X_\nu)$	Two polynomials of $\mathcal{D}[X_1, \ldots, X_\nu]$
ν	Number of different variables in a polynomial
m	Number of monomials in the equation
B	Constant right-hand term of an equation
A	Coefficient of a monomial in the equation
S^w	Product of w copies of the DDS S

Products and Cancellation over graphs and digraphs

G, H	Graphs or Digraphs
$V(G)$	Set of vertices of G
$E(G)$	Set of edges (or arcs) of G
g	A vertex of G
h	A vertex of H
(g_1, g_2)	An oriented arc of G
$[g_1, g_2]$	An edge of G
$G \cdot H$	Direct product of G and H
$G \,\square\, H$	Cartesian product of G and H
$G \boxtimes H$	Strong product of G and H
$d_G(g_1, g_2)$	Distance between two nodes in G
l	Length of a walk or a path
$\rho_i(g)$	i-th component of a state $g \in V(G)$ with $G = G_1 \cdot \ldots \cdot G_k$
L	Number of cycles
$\mathsf{g}(G)$	$\gcd$ between cycles length in G
C	Strong component
$hom(J, G)$	Number of homomorphisms from J to G
$\cong$	Isomorphism of graphs
φ	An homomorphism
μ	An anti-automorphism
$Ant(G)$	Set of anti-automorphism of G
$\simeq$	Equivalence between anti-automorphisms
$[\mu]$	Equivalence class of anti-automorphisms containing μ
$G!$	Factorial of a graph G
$Perm(V(G))$	Set of permutations of $V(G)$
π	A permutation of $Perm(V(G))$
C_p	Oriented cycle of length p
P_n	Oriented path with n vertices and $n-1$ arcs
V_r^t, Λ_r^t	Two non-isomorphic families of cancellation digraphs

Binary Decision Diagrams & Multi-valued Decision Diagrams

f_B	Boolean formula
f_M	Multi-valued formula
b_i	i-th variable
$\mathtt{0}, \mathtt{1}$	$false, true$
$\bot, \top$	Terminal nodes of a BDD
root	Root of a Decision Diagram
tt	True terminal node of an MDD
M	An MDD
N	Set of nodes of an MDD
E	Set of arcs of an MDD
α, β, ω	Nodes of an MDD
E_α^+	Ordered set of outgoing arcs from the node α
E_α^-	Ordered set of incoming arcs in the node α
r	Number of variables in the formula
D_i	i-th variable domain
D	Larger variable domain
d	Element of a domain (label of an edge)
$M_1 \times M_2$	Cartesian product of MDD
$M_1 \cup M_2$	Union of MDD
$M_1 \cap M_2$	Intersection of MDD
$M_1 - M_2$	Difference between MDD
$M_1 \triangle M_2$	Symmetric difference between MDD

Change-Making Problem

T	Total integer sum
\$	Coin system
c_i	The i-th coin
t	Number of different coin denomination in \$

C-abstraction

$\|S\|$	C-abstraction of a DDS S
$\|S_1\| + \|S_2\|$	Sum of c-abstractions
$\|S_1\| \cdot \|S_2\|$	Product of c-abstractions
B_z	Number of states of B generated by the z-th monomial
D_i	Possible labels of the outgoing arcs from a layer i of the MDD
$\ell ab((\alpha, \beta))$	Label of the arc (α, β) of the MDD
$val(\alpha)$	Value associated to a node α

A-abstraction

$\mathring{S}$	A-abstraction of a DDS S
$\mathring{S}_1 \oplus \mathring{S}_2$	Sum of a-abstractions
$\mathring{S}_1 \odot \mathring{S}_2$	Product of a-abstractions
C_p^n	Union of any n disjoint cycles of length p
l_S	Number of different cycle lengths in S
$\mathfrak{m}$	Number of different pairs $(\mathring{X}, w)$ in the equation
ℓ_z	Number of monomials with the z-th pair $(\mathring{X}, w)$
$\mathring{\mathbb{X}}_z$	The z-th pair $(\mathring{X}, w)$
$p_{z,i}$	The i-th period length involved in a monomial with $\mathring{\mathbb{X}}_z$
$n_{z,i}$	Number of cycles of length $p_{z,i}$
p_j	The j-th cycle length involved in B
n_j	Number of cycles of length p_j in B
$n_j^{z,i}$	Number of cycles of length p_j generated by the monomial (z, i)
p, n, q	Parameters of a basic equation $C_p^1 \odot \mathring{\mathbb{X}} = C_q^n$
C_q^r	Possible subset of C_q^n with $r \leq n$
s	Number of cycles in a solution of a basic equation
p'	A cycle length in a solution
$\mathsf{p}_1^{h_1}\mathsf{p}_2^{h_2}, \ldots, \mathsf{p}_\psi^{h_\psi}$	Prime factorisation of p
factors(p)	Set of factors $\mathsf{p}_1, \mathsf{p}_2, \ldots, \mathsf{p}_\psi$ of p
$\mathsf{n}_1^{t_1}\mathsf{n}_2^{t_2}, \ldots, \mathsf{n}_\tau^{t_\tau}$	Prime factorisation of n
$\mathsf{q}_1^{o_1}\mathsf{q}_2^{o_2}, \ldots, \mathsf{q}_\iota^{o_\iota}$	Prime factorisation of q
$\Pi_{\mathbf{F}}(q, p)$	Function to compute the product of $\mathsf{q}_j^{o_j}$ with $\mathsf{q}_j \notin$ factors(p)
$\Pi_{\mathbf{E}}(q, p)$	Function to compute the product of $\mathsf{q}_j^{h_i}$ with $\mathsf{q}_j = \mathsf{p}_i$
$div(q, n)$	Set containing the divisors of q smaller than or equal to n
CTM$_{p,q,n}$	Solutions set returned by the Colored-Tree Method
EQ$_{p,q,n}$	Solutions set of a basic equation
$M_{p,q,n}$	SB-MDD of a basic equation
$D_{p,q}$	Set of feasible divisors of a basic equation
Z	Number of layer in a $M_{p,q,n}$
$e(\alpha)$	Label of the incoming edge of a node α in $M_{p,q,n}$
CS	MDD containing all feasible systems according to the necessary equations
CS_j	MDD containing all feasible ways to generate the $C_{p_j}^{n_j}$ cycles using the monomials
N_j, E_j	Nodes and arcs of CS_j
$N_{j,(z,i)}$	A Layer of CS_j
$D_{p_{z,i},p_j}$	Possible labels of the outgoing arcs of the level $N_{j,(z,i)}$
$\overline{M}$	Set containing SB-MDD and SB-Cartesian MDD
M	An MDD in $\overline{M}$
$\overline{\mathcal{S}}$	Initial guess of the SB-Cartesian Intersection
$\mathcal{S}$	A solution in $\overline{\mathcal{S}}$

T-abstraction

$\check{S}$	T-abstraction of a DDS S
g	Indexing function for cycles
$g(r)$	Periodic point with index r
$\mathcal{T}_{z,j}$	Set of transient points of the j-th component in a DDS z
$\mathcal{T}^{z,j}$	Matrix of transient points of the j-th component in a DDS z
$\mathcal{T}^{z,j}_{r,h}$	Set of transient points of the j-th component in a DDS z with a path of length h ending in $g(r)$
$T^{z,j}$	T-abstraction matrix of the j-th component of DDS z
$T^{z,j}_{r,h}$	Multiset of the number of predecessors of each node in $\mathcal{T}^{z,j}_{r,h}$
h^{max}_j	Maximum length of a transient in the j-component
$\mathcal{T}_z$	Set of transient points of the connected DDS z
$\mathcal{T}^z$	Matrix of transient points of the connected DDS z
T^z	T-abstraction matrix of a connected DDS z
$\mathcal{P}_z$	Periodic point of a DDS z
$\mathcal{M}$	A multiset of integers
d	An element of $\mathcal{M}$
$\mathcal{M} + \mathcal{M}'$	Disjoint union of multisets
$\mathcal{M} \otimes \mathcal{M}'$	Product of multisets
$\check{B}^{1,1}, \check{B}^{1,2}, \ldots, \check{B}^{m,L_m}$	Elements of the partition of the multiset $\check{B}$
T^A, T^X, T^B	T-abstraction matrices of a basic equation
$A \cdot X \supseteq B$	Product $A \cdot X$ contains a subset of components isomorphic to B
$U_v(S)$	Unroll of a DDS S from v
I	An in-tree
$\mathcal{V}$	Set of vertices of an in-tree
$\mathcal{E}$	Set of edges of an in-tree
$I_1 \star I_2$	Product of in-trees
cut	Function cut
$C_{a,t}(A)$	The cut of an unroll $U_a(A)$, at a level t
F	A finite in-tree
$Hom(G, F)$	Set of homomorphisms from G to F
π	A projection
a, x, b	Periodic points of A, X, and B
$R_\ell(I)$	The roll (of length ℓ) of I
p_X	Number of cyclic point in X
$\mathcal{M}_1, \mathcal{M}_2, \mathcal{M}_3$	Multisets for the reconstruction of X
$\mathcal{O}_{r,h}$	Set of different origins of the nodes in a $\mathcal{T}^B_{r,h}$
o	An element of $\mathcal{O}_{r,h}$
$\mathcal{T}^B_{r,h\|o}$	Set of all nodes $\mathcal{T}^B_{r,h}$ such that $f_B(v)$ has origin o
$T^B_{r,h\|o}$	Multiset of indegrees of the nodes in $\mathcal{T}^B_{r,h-1\|o}$
$\mathscr{H}(v)$	The height of the subtree rooted on v

Cellular Automata & Boolean Networks

n	number of components of a BN (or cells of a CA)
F	Global rule
$\{0,1\}$	Set of states for the components
$[\![n]\!]$	$\{0,\ldots,n-1\}$
x_i	The current state of the i-th automaton
x	A configuration of the BN or of a CA
f_i	Local function of the i-th automaton
G_F	Interaction graph
α	Number of a Elementary Cellular Automata (ECA)
r_α	Local function of a ECA
$\tau_\iota, \tau_\varrho, \tau_\eta, \tau_{\varrho\eta}$	Tranformations to preserve dynamics of a ECA
P_n	The set of ordered partitions of $[\![n]\!]$ cells
Δ	Block-sequential update schedule
Δ_i	A block of Δ
Δ^{sync}	The synchronous update schedule
$F^{(\Delta)}$	Global rule according to Δ

Sensitivity to synchronism in ECA

$\overline{x}^i$	Configuration obtained from x by flipping the state of component i
$\hat{G_F}$	G_F without loops
$D_{F(\Delta)}$	Transition digraph
$\mathcal{D}(F)$	Set of dynamics of a CA
$U_{F(\Delta)}$	Update digraph
$lab_\Delta((i,j))$	label of the arc (i,j) in $U_{F(\Delta)}$
$\mathcal{U}(F)$	Set of equivalence classes of update schedules for a CA
$\mu_S(F)$	The sensitivity to synchronism
$f^{(\Delta)}(x)_i$	Updated value of the component i according to Δ
G_n^{ECA}	Interaction digraph of size n for ECA
$\mathcal{U}^{ECA}(F)$	Set of valid labeling of G_n^{ECA}
$F_{\alpha,n}$	Global function of an ECA α with n cells
$\mu_S(F_{\alpha,n})$	The sensitivity to synchronism of ECA rule number α
$\overleftarrow{\mathtt{d}}_\Delta(i), \overrightarrow{\mathtt{d}}_\Delta(i)$	Length of the chains of influences for a cell i
$\Delta \equiv \Delta'$	Update schedules equivalent
$d_\Delta(i)$	Cells on which the result of updating cell i depends
$\mathcal{D}_u(F_{8,n})$	Partition of $\mathcal{D}(F_{8,n})$
$\sigma(\Delta)$	Left rotation
$\rho(\Delta)$	Left/right exchange

State-of-the-art

CHAPTER 1

Discrete Dynamical Systems

Finite Discrete-time Dynamical Systems (DDS) are a formal model for studying complex phenomena such as those appearing in Physics, Biology, Chemistry, etc. They are based on a finite set of states *and a function, called* next-state map*, that describes how a phenomenon evolves from the current state to the next one. The* dynamics, i.e. *the overall evolution of the system, can be represented with a functional graph,* i.e. *a digraph in which all nodes have an outgoing degree equal to 1. These systems can be combined using sum and product operations, which allow us to incorporate different dynamics in alternative or in parallel executions. Finite Discrete-time Dynamical Systems equipped with these operations form a commutative semiring. This algebraic structure naturally leads to use polynomial equations over DDS for modelling questions on the dynamics and the structure of the system. We will see that according to the structure of these equations, the problem of deciding if they have solutions might span from undecidability to decidability (with different degrees of complexity). The chapter will end with some basic examples.*

1.1 Finite Discrete Dynamical Systems

A dynamical system (DS) is a formal model used to represent the evolution of a phenomenon over time. To do so, by convention, *time* can be primarily discrete or continuous. Continuous-time DS are governed by differential equations, contrarywise, in discrete-time DS, the evolution progress at discrete time steps and ordinary continuous functions (or relations) are used. A dynamical system is based on a *state space i.e.* the set of possible states of the system (the observed dynamics) at some moment in time. State changes follow a *fixed rule* that describes how (according to certain parameters) the system evolves. In other words, this rule defines the next state of the system.

This is a simple introduction to dynamical systems, for more details, the reader can delve into the subject in one of the classic books [Devaney (2018), Sandefur (1993)].

In the following, we will always refer to dynamical systems with *finite* state space and *discrete* time. We will call these *Finite Discrete-Time Dynamical Systems*, Discrete Dynamical Systems (DDS) or dynamical systems for simplicity. This chapter is intended to introduce the essential concepts, which will be useful in the next chapters.

1.1.1 Basic notions

Definition 1.1.1 (DDS)**.** A **Finite Discrete-Time Dynamical System** S is a pair $(\mathcal{X}, f)$ where $\mathcal{X}$ is a finite set of states, and $f : \mathcal{X} \to \mathcal{X}$ is a function, called *next-state map*, which defines the successor for each state of $\mathcal{X}$.

A DDS $S = (\mathcal{X}, f)$ can also be represented by a digraph $G_S = (V, E)$, called the **dynamics graph**, where the vertices are the possible states of the system (*i.e.* $V = \mathcal{X}$), and the function f defines the existing edges, or in other words, $E = \{(v, f(v)) \,|\, v \in V\}$. According to this definition, all the nodes of the digraph have outgoing degree equal to 1, and it is planar. In the literature, this type of graphs are also called *functional graphs* since they are the graph of the corresponding functions.

Given any subset $\mathcal{Y}$ of $\mathcal{X}$ (such that for all $v \in \mathcal{Y}$, $f(v) \subseteq \mathcal{Y}$), the DDS $(\mathcal{Y}, f|_{\mathcal{Y}})$ is said to be the **dynamical subsystem** of $(\mathcal{X}, f)$ induced by $\mathcal{Y}$ (here, $f|_{\mathcal{Y}}$ means the restriction of f to $\mathcal{Y}$). Clearly, the dynamics graph of $(\mathcal{Y}, f|_{\mathcal{Y}})$ is the subgraph of G_S induced by $\mathcal{Y}$. Remark that the most simple system is the empty dynamical system, in which $\mathcal{X}$ is the empty set and f is the empty function (a function whose domain is the empty set).

Since $\mathcal{X}$ is finite, each state $v \in \mathcal{X}$ is **ultimately periodic**, *i.e.* each long enough path in the graph originating in v will run at some point into a cycle.

Let f^n denote the n-th composition of the function f with itself, where f^0 is the identity map on $\mathcal{X}$. The **orbit** of an initial state $v_0 \in \mathcal{X}$ is the sequence $(v_n)_{n\in\mathbb{N}}$ where $v_n = f^n(v_0)$. A state $v \in \mathcal{X}$ is a **periodic point** of S if there exists an integer $p > 0$ such that $f^p(v) = v$. The smallest such p is called the **period** of v. If $p = 1$, the state v is called a **fixed point**.

A *cycle* (of length p) of S is any set $\mathcal{C} = \{v, f(v), ..., f^{p-1}(v)\}$ where $v \in \mathcal{X}$ is a periodic point of period p. Then, formally, a point v is said to be ultimately periodic if $f^{h+p}(v) = f^h(v)$ for some integer $h \in \mathbb{N}$ and $p > 0$, or, in other words, if there exists a cycle $\mathcal{C}$ in G_S such that $f^h(v) \in \mathcal{C}$ for some $h \in \mathbb{N}$.

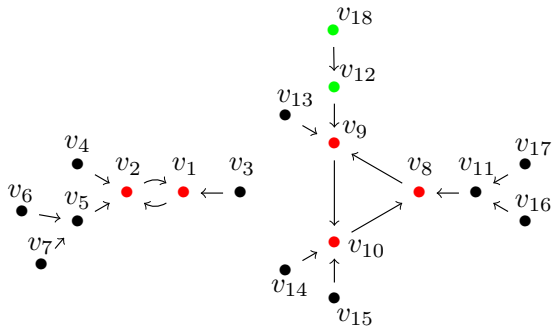


Figure 1.1 – A dynamics graph G_S corresponding to a DDS S with two cycles and some transients nodes connected to both cycles. We have $S = (\mathcal{X}, f)$ with $\mathcal{X} = \mathcal{P} \cup \mathcal{T}$, $\mathcal{P} = \mathcal{C}_1 \cup \mathcal{C}_2 = \{v_1, v_2\} \cup \{v_8, v_9, v_{10}\}$ and $\mathcal{T} = \mathcal{T}_1 \cup \mathcal{T}_2 = \{v_3, v_4, v_5, v_6, v_7\} \cup \{v_{11}, v_{12}, v_{13}, v_{14}, v_{15}, v_{16}, v_{17}, v_{18}\}$. As an example of a transient, we can consider the green nodes. Node v_{18} is contained in transient $\{v_{18}, v_{12}\}$ of length 2.

Clearly, the set $\mathcal{P}$ of all the periodic points can be viewed as union of disjoint cycles. Moreover, both $(\mathcal{C}, f|_{\mathcal{C}})$ and $(\mathcal{P}, f|_{\mathcal{P}})$ are dynamical subsystems of S and their dynamics graphs just consist of one among, resp., all, the strongly connected components[1] of G_S.

Given a cycle $\mathcal{C}$ and a $v \in \mathcal{X}$, if h is the smallest natural number such that $f^h(v) \in \mathcal{C}$, we call the set $\{v, f(v), \ldots, f^{h-1}(v)\}$ a **transient of length** h and its points **transient states**. The orbit of any point thus consists of at most two disjoint parts: its transient and a cycle. Clearly, if we denote by $\mathcal{P}$ the set of periodic states and $\mathcal{T}$ the set of transient states of S, we have $\mathcal{T} \cup \mathcal{P} = \mathcal{X}$. Figure 1.1 provides an example of the concepts introduced so far.

The G_S graph of a DDS S may contain one to several *weakly connected components* (connected components ignoring the orientation of the arcs [Pemmaraju et al. (2003), Hong et al. (2013)]) but each of them will always be characterized by only one cycle. This is the typical form of dynamic graphs in our context. Then, if we consider S with L cycles, the dynamics graph has L weakly connected components. Each component $\left(\mathcal{X}_j, f|_{\mathcal{X}_j}\right)$ with $j \in \{1, \ldots, L\}$ has one cycle $\mathcal{C}_j$ and $\mathcal{X}_j = \mathcal{T}_j \cup \mathcal{C}_j$. In the following for simplicity, we will use the term weakly connected components and connected components as synonyms. The transient points $\mathcal{T}_j$ of a component j are the nodes $v \in \mathcal{T}$ such that $f^h(v) \in \mathcal{C}_j$ for some $h > 0$. In the following, with an abuse of notation, we will refer to each component by its set $\mathcal{X}_j$.

1.1.2 Isomorphism of DDS

Given the fact that, for this work, we are not interested in the precise nature of the different states that characterise a given DDS, we can consider systems up to an isomorphism. Indeed, two systems are isomorphic if and only if there is an isomorphism between their dynamics graphs. From a graph theory point of view, an isomorphism of two graphs G and H (two dynamics graphs in our case) is a bijection between the vertex sets of G and H such that any two vertices v_1 and v_2 are adjacent in G if and only if $\gamma(v_1)$ and $\gamma(v_2)$ are adjacent in H. When this happens, the systems are indistinguishable from the dynamical point of view. In particular, periodic points and cycles

[1] A strongly connected component is the portion of a directed graph in which there is a path from each vertex to another vertex.

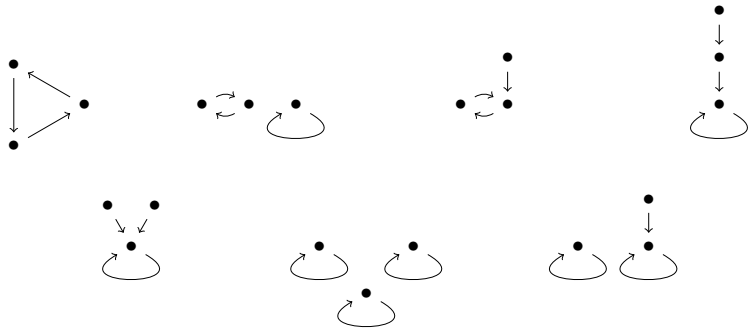


Figure 1.2 – All non-isomorphic DDS with three states.

of a system are in one-to-one correspondence with periodic points and cycles of the other system, and the dynamical subsystems induced by them in the respective DDS are isomorphic too.

Formally, a DDS $S_1 = (\mathcal{X}_1, f_1)$ is isomorphic to $S_2 = (\mathcal{X}_2, f_2)$ if and only if there exists a bijection γ such that $f_2 \circ \gamma = \gamma \circ f_1$.

Finite dynamical systems form a category $\mathcal{D}$ [Lawvere and Schanuel (1997), Article III]. Let us recall that, a category is a collection of objects, here DDS, that are linked by arrows between isomorphic systems. A category has two important properties: the ability to compose the arrows associatively and the existence of an identity arrow for each object. This category has as its initial object[2] the empty dynamical system, and as terminal objects[3] any single-state dynamical system with the identity function as next-state map.

The relation of isomorphism between DDS is an equivalence relation. It is easy to see that, if two systems are isomorphic, then the graphs associated with their dynamics are isomorphic too. The quotient set $\mathcal{D}$ is derived from the collection of DDS by the equivalence relation. Remark that this set is countable, and that considering dynamical systems up to isomorphism, we are defining equivalence classes between DDS. Given a fixed number of states of a DDS, the number of non-isomorphic DDS is given by the sequence A001372 of OEIS [The Online Encyclopedia of Integer Sequences (1996)a]. Figure 1.2 shows an example for DDS with three states.

1.1.3 Algebraic operations

In order to manipulate and combine discrete dynamical systems, two different operations have been defined: the sum and the product. Indeed, one can imagine that a certain complex dynamics is the result of two independent sub-dynamics, as well as the result of collaboration, or interaction, between different subsystems. The definition of operations thus serves precisely to be able to study different types of interaction between systems and to study how complex dynamics can be broken down or factorised into something smaller and therefore also easier to study. In fact, as we are going to see, equations on DDS are introduced for this purpose. However, in order to be able to investigate the solutions to these equations (*i.e.* different factorisations and decompositions), one must know how the operations involved in these work. For more about the following concepts, the reader can refer to [Dennunzio et al. (2018)].

[2] The initial object is an object of the collection such that there exists precisely one morphism to any other object.

[3] The terminal object is an object of the collection such that, from every object in the collection, there exists precisely one morphism to itself.

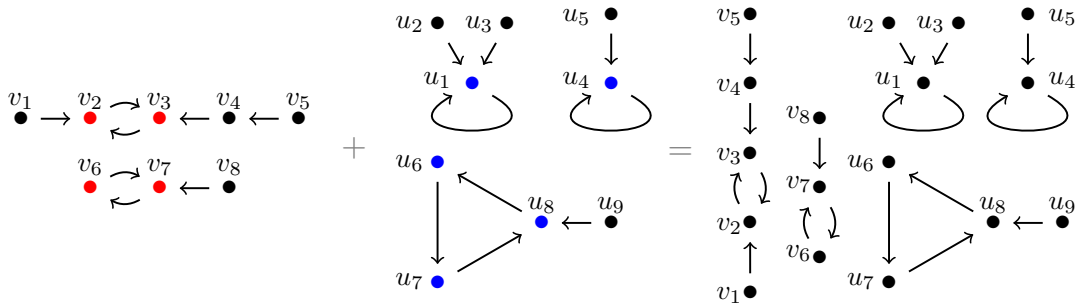


Figure 1.3 – Example of sum of two DDS. The resulting DDS contains all the original components of the two systems. Remark that name of the states in the result are simplified in v_i for all $(v_i, 0)$ with $i \in \{1, \dots, 8\}$ and in u_i for all $(u_i, 1)$ with $i \in \{1, \dots, 9\}$.

1.1.3.1 Sum of discrete dynamical systems

Before introducing the sum of two DDS, let us recall that the *disjoint union* $\mathcal{X}_1 \sqcup \mathcal{X}_2$ between two sets $\mathcal{X}_1$ and $\mathcal{X}_2$ is defined as $(\mathcal{X}_1 \times \{0\}) \cup (\mathcal{X}_2 \times \{1\})$. Intuitively, in this type of union all elements are combined but the information about the original sets is not lost.

Definition 1.1.2 (Sum of DDS)**.** The sum $(\mathcal{X}_1, f_1) + (\mathcal{X}_2, f_2)$ of any two DDS $S_1 = (\mathcal{X}_1, f_1)$ and $S_2 = (\mathcal{X}_2, f_2)$ is the DDS $(\mathcal{X}_1 \sqcup \mathcal{X}_2, f_1 \sqcup f_2)$ where the function $f_1 \sqcup f_2 : \mathcal{X}_1 \sqcup \mathcal{X}_2 \to \mathcal{X}_1 \sqcup \mathcal{X}_2$ is defined as:

$$\forall (v, i) \in \mathcal{X}_1 \sqcup \mathcal{X}_2, \quad (f_1 \sqcup f_2)(v, i) = \begin{cases} (f_1(v), i) & \text{if } v \in \mathcal{X}_1 \text{ and } i = 0, \\ (f_2(v), i) & \text{if } v \in \mathcal{X}_2 \text{ and } i = 1. \end{cases}$$

Since we will use the equivalence between DDS and dynamics graphs, let us highlight that the sum as defined above corresponds to the disjoint union of the components of the graphs. In fact, the result of the sum operation between two systems is a new one containing both dynamics (see Figure 1.3). In other words, the resulting system presents the two alternative behaviours of the original systems, and according to the initial state, one of the two original dynamics will follow.

The empty dynamical system is the identity element, or neutral element, for this operation. Remark that the sum of DDS is commutative and associative (up to isomorphism since the disjoint union names the nodes of the result differently).

In the sequel, for any natural $k > 0$ and any DDS S, the sum $S + \ldots + S = \sum_{i=1}^{k} S$ of k copies of S will be denoted by $k\,S$.

1.1.3.2 Product of discrete dynamical systems

The product between DDS is the **direct product** of dynamics graphs (sometimes also called *tensor product* of graphs). In the graph community, this product of graphs is denoted with $\times$. Let us clarify that, here, $\times$ is used to denote the Cartesian product of sets and $\cdot$ to denote the product of DDS or the integer product. Chapter 2 contains more details about this specific graph product and its known properties according to the graph literature. In addition, let us point out that if one considers the adjacency matrices of the dynamics graphs, one can also refer to the *Kronecker product*.

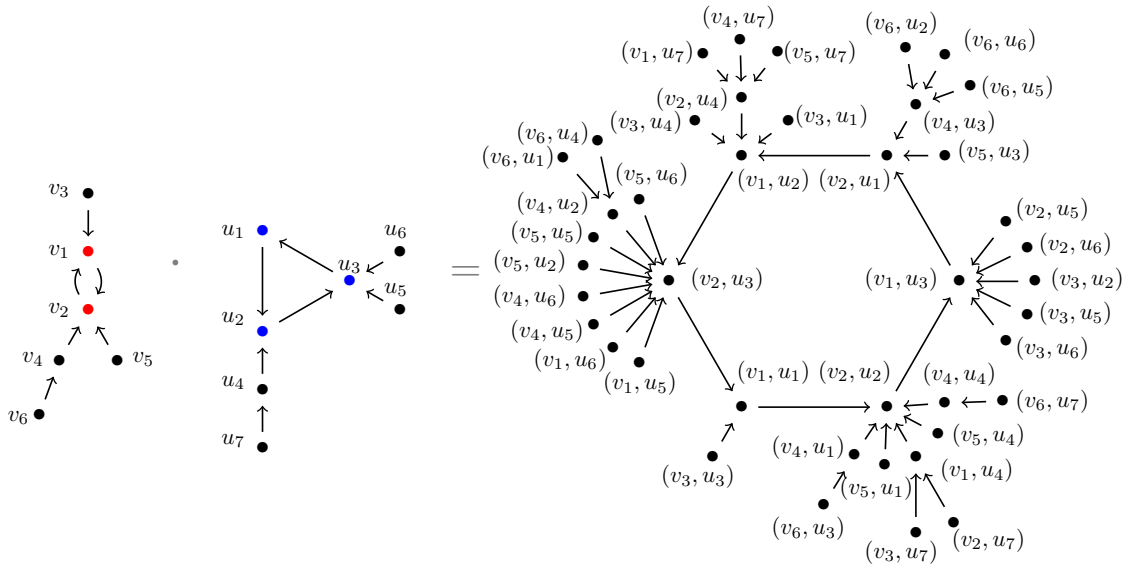


Figure 1.4 – An example of product operation between two DDS with just one component each. The result of the product operation contains just one component too because the cycle lengths are coprime.

Definition 1.1.3 (Product of DDS)**.** The product $(\mathcal{X}_1, f_1)\cdot(\mathcal{X}_2, f_2)$ of any two DDS $S_1 = (\mathcal{X}_1, f_1)$ and $S_2 = (\mathcal{X}_2, f_2)$ is the DDS $(\mathcal{X}_1 \times \mathcal{X}_2, f_1 \times f_2)$ where the function $f_1 \times f_2 : \mathcal{X}_1 \times \mathcal{X}_2 \to \mathcal{X}_1 \times \mathcal{X}_2$ is the standard product of functions defined as $\forall (v_1, v_2) \in \mathcal{X}_1 \times \mathcal{X}_2, (f_1 \times f_2)(v_1, v_2) = (f_1(v_1), f_2(v_2))$.

Let us precise that multiplying any dynamical system by the empty system gives the empty system (*i.e.* the annihilation law holds), and that multiplying a dynamical system with a simple fixed point produces a system isomorphic to the original one. In all other cases, the product of DDS consists of a Cartesian product of the original sets of states and in a new next-state map.

As will be detailed in the next chapter, the number of components resulting from a product operation depends on whether the cycle lengths involved are coprime or not [Hammack et al. (2011)] [Dennunzio et al. (2020)]. This will be explored in more detail later on, but Figures 1.4 and 1.5 already show an example where only one component is obtained and another one where several components are obtained.

The product operation of DDS is associative and commutative modulo isomorphism of dynamical systems. In fact, this operation needs a Cartesian product, as a consequence changing the order of the operators changes the resulting set of states. However, the evolution of the resulting systems are isomorphic (see Figure 1.6).

In the sequel, for any natural $w > 0$ and any DDS S, the product $S \cdot \ldots \cdot S = \prod_{i=1}^{w} S$ of w copies of S will be naturally denoted by S^w. In this way, one can state the following proposition which is nothing but the counterpart in this setting of the well-known standard multinomial theorem.

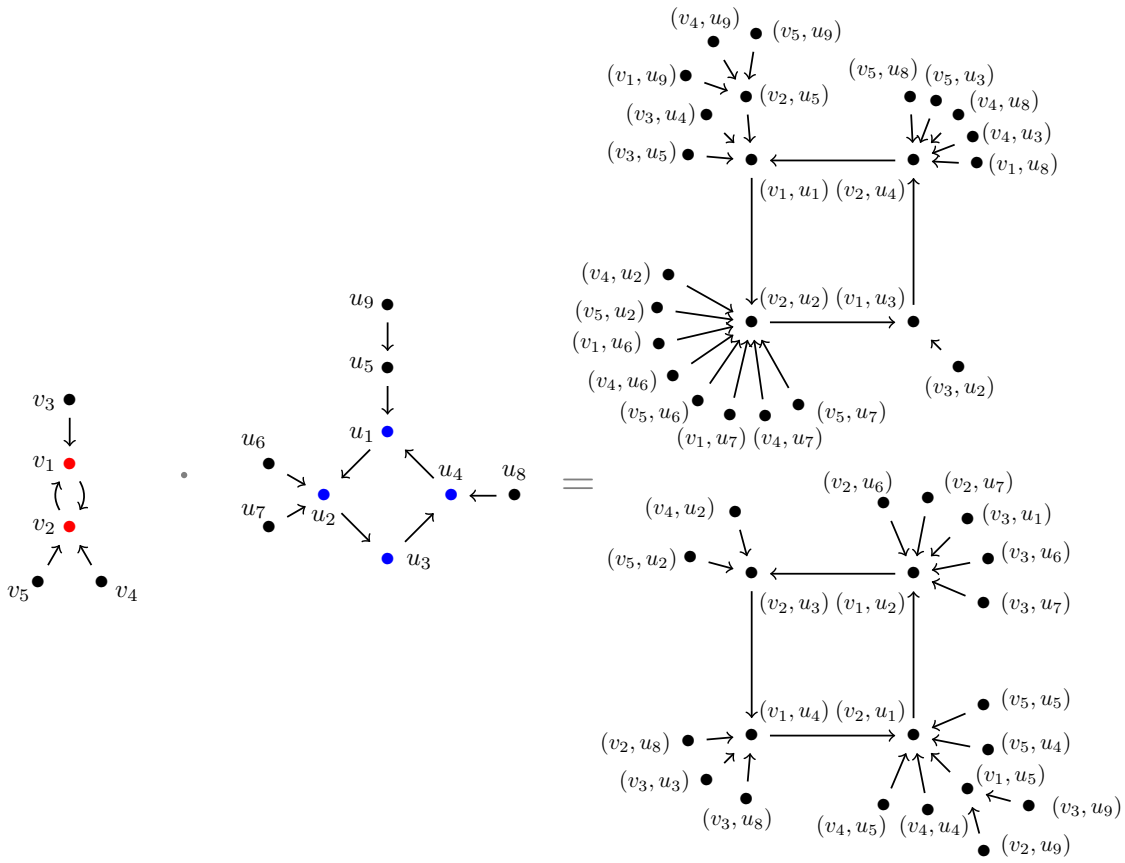


Figure 1.5 – An example of product operation between two DDS with just one component each. The result of the product operation contains different components because the cycle lengths are not coprime.

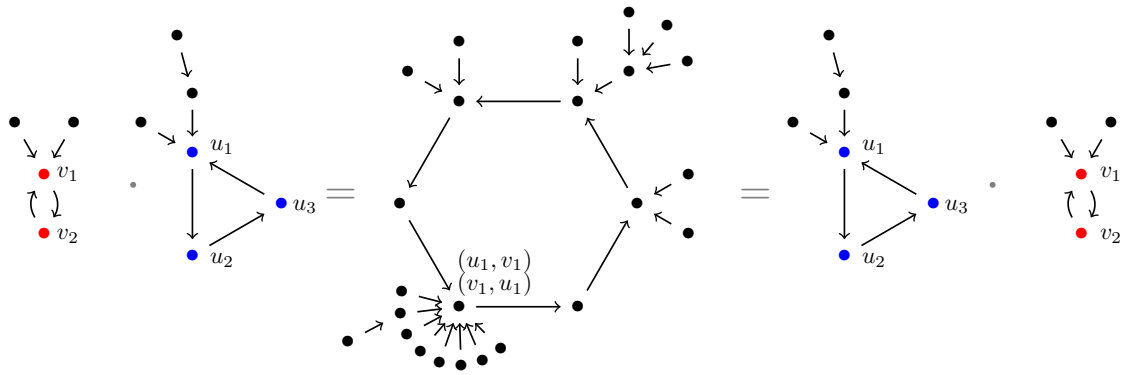


Figure 1.6 – The example shows the product of two DDS performed by changing the order of the operands. Only some names of the states are shown to make the picture clearer. One can see how the product in both cases leads to isomorphic dynamics, but the same state has different names.

Proposition 1.1.1. *For any positive naturals w and z, and any DDS $S_1, \ldots, S_z$, it holds that*

$$(S_1 + \ldots + S_z)^w = \sum_{\substack{k_1+\ldots+k_z=w \\ 0\leq k_1,\ldots,k_z\leq w}} \binom{w}{k_1, \ldots, k_z} \prod_{t=1}^{z} (S_t)^{k_t} \quad .$$

To better understand the product of DDS, it is useful to refer to the Cayley table (part of it is shown in Figure 1.7). It shows the result of different products among DDS by increasing the number of nodes and evaluating the different functions that can be described within them.

Thanks to the Cayley table one can notice an important property. The elements of $\mathcal{D}$ do not exhibit unique factorisation. Figure 1.8 highlights an example. At the same time, however, we know that there are systems that we can define as *irreducible*, meaning that there is no nontrivial factorisation for them [Dennunzio et al. (2018)]. In other words, one can find them in the Cayley table only as the result of a product involving a fixed point (product identity). Note that, for example, a DDS containing only one cycle of length two is irreducible. Intuitively, these systems have a prime number of states (for example two) and to produce a set of states of prime cardinality, one needs a set with a prime number of elements and another containing only one element. Hence, any system with a prime number of states is irreducible. In addition, it has been shown that given any integer number k, there is always at least one system admitting at least k different factorisations in irreducible systems (see Figure 1.9). Let us point out that factorisation is a tricky topic on this type of structure and product. Indeed, there are still several open questions [2].

1.2 The Semiring of Dynamical Systems

From the previous section, we know that $(\mathcal{D}, +)$ is a commutative monoid with the empty dynamical system as the neutral element, and $(\mathcal{D}, \cdot)$ is a commutative monoid with a single fixed point system as the neutral element. Moreover, products distribute over sums, *i.e.* $A \cdot (B+C) = A \cdot B + A \cdot C$.

It has been proved that the set $\mathcal{D}$ equipped with these operations of sum and product is a commutative semiring [Dennunzio et al. (2018), Hebisch and Weinert (1998)]. A semiring is similar to a ring, but without the requirement that each element must have an additive inverse. Indeed, the inverse of the sum is not defined in our semiring, because adding any two dynamical systems can only increase or leave unchanged the total number of states. For this reason, it is not possible to define the operation of subtraction.

More formally, a semiring is a set together with two binary operators $(+, \cdot)$ satisfying the following conditions:

- both operations are associative, then $\forall A, B, C \in \mathcal{D}$:

$$(A + B) + C = A + (B + C)$$

$$(A \cdot B) \cdot C = A \cdot (B \cdot C)$$

- the sum is commutative, that is to say $\forall A, B \in \mathcal{D}$:

$$A + B = B + A$$

- left and right distributive, that is to say $\forall A, B, C \in \mathcal{D}$:

$$A \cdot (B + C) = (A \cdot B) + (A \cdot C)$$
$$(B + C) \cdot A = (B \cdot A) + (C \cdot A)$$

Finally, $\mathcal{D}$ is a commutative semiring since the product is commutative too. In [Dennunzio et al. (2018)], another interesting property of this semiring has been shown.

Proposition 1.2.1. *The semiring $\mathcal{D}$ contains a subsemiring $\mathcal{N}$ isomorphic to the natural numbers.*

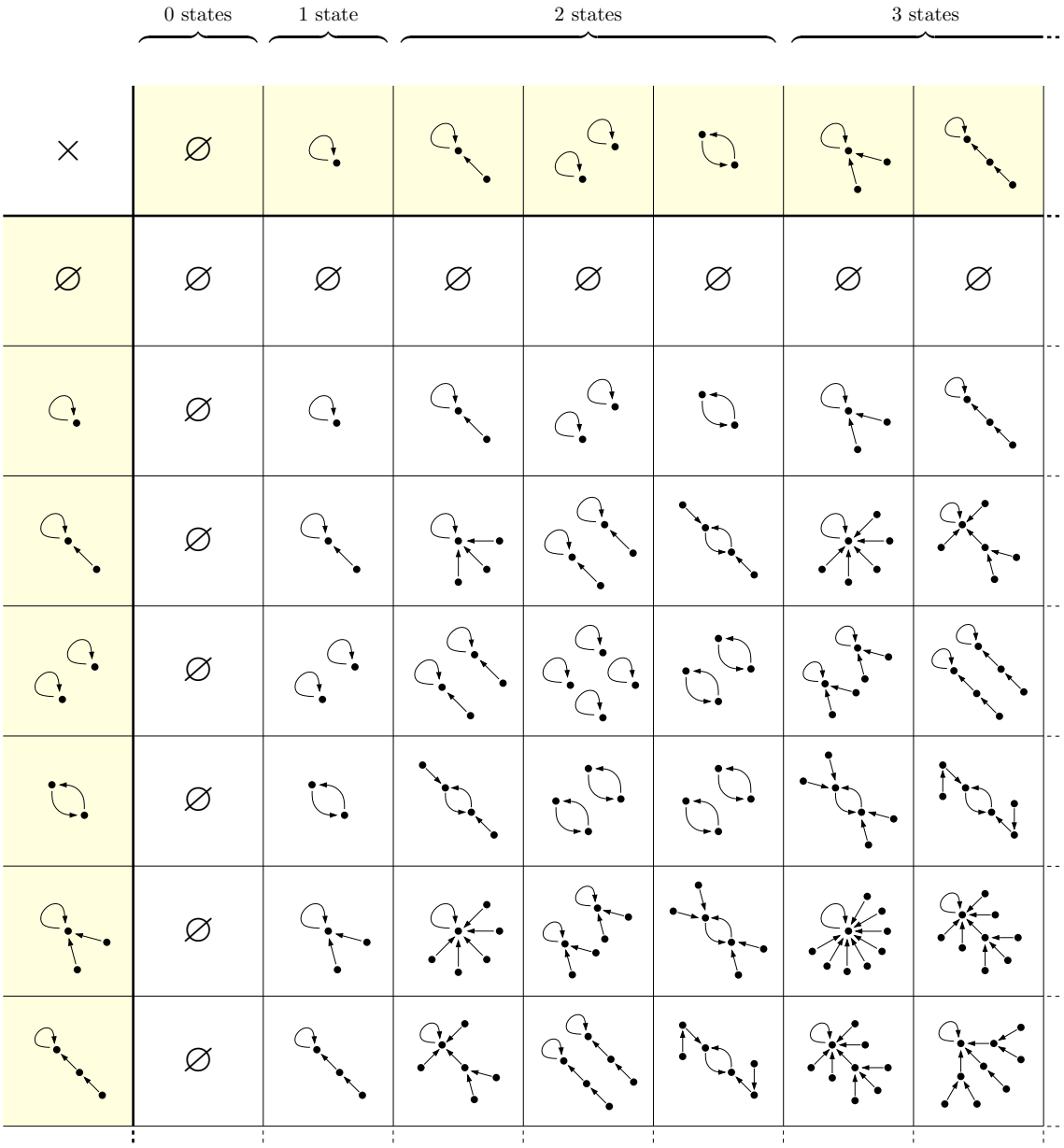


Figure 1.7 – A portion of the Cayley table of the product of DDS, including products of all non-isomorphic DDS in increasing order of size [Dennunzio et al. (2018)].

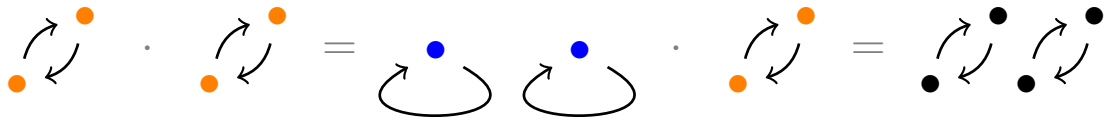


Figure 1.8 – The dynamics graph consisting of only two cycles of length two is an example of dynamics with non-unique factorisation (in black above). Indeed, it can be obtained by multiplying two graphs each containing a cycle of length two, or by multiplying a cycle of length two with a graph containing two fixed points. A DDS containing a cycle of length 2 is colored in orange, and a DDS with two fixed points is in blue.

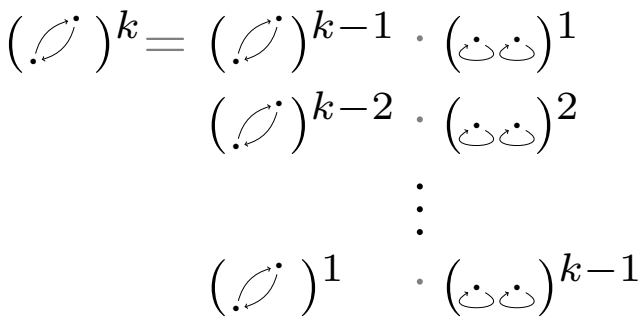


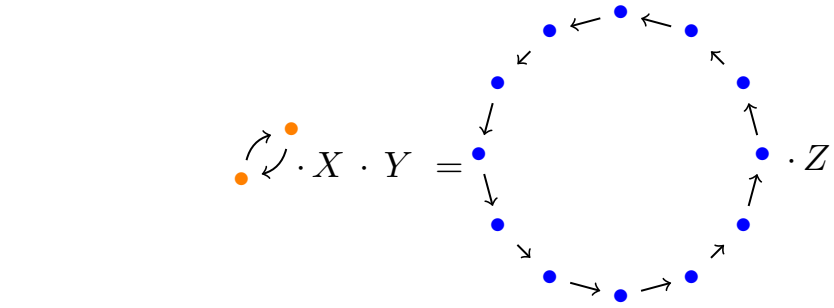
Figure 1.9 – An example of system admitting at least k distinct factorisations in irreducible systems for any $k \in \mathbb{N}$ [2].

Recall that the semiring of the natural numbers is initial in the category of commutative semirings. This means that for each semiring there is just a homomorphism of naturals to the semiring. Moreover, in this particular case it will be a monomorphism since it is injective too. This last proposition is important when the objective is to solve equations over $\mathcal{D}$.

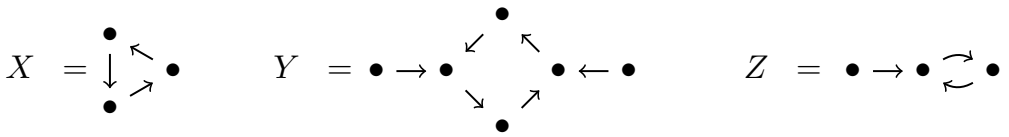
Remark that $\mathcal{D}$ is also a $\mathbb{N}$-semimodule. Indeed, it is easy to prove the semimodule axioms since $\mathbb{N}$ is a sub-semiring of $\mathcal{D}$. Moreover, it has a unique countably infinite base, the set of all connected non-empty DDS. In fact, recall that a generic DDS can been seen as a multiset of connected non-empty smaller DDS. This is also another important aspect when we want to solve equations over $\mathcal{D}$.

Having introduced the semiring structure on DDS, we can now study various problems through polynomial equations. Note that polynomials over a commutative semiring are themselves commutative semirings [Golan (2013), Chapter 1].

For example, one can study different ways to express a certain parametric behaviour (see Figure 1.10), or analyse a given dynamical system in terms of smaller (or simpler) components. In Chapter 4, polynomial equations with a constant right-hand side (able to model this last problem) will be presented in greater details as they will be the main object of study in the following chapters.



(1.1)



(1.2)

Figure 1.10 – Equation 1.1 above allows us to express the parametric behaviour on the right as different products of three components. A solution is given in 1.2.

1.2.1 Polynomial equations and Complexity

Since $\mathcal{D}$ is a commutative semiring, it is natural to define polynomials over it. In a more general context, a polynomial is an expression consisting of variables (also called *indeterminate*) and coefficients, that involve only the operations of addition, subtraction, multiplication. A polynomial over $\mathcal{D}$ has variables and coefficients in $\mathcal{D}$ but involves only additions and multiplications. The set $\mathcal{D}[X]$ is obtained adding to $\mathcal{D}$ a new element $X \in \mathcal{D}$ and closing the new set under sum and multiplication. In other words, $\mathcal{D}[X]$ denotes the semiring of polynomials over the indeterminate X.

In order to focus on the complexity results, let us distinguish the cases of polynomial equations with and without a constant term. Recall that due to the lack of the subtraction operation, we cannot move the terms between the two sides of an equation (as in equations over rings). Then, the generic form of a polynomial equation over $\mathcal{D}$ is $P(X_1, \ldots, X_\nu) = Q(X_1, \ldots, X_\nu)$ where $P(X_1, \ldots, X_\nu)$ and $Q(X_1, \ldots, X_\nu)$ are two polynomials over $\mathcal{D}$ in the variables $X_1, \ldots, X_\nu$.

1.2.1.1 Generic polynomial equations: a fundamental theorem

Unfortunately, finding solutions to polynomial equations over $\mathcal{D}$ is, in general, a hard task. Indeed, the following theorem proves the undecidability of establishing if a generic polynomial equation over $\mathcal{D}$ has a solution.

Theorem 1.2.2 ([Dennunzio et al. (2018)]). *Given two polynomials $P(X_1, \ldots, X_\nu)$ and $Q(X_1, \ldots, X_\nu)$ in $\mathcal{D}[X_1, \ldots, X_\nu]$, consider the following equation:*

$$P(X_1, \ldots, X_\nu) = Q(X_1, \ldots, X_\nu). \tag{1.3}$$

The problem of finding a solution to Equation 1.3 is undecidable.

It has been shown that the algorithmic solution of polynomial equations over $\mathcal{D}$ turns out to be algorithmically impossible by reduction from Hilbert's tenth problem [Matiyasevich (1993)]. Remark that, by implication, also finding a solution to such equations, when it exists, is undecidable. In brief, this result comes from a connection between equations over DDS and diophantine equations. A diophantine equation is a polynomial equation in which only integer coefficients and integer solutions are allowed. The diophantine equation is introduced by considering the number of states in the DDS involved in the equation on $\mathcal{D}$. Hilbert's 10th problem asked if an algorithm existed for determining whether an arbitrary Diophantine equation has a solution, but the impossibility of obtaining a general solution was proven by Yuri Matiyasevich in 1970 [Matijaszevic (1970)]. It has been proved that the problem of knowing if a diophantine equation has a natural number solution is undecidable by reduction from the 10th Hilbert's problem [Matiyasevich (1996)].

Moreover, if Equation 1.3 is linear or quadratic, it has been proved that the problem of finding a solution is in NP. The idea is that by limiting the maximum degree of the equation, simpler diophantine equations, e.g. linear, are considered. It is known that, by starting from the solutions of linear diophantine equations (which can be calculated in polynomial time), one can guess the solutions between the DDS of the dimension found and then check whether they are real solutions of the equation.

1.2.1.2 Polynomial equations with constant term

Concerning polynomial equations with constant term over DDS, let us first consider the univariate case. Given an equation $P(X) = B$, where $P(X) \in \mathcal{D}[X]$ and B is the empty dynamical system, it is trivial since it has solution X, the empty DDS, just if there is no constant term in the polynomial part. For single variable equations with all the coefficients and the constant side consisting of self-loops, the equation is solvable in polynomial time [Dorigatti (2018)]. Then, in the case of $P(X) = B$ with a generic DDS $B \in \mathcal{D}$, intuitively, one is able to perform a search between all DDS with at most the same amount of states of B. In general, when solving equations over DDS, it is important to point out that the difficulty comes from finding the "correct" systems for the different variables and not from verifying the isomorphism between the two sides of an equation (since this goes back to the problem of isomorphism between planar graphs) [Hopcroft and Wong (1974), Datta et al. (2009)]. This idea holds also in the case of multivariate polynomial equations.

Theorem 1.2.3 ([Dennunzio et al. (2018)]). *The problem of finding solutions of polynomial equations over $\mathcal{D}$ with a constant side is in NP.*

If we consider systems of linear equations with a constant side, it has been shown to be an NP-complete problem by reduction to one-in-three-3SAT problem. This last problem consists of deciding whether, given a 3CNF Boolean formula, there is an assignment such that there is only one literal per clause that is true.

Through this first result, it was then shown that a linear equation with a constant side is also NP-complete. As a consequence of the above, it is known that also multivariate equation with bounded degree (greater than or equal to 1) are NP-complete [1].

1.3 Examples of Dynamical Systems

Having presented DDS, their operations, the commutative semiring $\mathcal{D}$ and some complexity results, we now want to introduce two examples that allow us to understand how through a simple model, such as DDS, we can study complex dynamics arising from different domains. This section aims just to show how the dynamics that can be obtained from Boolean Networks and Cellular Automata are the dynamic systems of interest to this study. For more complete information about these models one can refer to, for instance, [Sené (2012), Paulevé (2020), Kari (2005), Fatès (2014)] .

1.3.1 Boolean Networks

A Boolean Network (BN) of size n is an arrangement of n finite Boolean automata (or components) interacting with each other according to a *global rule* $F\colon \{0,1\}^n \to \{0,1\}^n$, which describes how the global state changes after one time step. Let $[\![n]\!] = \{0,\dots,n-1\}$. Each automaton is identified with a unique integer $i \in [\![n]\!]$ and x_i denotes the current state of the automaton. A *configuration* $x \in \{0,1\}^n$ is a snapshot of the current state of all automata and represents the global state of the BN. For convenience, configurations are identified with words on $\{0,1\}^n$. Remark that the global function $F : \{0,1\}^n \to \{0,1\}^n$ of a BN of size n induces a set of n local functions $f_i : \{0,1\}^n \to \{0,1\}$, one per each component, such that $F(x) = (f_0(x), f_1(x), \dots, f_{n-1}(x))$ for all $x \in \{0,1\}^n$.

Boolean networks are applied in various domains but a common application of this model is in the biological field to describe the behaviour of genes, proteins, etc. Obviously, depending on the application scenario, the interpretation of the states of the different components differs (see Example 1.3.1). Local functions model the activation conditions of each component according to the state of the network.

To describe a network, one usually starts by giving its architecture. To do so, it is possible to represent the interactions between automata with a directed graph (called *interaction graph* $G_F = (V, A)$) whose vertices represent the set of automata and the arcs model the interactions between them in the network. In this graph, an edge indicates that an automaton can influence another one.

This gives a static description of a DDS, and it remains to set the order in which components are updated to get a dynamics. This may be done synchronously or asynchronously. Remark that no matter how the components are updated, a BN of size n will always have 2^n possible states.

The simplest way to update the components is the *synchronous* one in which all of them change state in the same moment in time [Kauffman (1969)]. This corresponds to a deterministic dynamics in which, for each configuration, there is just one possible configuration that will follow. In other words, the resulting dynamics is a DDS as presented before in the chapter.

Another way to update the components is the *asynchronous* one, but in this case one must define how the asynchronism is introduced. The *fully asynchronous* update procedure is based on the idea that at each moment in time only one random component is updated [Thomas (1973)]. Then, from each state of the dynamics graph, one can reach up to n possible states. Asynchronous updating is certainly more realistic, however it can lead to timings distant from the real ones observed in the biological field. Moreover, it has been showed that synchronous updating may be more relevant for evaluating some properties of the network [Schwab et al. (2020)a].

Generalised asynchronous updating allows all the combinations of simultaneous updates subsets of nodes, from single nodes to the synchronous scenario. Other updating modes like sequential or block-sequential have also been considered (see Example 1.3.2). Remark that these last updating procedures are just some of the generalised asynchronous ones.

Example 1.3.1 – Let us consider three genes g_0, g_1 and g_2. A Boolean network on size $n = 3$ can model the interactions between them. The state $x_i = 1$ models that a gene g_i is active and the state 0 models the inactivity (with $i \in \{0, \dots, 2\}$). One can suppose that $f_0(x) = x_1$, $f_1(x) = x_0 \wedge x_2$ and $f_2(x) = \neg x_0$.

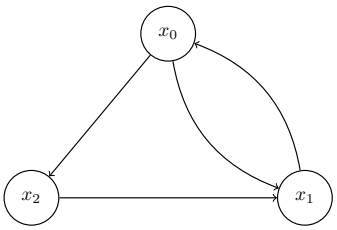


Figure 1.11 – The interaction graph of the BN.

The dynamics graph, based on the synchronous update procedure, have 2^3 states and edges (see Figure 1.12). For example, considering the initial state $x = 100$, the following state will be $F(x) = (f_0(100), f_1(100), f_2(100)) = (000)$ and the orbit will be $\{100, 000, 001\}$. Remark that the result of the update procedure over $x = 001$ lead to identify 001 as a fixed point since $F(x) = (f_0(001), f_1(001), f_2(001)) = (001)$. According to this reasoning, it is possible to identify the following dynamics graph.

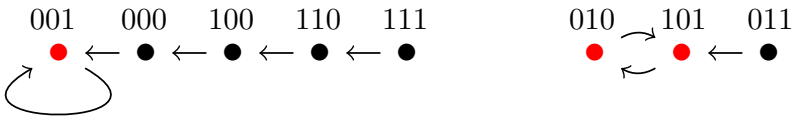


Figure 1.12 – The synchronous dynamics graph of the BN.

The dynamics graph differs if one consider, for example, the fully asynchronous update procedure (see Figure 1.13). In this case, considering the initial state $x = 100$, the following state will be $F(x) = (f_0(100), 0, 0) = (000)$, if the first automaton is updated, $F(x) = (100)$ if one of the others is updated. Remark that $F(x) = (1, f_1(100), 0) = (100)$ and $F(x) = (1, 0, f_2(100)) = (000)$.

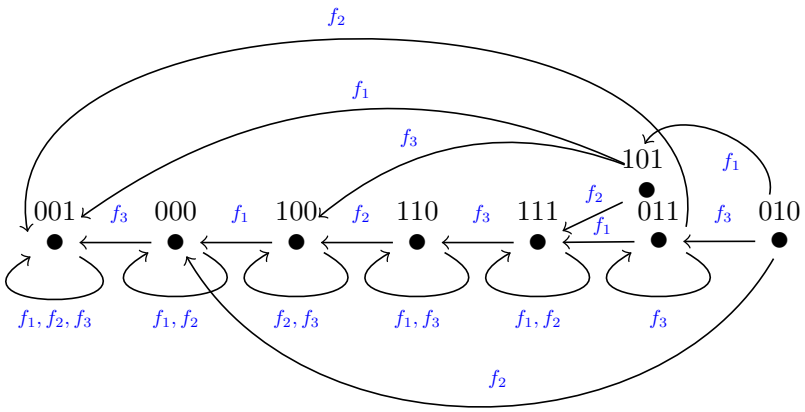


Figure 1.13 – The fully asynchronous dynamics graph of the BN. The resulting dynamics graph is not deterministic. The labels on the arcs therefore indicate which component is updated. Thus, f_i indicates that component i is updated. Remark that it is possible that updating different cells brings at the same state.

1.3.2 Cellular Automata

Cellular automata (CA) are a discrete model invented in 1940 by Stanislaw Ulam and John von Neumann. They are based on the idea of describing complex evolution by means of simple local rules. Formally, Cellular automata are dynamical systems with discrete time and state variables. They are interesting mathematical and computational objects suitable for modeling real-world complex systems. Despite their apparent simplicity, CA display non-trivial global emergent behavior, and some of them can even reach computational universality [Cook (2004), Gardner (1970)].

They are defined over a regular grid of cells of fixed dimension. For each of these dimensions the number of *cells* can be finite or infinte. The dynamics of a CA is locally-defined: every cell computes its next state (which belongs to a finite set of possible values) based upon its own state and neighbors states (according to a specific definition of neighborhood). The snapshot of the current states of cells is called configuration, as in the Boolean networks case. Here, there is a fixed rule that decides how each cell is updated to a new state according to the values of the neighborhood. Typically, this rule does not change over time.

For the sake of simplicity, CA will be presented here in the more general framework of Boolean automata networks (with the notations presented before), but this is not always the case in the literature. Moreover, here, we will consider CA with a finite number of cells.

Elementary cellular automata (ECA) are a subclass of BN in which, given a size n, the local function $r : \{0,1\}^3 \to \{0,1\}$ is the BN F such that

$$\forall i \in [\![n]\!], \quad f_i(x) = r(x_{i-1}, x_i, x_{i+1})$$

where components are taken modulo n. Then, each ECA can be identified according to the convention introduced by Wolfram to designate each of the 256 ECA local rule $r : \{0,1\}^3 \to \{0,1\}$

as the number

$$w(r) = \sum_{(x_1,x_2,x_3)\in\{0,1\}^3} r(x_1,x_2,x_3)\cdot 2^{(2^2x_1+2^1x_2+2^0x_3)}.$$

The symbol r_α denotes the Boolean function such that $w(r)=\alpha$ with $\alpha\in\{0,\dots,255\}$.

Given $r:\{0,1\}^3\to\{0,1\}$, one can consider the following transformations over local rules: $\tau_\iota(r)(x,y,z)=r(x,y,z)$, $\tau_\varrho(r)(x,y,z)=r(z,y,x)$, $\tau_\eta(r)(x,y,z)=1-r(1-z,1-y,1-x)$ and $\tau_{\varrho\eta}(r)(x,y,z)=1-r(1-z,1-y,1-x)$ for all $x,y,z\in\{0,1\}$. It is known that these transformations preserve the dynamics [Cattaneo et al. (1997)]. For this reason, one can consider only 88 rules up to equivalences with τ_ι, τ_ϱ, τ_η and $\tau_{\varrho\eta}$ to study the different dynamics of ECA.

Also considering CA, one can use the interaction graph to represent the interactions between cells. The graph depends from the definition of neighborhood. However, only the effective dependencies among the set of cells need to be represented. The cell i *influences* the cell j if $\exists x\in\{0,1\}^n$ such that the result of the update procedure of j is different according to whether the state of component i is 0 or 1. According to this idea, the *interaction digraph* $G_F=(V,A)$ of the function F is based on

$$V=[\![n]\!] \quad\text{and}\quad A=\{(i,j)\mid i \text{ influences } j\}.$$

Cellular automata are liable to different update procedures as BN. The different update procedures already presented apply also to CA. However, over the last decade, asynchronous CA have attracted increasing attention in the scientific community. In order to ensure compliance with the definition of DDS provided here, deterministic asynchronous update procedures must be considered. As, an example one can cite the deterministic *block-sequential* update procedure [Aracena et al. (2009), Aracena et al. (2011)]. In this case, the cells of a CA are partitioned into disjoint blocks with different priorities of being updated. The initial updating order (priority scheme) is unchanged throughout the temporal evolution. A possible cells partition is called a block-sequential update schedule.

More formally, for $n\in\mathbb{N}$, one can denote by P_n the set of ordered partitions of $[\![n]\!]$ and by F a BN of size n. A block-sequential update schedule $\Delta=(\Delta_1,\dots,\Delta_k)$ is an element of P_n. It defines the following dynamics $F^{(\Delta)}:\{0,1\}^n\to\{0,1\}^n$,

$$F^{(\Delta)}=F^{(\Delta_k)}\circ\cdots\circ F^{(\Delta_2)}\circ F^{(\Delta_1)} \quad\text{with}\quad f^{(\Delta_j)}(x)_i=\begin{cases} f_i(x) & \text{if } i\in\Delta_j,\\ x_i & \text{if } i\notin\Delta_j.\end{cases}$$

In other words, the components are updated in the order given by Δ: sequentially block by block, and in parallel within each block. Remark that the synchronous update schedule is $\Delta^{\mathtt{sync}}=([\![n]\!])$. This update procedure is defined as *fair*, in the sense that all components are updated exactly the same number of times, and *periodic* in the sense that the same ordered partition is repeated (see Example 1.3.2).

In the case of block-sequential update schedules, one can observe that the dynamics according to a certain Δ can differ from the one obtained according to another Δ' because they can introduce different priorities between cells. A natural question is to understand the relationship between the number of different dynamics one can obtain and the number of different block-sequential update schedules. This require to be able to count the number of *valid* non-equivalent update schedules (since some of them can introduce invalid update priorities) and also to detect which Δ,Δ' lead to the same dynamics for a certain ECA. This aspect is analysed in Appendix A.

Example 1.3.2 – Consider the ECA 146 and the visual representation of the rule provided in Figure 1.14. As with all ECA, we consider a three-cell neighbourhood. For each possible value of the latter, the new value of the cell is displayed below. Graphically, the black cells represent the value 1 and the white cells represent 0.

Figure 1.14 – Visual representation of the ECA 146.

Consider an ECA with 4 cells and a block-sequential update schedule $\Delta = (\{0,3\},\{2\},\{1\})$. When updating a cell in an ECA, the result depends on the state of the cell itself and that of its left and right neighbours (as shown by the interaction graph in Figure 1.15). The dynamics graph contains 2^4 states and for each one of them there is just one result of the update procedure (see Figure 1.16).

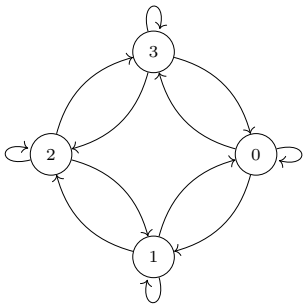


Figure 1.15 – The interaction graph of the ECA presented in Example 1.3.2.

In the literature, the dynamics graph is often called *transition digraph*. In Figure 1.16, one can see that the configuration 0011 is updated into 1100. In fact:

$$F(0011) = (r_{146}(100), r_{146}(r_{146}(100), 0, r_{146}(0, 1, r_{146}(110))), r_{146}(0, 1, r_{146}(110)), r_{146}(110))$$

$$= (1, r_{146}(1, 0, r_{146}(0, 1, 0)), r_{146}(0, 1, 0), 0) = (1, r_{146}(1, 0, 0), 0, 0) = 1100.$$

The updating order Δ is unchanged throughout the temporal evolution, then considering all possible configurations and the result of the update procedure over them, one can obtain the dynamics graph for a certain ECA according to a fixed Δ.

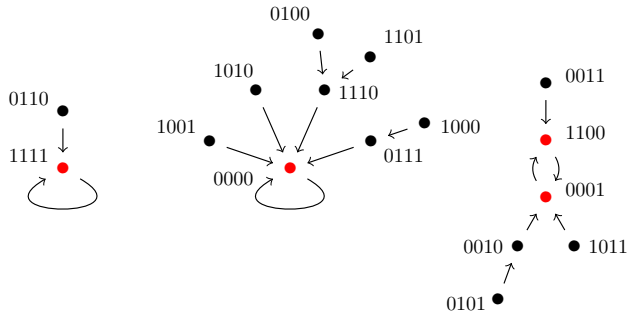


Figure 1.16 – The deterministic block-sequential dynamics graph of the ECA 146 with 4 cells according to $\Delta = (\{0, 3\}, \{2\}, \{1\})$.

CHAPTER 2

Direct Product and the Cancellation Problem

The Direct product is one of the several types of graph products that have been defined in the literature, and it corresponds to the product defined between DDS. Then, according to this product operation, this chapter introduces useful properties about the distance between vertices, connectivity, and prime factorisations (for both graphs and digraphs). Afterward, it introduces the problem of cancellation since it plays an important role to study the uniqueness of a solution of an equation over graphs. In fact, given a specific product of graphs $*$, the objective of this problem is to find under which conditions $G * H \cong G' * H$ implies $G \cong G'$. To achieve this goal, this chapter will present a set of conditions on H, G and G' such that the cancellation property holds.

2.1 The Direct Product

Graphs are objects that have been studied extensively in computer science, and various types of products have been defined on them. A generic product is a binary operation over graphs (that may or may not be oriented) based on a Cartesian product of the set of vertices and on specific adjacency conditions to define the arcs in the resulting graph. The main three products studied in the literature are: the *Cartesian product*, the *Direct product* and the *Strong product*.

Let $V(G)$ the set of vertices of a graph G and let $E(G)$ the set of edges (or arcs) of the same graph. The Cartesian product between two graphs G and H, usually denoted $G \square H$, defines two states (g, h) and (g', h'), with $g, g' \in G$ and $h, h' \in H$, adjacent[1] if $g = g'$ and $(h, h') \in E(H)$ or if $h = h'$ and $(g, g') \in E(G)$.

The Strong product between two graphs G and H, usually denoted $G \boxtimes H$, defines two states (g, h) and (g', h'), with $g, g' \in G$ and $h, h' \in H$, adjacent if $(g, g') \in E(G)$ and $(h, h') \in E(H)$, or $g = g'$ and $(h, h') \in E(H)$ (or the symmetric case). In other words, the edges of a Strong product $G \boxtimes H$ can be seen as the union of the edges of $G \square H$ and $G \cdot H$ (see Figure 2.1).

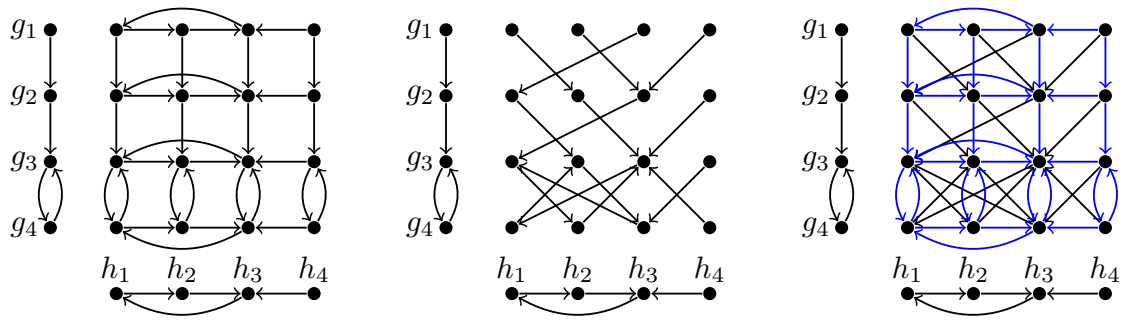


Figure 2.1 – Given two digraphs G and H, the Cartesian product (left), the Direct product (middle) and the Strong product (right) are computed. The digraphs G and H are placed on two different axes in such a way that the nodes of the resulting digraph are arranged in the form of the Cartesian product of the vertices of the original digraphs. It should be noted that the Strong product contains exactly the arcs of the Cartesian product (in blue) and those of the Direct product (in black).

The purpose of this first section is to introduce the direct product between graphs (and digraphs) and some of its properties. As previously specified, this is the product introduced for DDS and it is commutative and associative. Let us recall that, in the literature this product is also known as: *tensor product, Kronecker product, cardinal product*.

Given the objective of solving equations on dynamics graphs, it is important to study the uniqueness of the solutions. Given an equation $A \cdot X = B$ with $A, X, B \in \mathcal{D}$. The uniqueness of the value of the indeterminate X can be traced back to the cancellation problem. Then, the next section deals with the introduction of this problem closely related to the study of the uniqueness of the result of a product operation. The connection between the cancellation problem and the DDS equations will be explored in Chapter 6. For further study of the product of graphs, one can refer to several books, as for example [Hammack et al. (2011)].

It is noteworthy how the products definitions above can be applied both on graphs and digraphs. In the following, products will always involve oriented graphs (as we consider dynamics

[1] Two nodes (or vertices) of a graph are adjacent iff they are joined by an edge.

graphs), however, the case of undirected graphs is also studied in this chapter since the properties that apply to them are also relevant. Simply given the fact that an undirected graph corresponds to a symmetric digraph in which for each pair of adjacent nodes there are both directed arcs. For the sequel we recall that graphs like digraphs consist of a set of vertices, but in the former case the arcs are not oriented (so we denote them with an unordered pair $[g_1, g_2]$ such that $g_1, g_2 \in V(G)$) while in the latter they are (so we denote them with a pair (g_1, g_2) such that $g_1, g_2 \in V(G)$).

When studying the direct product over graphs, it is important to introduce some simple but important properties. First, about fixed points, $G \cdot H$ has a fixed point in (g, h) if and only if g and h are fixed points in the original systems. Then, about the distance[2] of vertices in the result of the product operation and the distance in the original graphs, two properties are known [Lamprey and Barnes (1974)].

Proposition 2.1.1 ([Hammack et al. (2011), Proposition 5.7]). *Suppose (g, h) and $(g', h') \in V(G \cdot H)$, and l is an integer for which G has a walk[3] from g to g' of length l and H has a walk from h to h' of length l. Then $G \cdot H$ has a walk of length l from (g, h) to (g', h'). The smallest such l is $d_{G \cdot H}((g, h), (g', h'))$. If it does not exist, $d_{G \cdot H}((g, h), (g', h')) = \infty$.*

Thanks to the associativity of this type of product operation, the previous proposition is valid also in the case of a product between multiple graphs. Let $\rho_i(g)$ denote the i-th component of a state g of the resulting graph of a product operation $G_1 \cdot G_2 \cdot \ldots \cdot G_k$. More formally, let $\rho_i : G_1 \cdot G_2 \cdot \ldots \cdot G_k \to G_i$ be a *projection map* such that $\rho_i(g_1, g_2, \ldots, g_k) = g_i$.

Proposition 2.1.2 ([Hammack et al. (2011), Proposition 5.8]). *Consider g and g' two states of $G = G_1 \cdot G_2 \cdot \ldots \cdot G_k$. The distance $d_G(g, g')$ is the $\min\{l \in \mathbb{N} \mid$ each G_i has a walk of lenght l from $\rho_i(g)$ to $\rho_i(g')\}$.*

Let us underline that these properties introduced above are valid also in the case of digraphs (see Figure 2.2). This ideas will make a difference in the study of the transients parts of DDS (in Chapter 6).

Remark that if the distance between two different points of the result of a product $d_{G \cdot H}((g, h), (g', h'))$ can be ∞, the result of a product operation can be disconnected. In the previous chapter, it was anticipated that the product between two connected DDS can produce a digraph with one to several components. Given the fact that the direct product distributes over the sum (*i.e.* disjoint union), it is interesting to study the result of multiplication operations between connected graphs. Weichsel's theorem provides a way to determine when the result of multiplying two connected and undirected graphs (with possible loops) is itself connected or not [Weichsel (1962)]. Recall that a graph is bipartite if the set of vertices can be partitioned into two disjoint sets such that no two vertices within the same set are adjacent. In other words, it is bipartite iff it is 2-colorable[4].

Theorem 2.1.3 (Weichsel's Theorem). *The result of $G \cdot H$ is connected if at least one of G and H, connected non trivial undirected graphs admitting loops, has an odd cycle. If both are bipartite, $G \cdot H$ has two components.*

[2] The distance $d_G(g, g')$ between two vertices g and g' of G is the length of the shortest path between the two.

[3] A walk of length l in a graph is a sequence of l (adjacent) not necessarily distinct vertices of the graph.

[4] A 2-colorable graph is a graph in which each vertex can be assigned a color and none of its adjacent vertices have the same color.

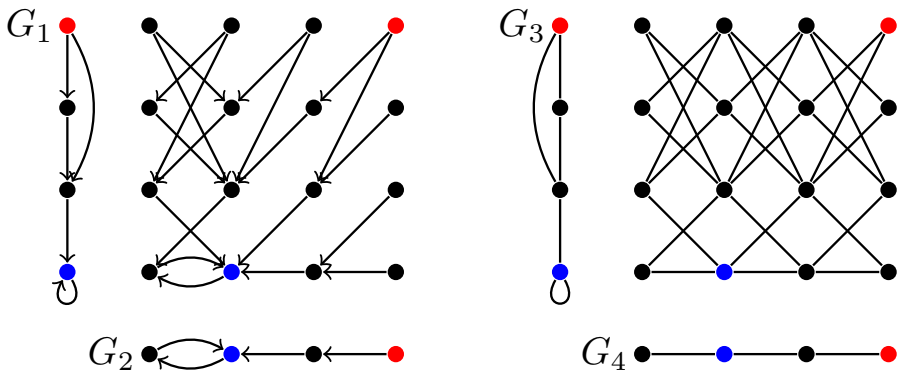


Figure 2.2 – From the red node in G_1 to the blue node there are paths of length $\{2, 3, 4, 5, 6, \ldots\}$. From the red node in G_2 to the blue node there are paths of length $\{2, 4, 6, \ldots\}$. In the result of their direct product there are paths of length $\{2, 4, 6, \ldots\}$ from the red node to the blue node. The same is also observed in the undirected case.

The idea is that if we consider two connected bipartite graphs, and consider two vertices of their product (g, h) and (g', h') such that g and g' belong to the same partition of G, and such that h and h' do not belong to the same partition of H, then the paths between the two nodes of G will always have even lengths in contrast to those between h and h'. This fact leads to the conclusion that the result is disconnected.

In the case of a multiplication between several connected graphs, it is known that a result with 2^{b-1} components is obtained (where b is the number of bipartite graphs involved). In other words, the result is connected only if there is at most one bipartite graph involved.

If one wonders what are the possible factors (graphs involved in the product) of a certain graph, it is important to point out that every nontrivial graph has prime factorisation with respect to the direct product, but there is no unique prime factorisation of: undirected graphs with loops at each vertex, connected undirected graphs admitting loops, and connected undirected graphs without loops. Remark that a graph G is *prime* if and only if it can be generated by the product $G_1 \cdot G_2$ of two graphs as long as one of them consists of only one loop (*i.e.* neutral element for the product operation). Then, according to this definition, a prime factorisation involves just prime factors. However, it has been proved that connected non-bipartite graphs admitting loops admit a unique prime factorisation. For more details, the reader can check the Chapter 8 of [Hammack et al. (2011)]. The lack of existence of a unique factorisation for connected oriented graphs was anticipated in the previous chapter, but it is important to point out the state-of-art of undirected graphs as well.

If we focus exclusively on directed graphs several problems become more complex. For example, results obtained on connectivity (such as Theorem 2.1.3) can no longer be applied as such given constraints imposed by the orientation of the arcs. Let $\mathrm{g}(G)$ be the $\gcd(|\mathcal{C}_1|, |\mathcal{C}_2|, \ldots, |\mathcal{C}_L|)$ of all cycle lengths present in a graph G with L cycles. Then, the following theorem allows one to calculate how many components are obtained from the multiplication of a finite number of strongly connected digraphs [McAndrew (1963)]. Let us recall that a *strong component* is a maximal strongly connected sub-digraph of a digraph G.

Theorem 2.1.4 ([McAndrew (1963)]). *The result of the Direct product $G_1 \cdot G_2 \cdot \ldots \cdot G_k$, with $G_1, G_2, \ldots, G_k$ strongly connected digraphs, contains*

$$\frac{\mathrm{g}(G_1) \cdot \mathrm{g}(G_2) \cdot \ldots \cdot \mathrm{g}(G_k)}{\mathrm{lcm}(\mathrm{g}(G_1), \mathrm{g}(G_2), \ldots, \mathrm{g}(G_k))}$$

strong components.

It is important to point out that in order to apply the theorem to the dynamics graphs of the previous chapter, it is necessary to consider only the periodic (*i.e.*, cyclic) part of the digraphs (this will be explored in Chapter 5). In the literature, it has also been proved that $G_1 \cdot G_2 \cdot \ldots \cdot G_k$ can be strongly connected iff each factor of the product is strongly connected too and iff $\mathrm{g}(G_1), \mathrm{g}(G_2), \ldots, \mathrm{g}(G_k)$ are co-prime [Hammack et al. (2011)].

To complete this presentation regarding the connectivity of the Direct product over digraphs, one can consider in which case $G_1 \cdot G_2 \cdot \ldots \cdot G_k$ is unilaterally connected. A digraph is *unilaterally connected* if for each pair of vertices there is a directed walk in one of the direction. It has been proved that $G_1 \cdot G_2 \cdot \ldots \cdot G_k$ is unilaterally connected if and only if just one factor G_i is unilaterally connected (but not strongly connected), the result G' of the product of the remaining factors is strongly connected, and iff, for each strong component C of G_i, the product $C \cdot G'$ is strongly connected too [Harary and Trauth (1966)].

Considering the prime factorisation of digraphs, one must consider specific type of digraph in order to have a unique factorisation. In fact, just finite digraphs in which any two vertices are connected by even anti-walks[5] admit unique prime factorisation [McKenzie (1971)].

2.2 Cancellation for the Direct Product

As anticipated, the objective in the sequel will be to solve polynomial equations over DDS with a constant right-hand term. Given the distributivity of the product operation presented before, Chapter 4 will detail that it is essential to study equations involving a single monomial. A first question that may arise concerns the uniqueness of the solution. This question is tightly connected with the cancellation problem in graph theory. In the cancellation problem one wants to study, given a specific product of graphs $*$, under which conditions $G * H \cong G' * H$ implies $G \cong G'$. It has been shown that cancellation holds for the Cartesian product [Hammack et al. (2011), Theorem 6.21, p.73] and Strong product [Hammack et al. (2011), Theorem 9.5, p.108], but does not hold in general for the direct product [Hammack et al. (2011), p.426]. In particular, in the case of the direct product, the problem is even more complex in the case of direct graphs [Hammack et al. (2011)]. Consequently, in this section too, the non-oriented graphs are considered first and then the oriented ones. Also in the literature, results were first introduced on graphs and then extended to digraphs.

2.2.1 Cancellation over graphs

In the case of the cancellation problem over undirected graphs, it is known that when H is bipartite, the cancellation property does not hold. An example is shown in Figure 2.3. Nevertheless, a number of conditions have been provided to characterise in which situations the cancellation property holds. Let us start to consider H.

[5] An anti-walk in a digraph is a walk in which the orientation of the arcs is alternated.

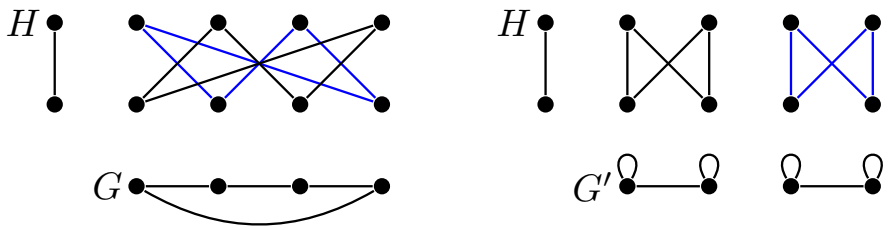


Figure 2.3 – An example where $G \cdot H \cong G' \cdot H$ but $G \ncong G'$.

The fundamental point of the reasoning is that $G \cong G'$ if $hom(J, G) = hom(J, G')$ for all directed graphs J admitting self loop, where $hom(J, G)$ is the number of homomorphisms[6] from J to G [Lovász (1971), Theorem 1]. Moreover, it is known that $hom(J, G \cdot H) = hom(J, G) \cdot hom(J, H)$.

Proposition 2.2.1. *If $G \cdot H \cong G' \cdot H$ and H has a self loop, then $G \cong G'$.*

The last proposition is based on the fact that, since $G \cdot H \cong G' \cdot H$, $hom(J, G \cdot H) = hom(J, G' \cdot H)$ and then $hom(J, G) \cdot hom(J, H) = hom(J, G') \cdot hom(J, H)$. However, for any directed graph J there is always the homomorphism that maps all the nodes of J to the node with loop of H. Then, $hom(J, H)$ cannot be equal to 0 and $hom(J, G) = hom(J, G')$ which implies $G \cong G'$. In addition, this last proposition can be extended as follows [Lovász (1971), Theorem 6].

Proposition 2.2.2. *If $G \cdot H \cong G' \cdot H$ and there are homomorphisms from G to H and from G' to H, then $G \cong G'$.*

Corollary 2.2.3. *If $G \cdot H \cong G' \cdot H$ and G, H and G' are bipartite, then $G \cong G'$.*

Studying the cancellation problem over undirected graphs, it is important to note that it has been shown that if $G \cdot H \cong G' \cdot H$ and there is a homomorphism from a graph H' to H, then $G \cdot H' \cong G' \cdot H'$. When H has at least one arc, the previous result allows to relate the problem back to the case where H' is the complete graph with just two nodes (often denoted K_2).

In [Lovász (1971), Theorems 9 and 7], two major generalisations of Proposition 2.2.1 have been introduced.

Theorem 2.2.4. *If $G \cdot H \cong G' \cdot H$ and H has an odd cycle, then $G \cong G'$.*

Theorem 2.2.5. *Given G, H and G' undirected graphs (admitting self loops) and $g \in V(G)$, $h \in V(H)$, and $g' \in V(G')$. If $G \cdot H \cong G' \cdot H$, then there is a homomorphism from $G \cdot H$ to $G' \cdot H$ of the form $(g, h) \mapsto (\varphi(g, h), h)$ for some homomorphism $\varphi : G \cdot H \to G'$.*

Let us point out that Theorem 2.2.4 implies that, when H bipartite, there exist G and G' such that $G \ncong G'$. However, one can try now to provide more conditions over G and G' to determine other cases in which the cancellation property holds. This has been made thanks to the notions of anti-automorphism and factorial.

[6] A homomorphism φ from J to G is a mapping from $V(J)$ to $V(G)$ for which $[\varphi(j), \varphi(j')] \in E(G)$ whenever $[j, j'] \in E(J)$.

2.2.1.1 Anti-automorphism and Factorials

In [Hammack (2009)], given a graph G and a bipartite graph H, Hammack characterises the graphs G' for which the cancellation property holds. This consists, basically, in the identification of all those G's for which it is certain that $G \cdot H \cong G' \cdot H$ implies $G \cong G'$. These results complete the characterisation given by Lovász. The idea is to exploit the concept of anti-automorphism.

Definition 2.2.1 (Anti-automorphism of a graph)**.** An *anti-automorphism* of a graph G is a bijection $\mu : V(G) \to V(G)$ with the property that an edge $[g, g'] \in E(G)$ iff $[\mu(g), \mu^{-1}(G')] \in E(G)$ for all $g, g' \in V(G)$.

The set of anti-automorphisms of a graph G is $Ant(G)$. Additionally, let G^μ be the graph with $V(G^\mu) = V(G)$ and $E(G^\mu) = \{[g, \mu(g')] \mid [g, g'] \in E(G)\}$.

Proposition 2.2.6 ([Hammack (2009), Proposition 5])**.** *If $G \cdot H \cong G' \cdot H$ and H is a bipartite graph with at least one edge, then $G' \cong G^\mu$ for some $\mu(G)$.*

Among the various anti-automorphisms that can be defined on a graph G, it has been shown that it is possible to introduce equivalence classes. Formally, given $\mu, \mu' \in Ant(G)$, one can define the two anti-automorphisms equivalent (*i.e.* $\mu \simeq \mu'$) if and only if $G^\mu \cong G^{\mu'}$.Then, given a $\mu \in Ant(G)$, $[\mu]$ denotes the equivalence class containing μ.

Theorem 2.2.7 ([Hammack (2009), Theorem 9])**.** *If the equivalence classes of $Ant(G)$ are $\{[\mu_1], [\mu_2], \ldots, [\mu_k]\}$, then the isomorphism classes of the graphs G' such that $G \cdot H \cong G' \cdot H$ are those in $\{G^{\mu_1}, G^{\mu_2}, \ldots, G^{\mu_k}\}$.*

A related question asks to find G such that the cancellation property always holds (regardless of G' and H, which however must have at least one edge). Such a G is called a *cancellation graph*. According to the previous theorem, it is easy to understand that in the latter case $Ant(G)$ contains only one equivalence class that in its turn contains all anti-automorphisms. Another interesting connection has been pointed out the connection between cancellation graphs and the factorial of graphs. The *factorial of a graph* G (denoted $G!$) is a graph where $V(G!)$ is the set of permutations of $V(G)$ (denoted $Perm(V(G))$), and $E(G!)$ contains all $(\pi, \pi') \in V(G!) \times V(G!)$ such that $(g_1, g_2) \in E(G) \Leftrightarrow (\pi(g_1), \pi'(g_2)) \in E(G)$ for all $g_1, g_2 \in V(G)$.

Theorem 2.2.8 ([Hammack (2009), Theorem 10])**.** *A graph G is a cancellation graph iff every anti-automorphism μ of G can be factored into the composition $\mu = \pi\pi'$ where $[\pi, \pi'] \in E(G!)$.*

Moreover, it has been proved that if every anti-automorphism of G has odd order, then G is a cancellation graph [Hammack (2009), Corollary 11]. As well as, a bipartite graph is a cancellation graph if and only if none of its components admits an involution that interchanges its parts[7] [Hammack (2009), Corollary 12].

2.2.2 Cancellation over digraphs

The cancellation still fails when considering direct products over digraphs (Figure 2.4 shows an example). Moreover, it is well known that the cancellation problem over digraphs is even more

[7] A bipartite graph is a graph whose vertices can be divided into two disjoint and independent sets usually called the parts of the graph.

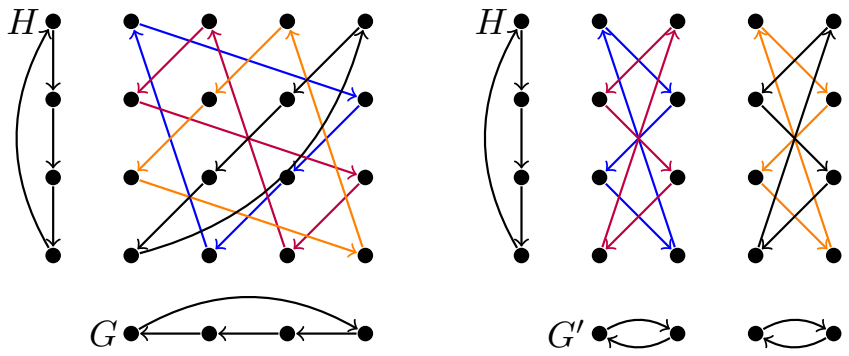


Figure 2.4 – An example where $G \cdot H \cong G' \cdot H$ but $G \ncong G'$.

challenging. Considering digraphs, an important known result consists of the characterisation of *zero divisors*. A graph H is a zero divisor if there exists non isomorphic digraphs G and G' such that $G \cdot H \cong G' \cdot H$. For example, according to the previous section, bipartite undirected graphs are zero divisors. Let C_p an oriented cycle of length p and P_n a directed path between n vertices (of length $n-1$).

Theorem 2.2.9 ([Lovász (1971), Theorem 8]). *A digraph H is a zero divisor if and only if there is a homomorphism $H \to C_{p_1} + C_{p_2} + \ldots + C_{p_k}$ with $p_1, p_2, \ldots, p_k$ prime numbers.*

Then, the following objective can be to enumerate, for a certain G and a certain zero divisor H, the digraphs G' such that $G \cdot H \cong G' \cdot H$. In [Hammack and Toman (2010)], a first contribution in this direction has been made. Considering $H \cong P_2$, they first introduce a construction to find all the graphs G' such that $G \cdot P_2 \cong G' \cdot P_2$ for a certain G. Given a permutation π of the vertices $V(G)$, the digraph G^π has the same vertices of G and the edges $(g_1, \pi(g_2))$ such that $(g_1, g_2) \in E(G)$. This construction allows to identify all digraphs G' such that $G \cdot P_2 \cong G' \cdot P_2$ (*i.e.* $G \cdot P_2 \cong G^\pi \cdot P_2$). Let us point out that G^π can be non isomorphic to G (see Example 2.2.1).

Proposition 2.2.10 ([Hammack and Toman (2010), Proposition 1]). *If G, G' are digraphs, $G \cdot P_2 \cong G' \cdot P_2$ iff $G \cong G^\pi$ for some $\pi \in Perm(V(G))$.*

Before continuing it is important to note that Theorem 2.2.5 and the fact that $G \cdot H \cong G' \cdot H$, such that there is a homomorphism from a graph H' to H, implies $G \cdot H' \cong G' \cdot H'$ are also true on directed graphs. Consequently, we can replace P_2 with all those graphs that admit homomorphism to the latter. Hence, considering the digraphs G, G', H such that there is a surjective homomorphism from H to P_2, it holds that $G \cdot H \cong G' \cdot H$ iff $G' \cong G^\pi$ for some $\pi \in Perm(V(G))$. Then, considering a digraph H with at least one arc, only one direction of the previous statement can be generalised, namely, if $G \cdot H \cong G' \cdot H$ then $G \cong G^\pi$ for some $\pi \in Perm(V(G))$. Thus, there can be $|V(G)|!$ potentially non-isomorphic G'. However, different permutations can lead to isomorphic digraphs. To resolve this, the same work generalises the notion of factorial to digraphs obtaining a similar theorem.

Theorem 2.2.11 ([Hammack and Toman (2010), Theorem 4]). *Suppose G and H are digraphs and there is a surjective homomorphism $H \to P_2$. Let $\pi_1, \pi_2, \ldots, \pi_k \in V(G!)$ be representatives from the k equivalence classes of $\simeq$. Then, the digraphs G' (up to isomorphism) for which $G \cdot H \cong G' \cdot H$ are exactly $G' \in \{G^{\pi_1}, G^{\pi_1}, \ldots, G^{\pi_k}\}$.*

Even in the context of oriented graphs, G is defined as a cancellation digraph when there is only one equivalence class that contains all possible permutations.

Example 2.2.1 – Consider a certain digraph G (see Figure 2.5). According to the definitions, one have six possible permutations $Perm(V(G)) = \{id, 12, 01, 02, 012, 021\}$. For clarity, id represents the identity and the others correspond to the following permutations π.

$$12 \rightarrow \pi(0) = 0, \pi(1) = 2, \pi(2) = 1,$$
$$01 \rightarrow \pi(0) = 1, \pi(1) = 0, \pi(2) = 2,$$
$$02 \rightarrow \pi(0) = 2, \pi(1) = 1, \pi(2) = 0,$$
$$012 \rightarrow \pi(0) = 1, \pi(1) = 2, \pi(2) = 0,$$
$$021 \rightarrow \pi(0) = 2, \pi(1) = 0, \pi(2) = 1.$$

According to Proposition 2.2.10, with these permutations one obtains the following G^π for which $G \cdot P_2 \cong G' \cdot P_2$ (see Figure 2.6).

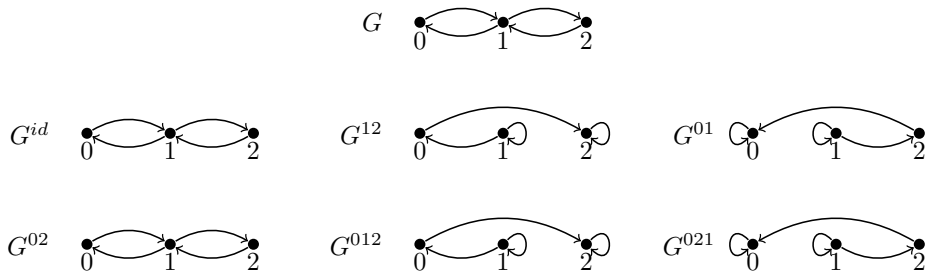


Figure 2.5 – Examples of G^π.

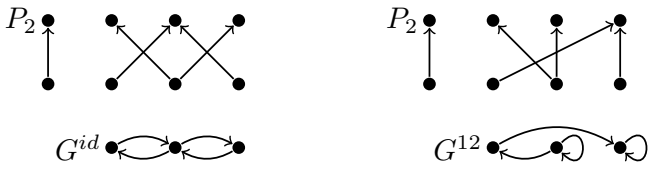


Figure 2.6 – Direct products $G^{id} \cdot P_2$ and $G^{12} \cdot P_2$. Remark that $G^{id} \cdot P_2 \cong G^{12} \cdot P_2$, where G is defined in Figure 2.5.

It can be seen that four graphs are isomorphic to each other as well as the rest to each other. This is also evidenced by the factorial of G.

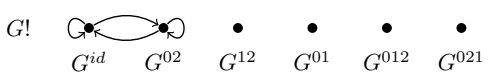


Figure 2.7 – Factorial of G (as defined in Figure 2.5).

To complete this example, let us consider a digraph H that does not admit surjective homomorphism to P_2 and another H' that does. As anticipated, in the first case the product result may not be isomorphic. Whereas in the second case, we observe that the cancellation property holds.

Figure 2.8 – Direct products $G^{id} \cdot H$ and $G^{12} \cdot H$. Remark that $G^{id} \cdot H \ncong G^{12} \cdot H$, where G is defined in Figure 2.5.

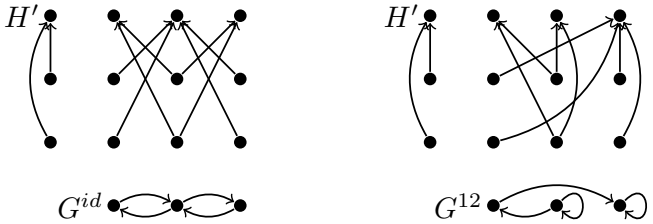


Figure 2.9 – Direct products $G^{id} \cdot H'$ and $G^{12} \cdot H'$. Remark that $G^{id} \cdot H' \cong G^{12} \cdot H'$, where G is defined in Figure 2.5.

To conclude the state of the art, it is important to note that two other families of digraphs, denoted with V_r^t and Λ_r^t, have been shown to be cancellation digraphs. Given non-negative integers r and t, the digraph V_r^t is based on the disjoint union of a complete oriented digraph with t vertices (admitting self loops) and r isolated nodes. Additionally, V_r^t also contains arcs pointing from each vertex of the complete sub-graph to every isolated vertex. Similarly, Λ_r^t is based on the same two sub-graphs but contains arcs pointing from each isolated vertex to every vertex of the complete sub-graph.

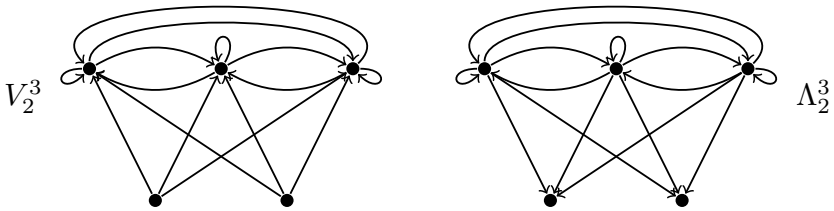


Figure 2.10 – Examples of V_r^t and Λ_r^t for $r = 2$ and $t = 3$.

To conclude, let us explore the problem of cancellation on dynamics graphs as introduced in the previous chapter. We know that H will always be a zero divisor according to Theorem 2.2.9. In fact, by the distributivity of the product, we can always consider H as a dynamics graph containing only one component and consequently only one cycle. It therefore admits homomorphism to a disjoint union of cycles of prime length as required in the theorem. In particular, it is possible to

map the cyclic nodes of the dynamics graph to a cycle of prime length C_P (that divides the length of cycle of the dynamics graph), and to map the transient nodes to C_p in such a way that distances established by the next-state map are preserved. To do this, for every $v \in \mathcal{T}$ such that $f^h(v) = v'$, one needs to map v to the h-th predecessor of the node to which v' is mapped.

However, a dynamics graph does not admit homomorphisms to P_2 since a cycle cannot be mapped. Finally, considering a dynamics graph G, it is not certain that all permutations would be valid G' such that $G \cdot H \cong G' \cdot H$. The connection between the cancellation and the set of solutions of an equation over DDS will play a central role in Chapter 6.

CHAPTER 3

Multi-valued Decision Diagrams

This chapter is a gentle introduction to Multi-valued Decision Diagrams (MDD). This is a quite convenient data structure that allows compact representations of functions or sets. We will review the main known algorithms to perform operations on MDD (reduction, intersection, difference, union, etc). As we will see, most of these operations can be performed without decompressing the structure. This last feature and the exponential compression that can be achieved by the reduction operation are some of the reasons behind the success of MDD. The next chapters will also deeply exploit these features.

3.1 BDD and MDD: useful data structures

Binary decision diagrams (BDD) were introduced by Lee in the second half of the 1990s [Lee (1959)] and from then on they started to become popular for their various applications [Akers (1978)]. They arise from the need to represent and manipulate Boolean functions that appear in many computer science fields (such as artificial intelligence, combinatorics, *etc.*). Many questions involving Boolean functions are pretty hard to solve (an NP-completeness is always lurking). The idea is therefore to symbolically represent these functions and to introduce clever algorithms which try to avoid the exponential aspect of the problems one wants to solve. In this sense, the real milestone was the publication by Bryant in 1986 [Bryant (1986)]. He introduced the ordering of variables in BDD to obtain a canonical form and present several efficient algorithms to manipulate these structures. It was only at this point that this data structure was fully recognised for its potential. Later, it has also been generalised to be applied to multi-valued functions. In this case, one is dealing with *Multi-valued Decision Diagrams* (MDD). This is the version of the data structure that will be used in the following. Hence, after an introduction to BDD, this chapter presents MDD, its reduction (a fundamental operation of this structure), and certain operations that will be useful later on. For further information, reference can be made to some classic books [Knuth (2011), Bergman et al. (2016)]. Recent research on these topics can found in [Jung and Régin (2022), Gillard et al. (2021)], for example.

3.1.1 Binary Decision Diagrams

A BDD is a Directed Acyclic Graph (DAG) used to represent a Boolean formula $f_B(b_1, b_2, \ldots, b_r)$ in a compact way. It comprises a *root* node (*i.e.* a node without incoming arcs) and a series of internal nodes, also called branch nodes, usually characterised by a name (or a value) to refer to one of the Boolean variables in the formula. Each node has two outgoing arcs to represent the assignment of a `0` (*false*) or a `1` (*true*) to a certain variable. Finally, there are two terminal nodes (also called sink nodes) which represent the final value of the Boolean function. In the literature, the terminal nodes of BDD are denoted by `0` and `1` (or by $\bot$ and $\top$ to avoid confusion with respect to the numbering of variables or the value assigned to them). Hence, in the whole structure, a path from the root to a terminal node represents an assignment to the variables of the formula. Figura 3.1 illustrates the fact that all assignments of a Boolean formula are represented by its truth table, as well as, by the corresponding MDD. The aim here is to have a structure that is more efficient in memory and allows us to answer certain problems or execute certain operations more efficiently [Knuth (2011)].

Example 3.1.1 – Consider the Boolean variables b_1, b_2, b_3, b_4 and the Boolean formula $f_B(b_1, b_2, b_3, b_4) = b_1 \wedge b_2 \vee b_3 \wedge \neg b_4$. In accordance with the definition of BDD provided, the structure in Figure 3.1 (right) corresponds to the set of assignments contained in the truth table of Figure 3.1 (left).

Consider the variables in order of appearance. The variable b_1 is then evaluated first and is modelled in the root node. A value of `0` or `1` can be assigned. In the case of BDD, one often finds these two possibilities represented by different types of arcs. Here, a dashed arc represents a `0`-value assignment and a continuous arc represents a `1`-value assignment. The root shows a dashed arc and a continuous arc to represent the two possibilities for b_1. Next, the

same process is repeated on b_2 and b_3. Finally, in accordance with the assignment of the fourth and last variable, the final value of the function is determined.

b_1	b_2	b_3	b_4	f
0	0	0	0	0
0	0	0	1	0
0	0	1	0	1
0	0	1	1	0
0	1	0	0	0
0	1	0	1	0
0	1	1	0	1
0	1	1	1	0
1	0	0	0	0
1	0	0	1	0
1	0	1	0	1
1	0	1	1	0
1	1	0	0	1
1	1	0	1	1
1	1	1	0	1
1	1	1	1	1

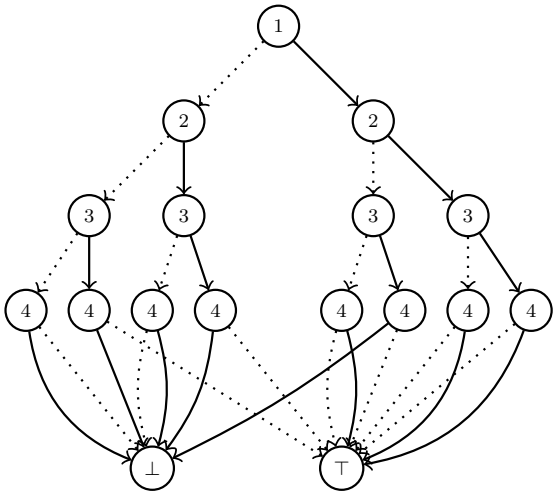


Figure 3.1 – A BDD associated with the function $f_B(b_1, b_2, b_3, b_4) = b_1 \wedge b_2 \vee b_3 \wedge \neg b_4$. The name of the nodes here indicates which variable is assigned. A node i refers to the variable b_i of the formula.

Example 3.1.1 aims at showing how BDD are capable of representing the possible values of variables and function in the structure. However, it is essential to highlight that, when we talk about BDD, we are actually referring to *Reduced Ordered Binary Decision Diagrams* (ROBDD) introduced by Bryant. The *reduced* term means that:

- all nodes with both outgoing arcs arriving at the same node are removed (as it signifies that the assignment is irrelevant to the final value of the function);
- all nodes with same arcs and identical destination (*i.e. equivalent* nodes) are merged ;
- nodes that are not on a root-to-terminal node path are removed.

The term *ordered* indicates the fact that variables are assigned or evaluated only once, and this is always done in the same order within the structure. This means that we cannot have sub-parts of the structure in which variables are evaluated in different orders. For example, the BDD in Figure 3.1 is ordered but it is not reduced. Example 3.1.2 shows the reduced version and introduces the idea behind an algorithm to reduce decision diagrams.

It is clear that different orders can lead to different BDD and, in particular, to BDD of different size (in terms of number of nodes and arcs). Figure 3.2 shows an example. Moreover, finding the order that minimises the size of the structure is known to be NP-complete. Given the previous definitions and the example, one can understand that, in the worst case, a function needs a graph of exponential size with respect to the number of arguments in order to be represented. These are well-known problems with decision diagrams. However, it is has been shown how in many

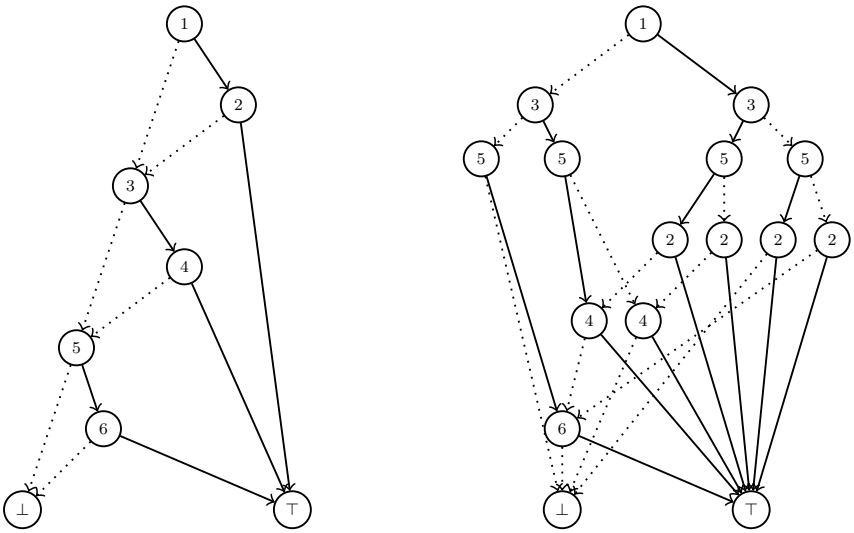


Figure 3.2 – Two BDD associated with the function $f_B(b_1, b_2, b_3, b_4, b_5, b_6) = (b_1 \wedge b_2) \vee (b_3 \wedge b_4) \vee (b_5 \wedge b_6)$ [Minato (2007)]. Both are reduced and ordered but they are based on two different variable orderings.

functions derived from applications, this explosion in memory does not happen (also thanks to the reduction process). It has also been observed that the choice of a good (though perhaps not optimal) ordering of variables is possible thanks to a deeper knowledge of the problem or to the use of heuristics. This reason further motivated the diffusion of BDD in applicative context among which one can list, for instance, logic, verification, model checking, and optimisation [Wille and Drechsler (2009), Drechsler et al. (2004), Chaki and Gurfinkel (2018)].

Thanks to the introduction of the order between variables, reduced BDD have a canonical form (*i.e.* each function has only one possible representation). This implies that two BDD are equivalent if and only if two Boolean functions are equivalent. This is not only important for the equivalence problem, but also allows efficient algorithms to be introduced. For example, the possibility of calculating Boolean operations between two functions, the complement of a function, testing implication, restricting the function according to specific values of certain variables, composing functions, and many others. All of these operations respond to a closure property for a fixed ordering of variables and are calculated by algorithms that scale well as the complexity is limited, in a general way, by the size of the graphs representing the functions [Bryant (1986)].

Additionally, these decision diagrams owe their popularity to a number of other interesting properties. These include:

- evaluating a function in at most r steps (where r is the number of variables). Indeed, one just needs to go through the structure;
- find the lexicographically smallest solution that satisfies the formula by always traversing the arc corresponding to `0` until it is found one that assigning `1` leads to the final $\top$ node (thus only r steps);
- count the number of solutions satisfying the formula in linear time (with respect to the number of nodes and variables in the formula);

- generate a randomised solution or all solutions satisfying the formula in linear time (with respect to the number of nodes and solutions in the formula).

Example 3.1.2 – Consider again the Boolean formula $f_B(b_1, b_2, b_3, b_4) = b_1 \wedge b_2 \vee b_3 \wedge \neg b_4$. Reference is normally made to ROBDD. Then, we must reduce the previous BDD by removing nodes where assigning the variable to `0` or `1` does not change the value of the function and by merging those that are equivalent. The reduction is based on the idea of looking for these nodes by traversing the structure from bottom to top.

Nodes highlighted in red are nodes where the assignment of the variable b_4 has no impact on the final value of the function (since both assignments lead to the same final node) and then can be removed. When a node is removed, we remove its outgoing arcs and redirect its incoming ones. As a result, here, the incoming arcs are redirected to the final nodes. The nodes highlighted in purple are equivalent. They can therefore be merged into a single node.

Going up one level in the structure, the nodes in blue are the ones that become equivalent (given the changes made at the lower level), and instead the brown node is still a case where a different assignment does not change the final value.

In the end, the pink node becomes negligible by the time the two child nodes are turned into the same node.

Through this reasoning, one gets the ordered and reduced version of the structure. Finally, we stress that this is the form that it is normally meant by the term BDD.

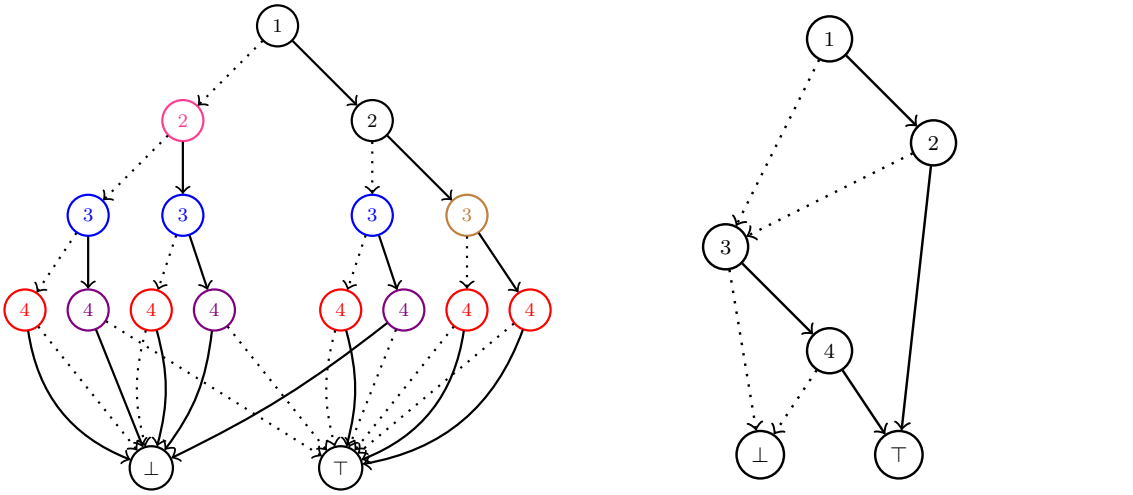


Figure 3.3 – An unreduced BDD for the function $f_B(b_1, b_2, b_3, b_4) = b_1 \wedge b_2 \vee b_3 \wedge \neg b_4$ (left) and its ROBDD (right).

3.1.2 Multi-valued Decision Diagrams

Multi-valued Decision Diagrams (MDD) are a generalisation of BDD [Bergman et al. (2016)] used to obtain efficient representations of functions with finite domains or finite sets of tuples. In the last years, MDD have been applied in many different research domains proving the potential of this structure. MDD are used, for instance, to improve random forest algorithms replacing the

classic binary decision trees [Nakahara et al. (2017)], to represent and analyse automotive product data of valid/invalid product configurations [Berndt et al. (2012)], and to perform trust analysis in social networks [Zhang et al. (2019)]. There are also applications related to mathematical models like Multi-State Systems (MSS) [Zaitseva et al. (2013)]. In this case, MDD represent the MSS structure in terms of Multi-Valued Logic to compute some measures. MDD find applications also in the analysis of the discrete dynamical systems. Indeed, in [Naldi et al. (2007)], MDD represents logic rules to analyse some properties and dynamics aspects in the case of regulatory networks (for example, to perform a stable states identification).

An MDD is a rooted DAG created from a finite set of variables with specific domains. In other words, it is a data structure to represent a multi-valued function $f_M : \{0, \ldots, t-1\}^r \rightarrow \{true, false\}$. Associating each variable with a level of the structure, the MDD represents a set of feasible assignments as a path from the *root* (in the first layer) to the final true terminal node, denoted *tt* (in the last one). Therefore, the data structure contains $r+1$ layers and a path from the *root* to the *tt* node represents a valid set of assignments. Each node is characterised by a variable (represented in the layer), a value (if necessary), and at most d outgoing edges directed from an upper layer to a lower one. An edge corresponds to an assignment of the variable to a certain value specified by the label of the edge. In general, this version of the data structure presents only one root and one leaf. In fact, the false terminal node, and the corresponding paths to reach this node, are often omitted.

When constructing an MDD, the starting point is the domains of the variables. However, depending on the problem (or function) to be represented, there are additional constraints on the existence of the arcs in the structure. Example 3.1.3 shows an MDD for a sum problem (a problem similar to the one we will see later), while Example 3.1.4 shows an MDD to find assignments satisfying the All Different (*Alldiff*) constraint. For this reason, more formal definitions of this structure will be provided in Chapters 4 and 5, as they will be related to the problem whose solutions will be modelled.

According to [Darwiche and Marquis (2002), Amilhastre et al. (2014)] , an MDD is normally reduced and ordered as BDD. It is *deterministic* if all the nodes have pairwise distinct labels on outgoing edges. In this manuscript, the structures in the following will always be of this type.

Example 3.1.3 – Consider three variables b_1, b_2 and b_3 with domains $D_1 = \{1, 2, 3, 5\}$, $D_2 = \{2, 3, 5, 7\}$ and $D_3 = \{4, 5, 6\}$, respectively. The problem for this example is to find all assignments such that $b_1 + b_2 + b_3 = 9$. Let us see how one can build an MDD by exploring the solution space (*i.e.* all assignments) to create a structure that will contain all ways to sum the three variables to a total value of 9.

In this example (as in the following), each variable is related to a level and the values within the nodes give us a specific useful information. In this case, the value within the nodes represents the partial sum up to that point. The first level (containing the root node) corresponds to b_1, the second to b_2 and the third to b_3. The fourth and final level contains the final node *tt*.

One starts from the root node with the partial sum set to zero. One considers the possible values d in the domain D_1 of b_1. If and only if the partial sum added to d is less than 9, one creates corresponding node and arc (if this condition is not met, the choice of d is not part of a solution). Therefore, here, all values in D_1 are feasible elements of a solution to the problem. Obviously in new nodes the partial sum is updated according to the value assigned to b_1.

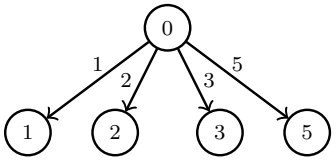


Figure 3.4 – First layer of the MDD of Example 3.1.3. Labels over the outgoing arcs show the possible values of b_1. The label of a node represents the partial sum from the *root* to that node.

For each node in the newly created layer, all possible values for b_2 (contained in D_2) are considered.

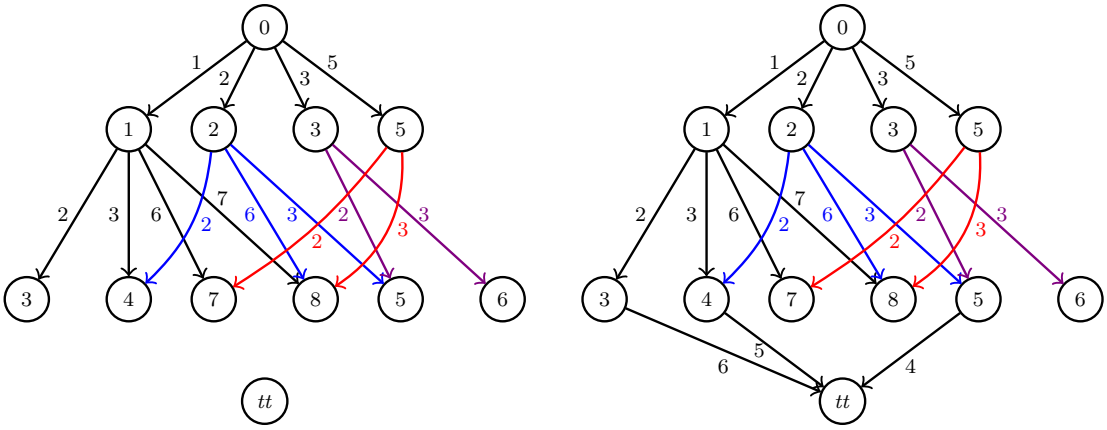


Figure 3.5 – Construction of the second (on the left) and third levels (on the right) of MDD in Example 3.1.3. The labels on the arcs outgoing from the nodes of the second layer show the possible values of b_2. The same is done in the third layer for b_3. The arcs of the second level are highlighted in different colours for clarity.

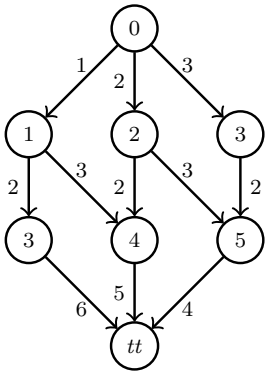


Figure 3.6 – Reduced and ordered MDD of Example 3.1.3.

When the partial sum is set to 1, all values contained in D_2 are admissible. If the partial sum is 2, no outgoing arc with value 7 can be found since the total sum at 9 is expected in the final node *tt* (once that also b_3 is assigned). Remark that $0 \notin D_3$. The problem requires to affect to all variables in their domain so the node is not created. Let us stress that it is a choice of how

one wants to model the problem. Other possible choices would have been to create the node and let it be removed in the reduction step, or to allow only two variables to be used (thus allowing a direct arc to *tt*). The process is finally repeated for b_3. Figure 3.5 shows the result at the end of the construction phase.

Since we refer to ordered and reduced MDD, it becomes necessary to remove all those parts of the structure which do not bring to the final node. In this case, this is the only reduction operation required. Figure 3.9 shows the result.

MDD can be created by, for example, exploring the solution space, from tuple sets, sub-problems, dynamic programming, automaton, and other sources. To discover some methods of creation, the reader may refer to [Perez (2017)].

A crucial feature of MDD is the exponential compression power of the reduction operation and the fact that many classical operations (intersection, union, *etc.*) can be performed without decompression [Perez (2017)].

Example 3.1.4 – Consider three variables b_1, b_2 and b_3 with domains $D_1 = D_2 = D_3 = \{a, b, c, d\}$. The problem for this example is to find all assignments such that $b_1 \neq b_2 \neq b_3$ (*i.e.* all assignments satisfying the *Alldiff* constraint). Let us see how one can build an MDD by exploring the solution space.

In this case, the value within the nodes represents the set of values already chosen from the domain to that point. The first level (containing the root node) corresponds to b_1, the second to b_2 and the third to b_3. The fourth and final level contains the final node *tt*.

One starts from the root node and considers the possible values d in the domain. All values in D_1 are feasible elements of a solution to the problem. In new nodes, the label of the nodes is updated according to the value assigned to b_1.

For each node in the newly created layer, all values for b_2 are considered. For this problem, we therefore consider all domain values not yet used. We therefore see how as the problem changes, the condition for the creation of the arcs changes. The result of the construction (following the reduction operation) is shown in Figure 3.7.

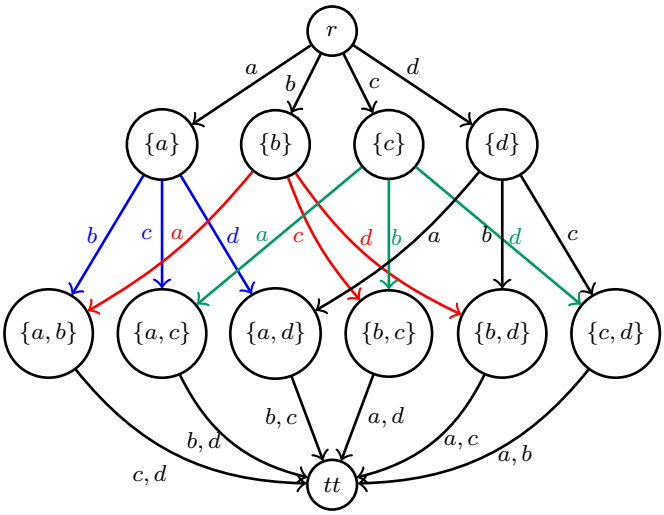


Figure 3.7 – Reduced and ordered MDD for $b_1 \neq b_2 \neq b_3$ with $D_1 = D_2 = D_3 = \{a, b, c, d\}$.

3.2 The reduction operation

This section introduces the *pReduce* reduction algorithm, which is the algorithm used and implemented in this thesis work [Perez and Régin (2015)]. For completeness sake, it is important to note that there are other reduction algorithms proposed in the literature. For example, there are several techniques based on a search (usually a DFS) and a structure able to memorise all the already visited nodes, but also other techniques based on the idea of building clusters of equivalent nodes [Cheng and Yap (2010), Andersen (1997)].

The reduction is a fundamental operation over MDD. In fact, one can gain an exponential factor in the use of memory. To illustrate the algorithm, let us show how the structure can be modelled.

An MDD can be represented as an ordered lists of outgoing arcs of the different nodes. In particular, let $M = (N, E)$ be an MDD. A node $\alpha \in N$ has a list of outgoing arcs E^+_α and a list of incoming arcs E^-_α. Each arc can be viewed as a triple: initial node, arrival node, and arc label. Then, for a node $\alpha \in N$, E^+_α is a list of pairs (d, β) with $d \in D_\alpha$ and $\beta \in N$ where (α, β) is an arc of M with label d.

Here, the list E^+_α is then ordered according to the labels of the arcs.

Example 3.2.1 – According to the definitions provided, the MDD in Figure 3.8 (on the right) corresponds to the data shown in the table (on the left).

$Node$	E^+
r	$\{(1,a),(2,b),(3,c),(5,d)\}$
a	$\{(2,f),(3,e)\}$
b	$\{(2,f)\}$
c	$\{(2,g),(4,h),(5,i)\}$
d	$\{(2,g),(4,h),(5,i)\}$
e	$\{(1,tt)\}$
f	$\{(1,tt),(2,tt)\}$
g	$\{(2,tt)\}$
h	$\{(2,tt)\}$
i	$\{\}$
tt	$\{\}$

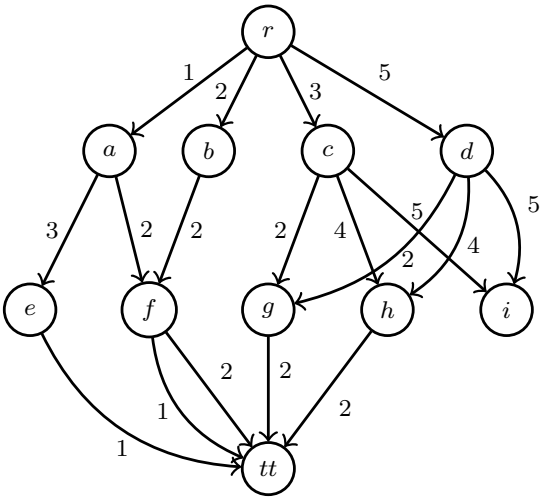


Figure 3.8 – Newly constructed MDD that needs to be reduced.

The reduction aims at merging equivalent nodes (*i.e.* that have equivalent outgoing paths) to reduce the dimension of the data structure but also to obtain a canonical form. Two nodes are equivalent if they have the same label and destination for each edge.

Definition 3.2.1. Two nodes $\alpha, \beta \in M$ are equivalent (*i.e.* $\alpha \equiv \beta$) iff $|E^+_\alpha| = |E^+_\beta|$ and for each arch $(d, \omega) \in E^+_\alpha$, there exists $(d, \omega) \in E^+_\beta$.

Algorithm 1 implements Definition 3.2.1 and thus provides a method for testing whether two nodes are equivalent. We stress that the definition states the fact that if a node has a set of outgoing

arcs that is a subset of the outgoing arcs of another, they cannot be equivalent and therefore cannot be merged. In addition, nodes will be tested for equivalence level by level from bottom to top of the MDD under consideration.

Algorithm 1: Equivalence

Input : α, β nodes of an MDD M
Output: $true$ iff α and β are equivalent, $false$ otherwise

```
1 if |E+_α| ≠ |E+_β| then
2     return false;
3 forall i ∈ {1, ..., |E+_α|} do
4     if E+_α[i] ≠ E+_β[i] then
5         return false ;        ▷ With ≠ that can verify both label and node
6 return true;
```

At this point one needs to describe how the algorithm identifies the nodes to be tested for equivalence. It implements the idea of dividing nodes into *packs* trying to construct groups of equivalent nodes. A pack consists of a set of nodes and an integer z. The integer z indicates that in a certain pack all nodes have the first z equivalent elements in the E^+ lists.

The pReduce starts by putting all the nodes of the layer just above *tt* into a pack (see Algorithm 3). Reducing a layer begins by identifying all those nodes that do not have outgoing arcs. They are removed as well as their incoming arcs (procedure $Delete$ into Algorithm 4). Then, the pack is divided into several sets by comparing the first outgoing edge (*i.e.* the one with the smallest label). To perform this, the algorithm exploits two lists and two structures one indexed by values and the other one by nodes. All the minimum values of the outgoing arcs of different nodes are stored in V_{list}. Then, V_A memorises which nodes have a certain minimum label, for each of the labels in V_{list}. This creates an initial distinction between nodes based on the minimum label of the outgoing arcs. Each of these groups is then re-divided based on the arrival node of the arc. The N_{list} stores the different arrival nodes and N_A stores for each arrival node the associated departure nodes. Now, the nodes have been divided into groups according to their arc with the smallest value and its destination. At this point, in each group, there are two possibilities for each node: all the outgoing arcs of the node have been analysed (in this case, the nodes would have only one outgoing arc), or it has other outgoing arcs. All nodes with only one arc can be merged. While the other nodes form a new pack with z equal to 1. This is because, we are sure that all nodes in the pack have arcs with the same first outgoing arc. The procedure is repeated and the pack is split according to the second arc in the E^+ lists. And so on until one gets empty packs or all the arcs of the nodes of a pack have been considered. In fact, once that all the edges are analysed, the nodes are alone or in the same set with the equivalent nodes.

It is important to point out that when nodes are merged, the different incoming arcs of the nodes being merged are redirected to the merged node (procedure $Merge$ in Algorithm 2). In addition, a Q queue is used to keep track of the packs that are generated and whose $z+1$ arc has yet to be investigated.

Algorithm 2: ReducePack

Input : p pack, $L[i]$ nodes of the layer, E^+ lists of outgoing arcs, $V_A, N_A, V_{list}, N_{list}$ lists, and Q queue

Output: $L[i]$ and E^+ updated according to the equivalent nodes identified

```
1  i ← index(p);
2  forall α ∈ nodes(p) do
3      val ← value(E_α^+[i + 1]);
4      if V_A[val] == ∅ then
5          add val to V_list;
6      add α to V_A[val];
7  forall val ∈ V_list do
8      forall α ∈ V_A[val] do
9          β ← node(E_α^+[i + 1]);
10         if N_A[β] == ∅ then
11             add (val, β) to N_list;
12         add α to N_A[β];
13     V_A[val] ← ∅;
14     forall (val, β) ∈ N_list do
15         if |N_A[β]| > 1 then
16             p' ← pack(∅, i + 1);
17             setM ← {α ∈ N_A[β] | |E_α^+| = i + 1};
18             Merge(setM, L[i], E^+);
19             add N_A[β] \ setM to nodes(p');
20             add p' to Q;
21         N_A[β] ← ∅;
22     N_list ← ∅;
23 V_list ← ∅;
24 return L[i], E^+;
```

It has been shown that the worst time and space complexity of this reduction algorithm is $O(|N| + |E| + |D|)$, linear therefore with respect to the number of nodes, the number of arcs, and the dimension of D (the largest domain in the structure). For completeness sake, it should be said that there exists a parallel version of the algorithm and that there exists a version which is able to reduce a reduced MDD when it is modified.

Example 3.2.2 – Consider the MDD in Figure 3.8. To reduce it, the algorithm begins by putting all the third-level nodes in a single pack $(\{e, f, g, h, i\}, 0)$. The node i is removed since it has no outgoing arcs. As a consequence, the arcs $(5, i)$ in E_c^+ and in E_d^+ are removed too. As a result, the ReducePack method is called on the pack $(\{e, f, g, h\}, 0)$. Thanks to the first loop, $V_{list} = \{1, 2\}$ is identified with $V_A[1] = \{e, f\}$ and $V_A[2] = \{g, h\}$. Among the nodes with minimum arc value 1, the only the possibility is to reach *tt*. In fact, $N_{list} = \{(1, tt)\}$ and

$N_A[tt] = \{e, f\}$. However, the node e presents only one arc while f two. This implies that no node is merged. Even among the nodes with minimum arc value 2, all of them reach the *tt* node. In fact, $N_{list} = \{(2, tt)\}$ and $N_A[tt] = \{g, h\}$. Both g and h have only one outgoing arc. Thus, they are merged into only one node. The incoming arcs are redirected to the latter. The third level of the structure is now reduced.

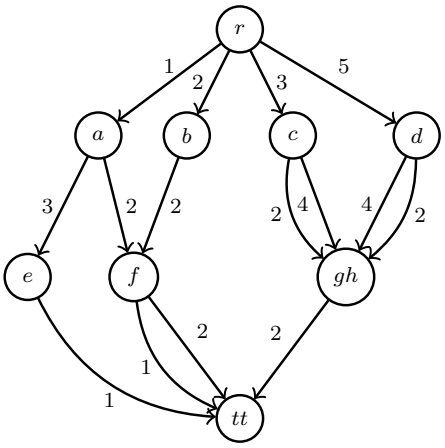


Figure 3.9 – The MDD of Figure 3.8 with reduced third layer.

The same process is repeated on the second level. To reduce it, the algorithm begins by putting all the nodes into a single pack $(\{a, b, c, d\}, 0)$. The ReducePack method is called on it. Thanks to the first loop, $V_{list} = \{2\}$ and $V_A[2] = \{a, b, c, d\}$ are computed. All nodes have minimum arcs of value 2 but some reach f and others g. In fact, $N_{list} = \{(2, f), (2, g)\}$ with $N_A[f] = \{a, b\}$ and $N_A[g] = \{c, d\}$. However, node b has only one arc while a has two. Then, a and b are not merged. To figure out whether c and d can be merged, the algorithm creates a new pack $(\{c, d\}, 1)$.

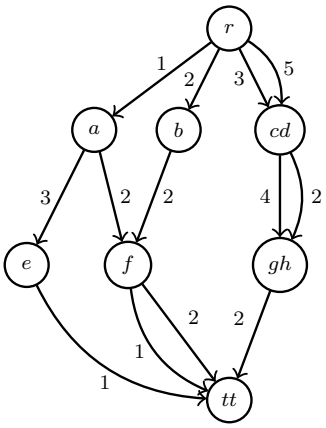


Figure 3.10 – The reduced and ordered version of the MDD in Example 3.2.1.

Reducing this pack results in $V_{list} = \{4\}$ and $V_A[4] = \{c, d\}$. Through the arcs of value 4 of these nodes, h is always reached. In fact, $N_{list} = \{(4, h)\}$ and $N_A[h] = \{c, d\}$. Being the last arc of the two nodes, c and d are merged. The result of the reduction is shown in Figure 3.10.

Algorithm 3: pReduce

Input : L list of nodes in a MDD (according to the layer), and E^+ outgoing arcs lists
Output: L and E^+ corresponding to the reduced MDD

$V_A, N_A, V_{list}, N_{list} \leftarrow \emptyset$;
forall $i \in \{r, \ldots, 1\}$ **do**
  $L[i], E^+ \leftarrow$ ReduceLayer($L[i], E^+, V_A, N_A, V_{list}, N_{list}$);
return L, E^+;

Algorithm 4: ReduceLayer

Input : $L[i]$ set of nodes of layer, E^+ lists of outgoing arcs, and $V_A, N_A, V_{list}, N_{list}$ lists
Output: $L[i]$ and E^+ corresponding to the reduced layer

forall $\alpha \in L[i]$ **do**
  if $|E^+_\alpha| == 0$ **then**
    Delete($L[i], \alpha$);
$p \leftarrow$ pack($L[i], 0$);
$Q \leftarrow \emptyset$;
$L[i], E^+ \leftarrow$ ReducePack($p, L[i], E^+, V_A, N_A, V_{list}, N_{list}, Q$);
while $Q \neq \emptyset$ **do**
  $L[i], E^+ \leftarrow$ ReducePack(takeFirstElement(Q),$L[i], E^+, V_A, N_A, V_{list}, N_{list}, Q$);
return $L[i], E^+$;

3.3 More operations over MDD

Another big advantage of using MDD is that classical set operations such as Cartesian product, complement[1], intersection, union, difference, and many others can be performed without decompressing the structure. For instance, the Cartesian product over MDD can be performed just by merging the *root* of an MDD with the *tt* node of another one (see Figure 3.11 for an example).

Example 3.3.1 – Consider an MDD containing the tuples $\{(1, 2, 2), (1, 2, 3), (2, 2, 3), (2, 3, 3)\}$ and another one with $\{(1, 2, 3), (1, 2, 2), (2, 3, 3), (2, 3, 2), (3, 3, 3), (3, 2, 3)\}$.

[1] see Algorithm *Complementary Operation* in [Perez and Régin (2015)]

The Cartesian product procedure consists in merging the node *tt* of the first MDD with the root of the second one into a single node. In fact, the resulting structure contains the solutions

$$\{(1,2,2,1,2,3),(1,2,2,1,2,2),(1,2,2,2,3,3),(1,2,2,2,3,2),(1,2,2,3,3,3),$$
$$(1,2,2,3,2,3),(1,2,3,1,2,3),(1,2,3,1,2,2),(1,2,3,2,3,3),\ldots,(2,3,3,1,2,3),$$
$$(2,3,3,1,2,2),(2,3,3,2,3,3),(2,3,3,2,3,2),(2,3,3,3,3,3),(2,3,3,3,2,3)\}.$$

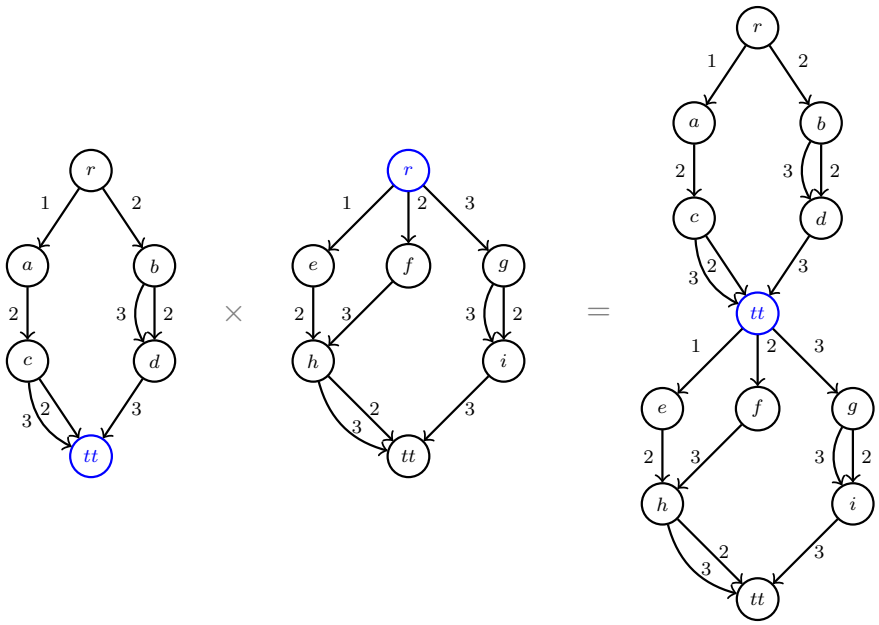


Figure 3.11 – Cartesian product between two MDD.

In [Perez and Régin (2015)], the *Apply* method is introduced to calculate: union, intersection, difference, symmetric difference, union complement, and intersection complement. It is based on the idea proposed by Bryant in [Bryant (1986)] but also on an intersection algorithm proposed later [Bergman et al. (2014)]. In this algorithm, the intersection was computed by exploiting the structure of graphs rather than the functions represented by them. The intuitive idea is to search (level by level in a top-down fashion) for arcs in common between different nodes. To achieve this goal, the first step creates a new root, and only the outgoing edges that are common between the two structures are added. In this way, each new node is linked to two original nodes and the process is iterated.

In the *Apply* method, the idea is basically the same but it allows to compute more than just the intersection. In [Perez and Régin (2015)], it has been shown that the space and time complexity of this method is linearly upper bounded by the size of the product of the two initial MDD. This section presents this method in its different functionalities. In the sequel, we will mainly use Cartesian products and intersections between MDD.

	$op[1]$		$op[2]$		$op[3]$		$op[4]$	
$layer$	$[1..r-1]$	r	$[1..r-1]$	r	$[1..r-1]$	r	$[1..r-1]$	r
$A \cap B$	F	F	F	F	F	F	T	T
$A \cup B$	F	F	T	T	T	T	T	T
$A - B$	F	F	F	F	T	T	T	F
$A \triangle B$	F	F	T	T	T	T	T	F
$\overline{A \cup B}$	T	T	T	F	T	F	T	F
$\overline{A \cap B}$	T	T	T	T	T	T	T	F

Figure 3.12 – The values of $op[i]$ defining the behaviour of the *Apply* method.

When two MDD are combined, each node in the result α (regardless of the type of operation) is associated with two nodes, one in the first MDD α_1 and one in the second α_2. The method is based on the fact that, given a node $\alpha = (\alpha_1, \alpha_2)$ and a value of a label d, there are four possibilities:

1) $\nexists e_1 \in E^+_{\alpha_1} \mid value(e_1) = d$ and $\nexists e_2 \in E^+_{\alpha_2} \mid value(e_2) = d$;

2) $\nexists e_1 \in E^+_{\alpha_1} \mid value(e_1) = d$ and $\exists e_2 \in E^+_{\alpha_2} \mid value(e_2) = d$;

3) $\exists e_1 \in E^+_{\alpha_1} \mid value(e_1) = d$ and $\nexists e_2 \in E^+_{\alpha_2} \mid value(e_2) = d$;

4) $\exists e_1 \in E^+_{\alpha_1} \mid value(e_1) = d$ and $\exists e_2 \in E^+_{\alpha_2} \mid value(e_2) = d$.

In other words, the arc with a certain label may be present for both nodes, for either one of them, or for neither of them. In each of these cases, the behaviour of the algorithm must change according to the operation that one is computing. For example, in the case of an arc which is present only in the first MDD, the arc is preserved in the case of a union but removed in the case of an intersection. Figure 3.12 shows how the *op* parameter of Algorithm 5 should be set according to the operation. In fact, it will be this parameter that allows different operations to be computed with the same algorithm. Beware that in some cases it may also have different values depending on the level i within the structure. This is necessary in order to compute complements and differences. We will denote $op[operation, j, i]$ the value in the Figure 3.12 in the case j for level i (with $j \in \{1, 2, 3, 4\}$ corresponding to the four cases presented earlier).

It is also important to note that an efficient implementation of the algorithm takes advantage of the fact that the lists of outgoing arcs are ordered. This is to make the comparison between the arcs faster.

Let us clarify a few aspects of the procedure presented. The $NewArcNode$ method deals with the creation of new nodes or arcs in the new MDD. This procedure checks, before creating a node, whether that node (β_1, β_2) does not already exist within the new structure. In that case, only the arc must be added.

In Algorithm 5, one can also see that this procedure is sometimes called on nodes *nil*. The node *nil* is a node with no outgoing arcs. It is useful in all those cases where one needs to add a node and an arc in the result of the operation but does not have the corresponding node in one or both of the initial MDD. This happens, for example, in the case of complements. The intuitive idea is that if we need to make a complement, we must add the arcs that did not exist. In other words, arcs for which one does not know the destination node. Another example is when one wants to compute the difference between two MDD. In this case, one wants to keep all the arcs that are part

Algorithm 5: ApplyMDD

Input : M_1, M_2 MDD, *op* array, and $operation$ to compute
Output: Resulting MDD M

```
1  L, E+ ← ∅;
2  root ← CreateNode(root(M1), root(M2));
3  L[1] ← root;
4  forall i ∈ {1, ..., r} do
5      L[i] ← ∅;
6      forall α ∈ L[i − 1] do
7          define α1, α2 such that α = (α1, α2);
8          V ← values(E+_α1 ∪ E+_α2);
9          forall val ∈ V do
10             if ∄(val, β1) ∈ E+_α1 then
11                 if ∄(val, β2) ∈ E+_α2 and op[operation, 1, i] then
12                     NewArcNode(L, i, α, val, nil, nil);
13                 if ∃(val, β2) ∈ E+_α2 and op[operation, 2, i] then
14                     NewArcNode(L, i, α, val, nil, β2);
15             else
16                 if ∄(val, β2) ∈ E+_α2 and op[operation, 3, i] then
17                     NewArcNode(L, i, α, val, β1, nil);
18                 if ∃(val, β2) ∈ E+_α2 and op[operation, 4, i] then
19                     NewArcNode(L, i, α, val, β1, β2);
20 Merge(L[r]);
21 return pReduce(L, E+);
```

of solutions not contained in another MDD. So this requires to be able to define nodes that are associated with one node in the first MDD and with none in the second.

Note that there may eventually be nodes without outgoing arcs or equivalent nodes. For example, in the case of a difference operation (*i.e.* $M_1 - M_2$), it may happen that by removing arcs from some nodes, they are now equivalent to others on the same level. It is for these reasons that the result must eventually be reduced. However, the reduction is important, in general, since the size of the structure resulting from the method may be larger than the size of the original structures.

The following examples illustrate how the Apply method works on two basic set operations.

Example 3.3.2 – Let us see how the Apply method computes the difference between the two MDD in Figure 3.13. At the first iteration, the values of the arcs coming out of the two roots are considered. Thus $V = \{0, 1, 2\}$. Both MDD have an outgoing arc with 0 from the roots. Therefore, we are in the case $op[-, 4, 1]$ and a new node (a, i) is created with an arc from the root having value 0. After that, only the first MDD has an arc of value 1 coming out. We are then in the case $op[-, 3, 1]$ where a new node (b, nil) with an arc of value 1 is created. The arc

with value 2 is only present in the second MDD. We are therefore in the case $op[-, 2, 1]$ and no node is created. This is intuitively correct since we are only interested in the tuples contained in the first structure and not in the second. The process is iterated considering nodes a and i. Arcs and nodes are created for values $V = \{0, 1, 2\}$, but in the case of 1 a node nil is used since there is no arc with label 1 leaving i. In the case b and nil, all arcs are conserved since they represent a part of the first structure that is not found in the second. The next level requires a different behaviour. In fact, the part of the structure from the root to the node (d, j) is preserved by the method in order to keep any solutions that differ only in the last label from those contained in the second MDD. The method thus retains only the arcs exiting d but not from j. In this case, there are none. This explains why (d, j) (highlighted in grey in the Figure 3.13) has no outgoing arcs, and why (c, nil) has arcs to *tt* labelled with 0 and 1. The node (d, j) will then be removed by the reduction method. The nodes contained in the final layer will be merged into a unique terminal node.

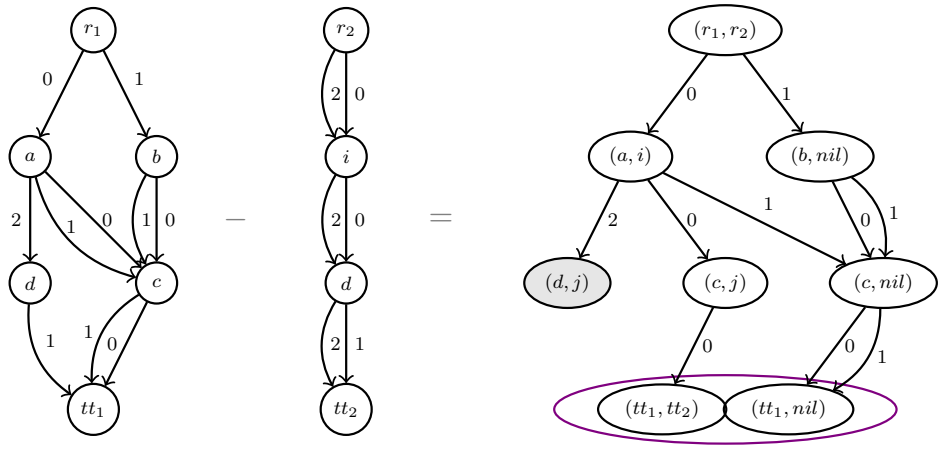


Figure 3.13 – Difference operation between two MDD.

As mentioned above, the result of the Apply method requires to be reduced. For example, the (d, j) node would then be eliminated.

Example 3.3.3 – Let us see how the Apply method computes the intersection between the two MDD in Figure 3.14. At the first iteration, $V = \{1, 2, 3, 4, 5\}$. Both MDD have an outgoing arc with 1 from the roots. Thus, we are in the case $op[\cap, 4, 1]$ and a new node (a, f) is created with an arc from the root having value 1. After that, only the first MDD has an arc of value 2 coming out. We are then in the case $op[\cap, 3, 1]$ where no node is created. This is intuitively correct since we are only interested in the tuples contained in both MDD. It is the same in the cases of arcs with value 3 and 5. If we look at the configuration of op in the intersection case, a node is created only when there are the two arcs in the initial MDD. Both MDD have an outgoing arc with 4 from the roots. Then, a new node (c, h) is created with an arc from the root having value 4.

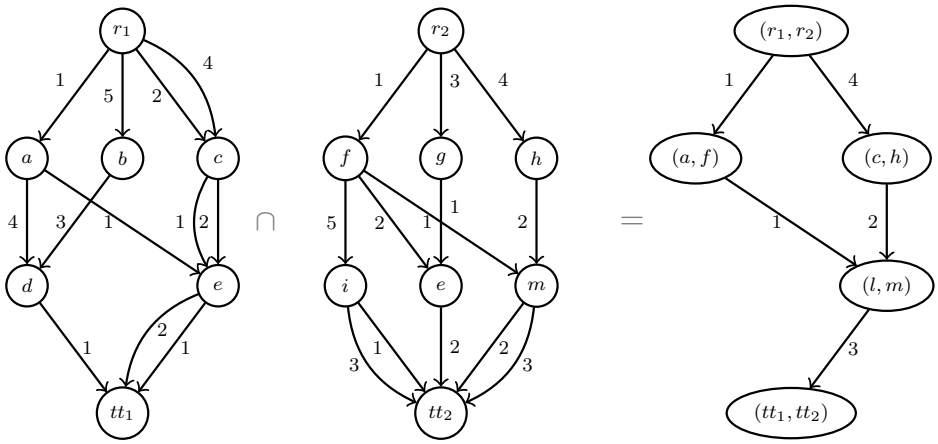


Figure 3.14 – Intersection between two MDD.

The process is iterated in the next layer. Considering the nodes a and f, the only common arc is the one with label 1. Then, the node (e, m) is generated. Considering the nodes c and h, the only common arc is the one with label 2. Then, the arc to (e, m) with latel 2 is added. Finally, the nodes e and m have an arc with label 2. Then, the node (tt_1, tt_2) and the corresponding arc are generated. In this case we obtain a result that is already reduced, but this is not always the case.

Contributions

CHAPTER 4

Equations and Abstractions over DDS

This chapter presents the polynomial equations with a constant right-hand term on $\mathcal{D}$ that allow us to investigate if a certain known dynamics is the result of several independent smaller systems interacting with each other. Given the complexity of the problem of finding a solution to equations over $\mathcal{D}$, the idea will be to study independently the characteristics of the solutions (number of states, the asymptotic or the transient behaviour). This will be made by a finite number of simpler equations (called abstractions) which reduce the solution space of the original equation over DDS one property at a time. Then, solutions of the original equation over DDS can be found in the intersection of the solution spaces of the abstractions.
In this chapter, the first abstraction to study the cardinality of the set of states of the variables is formally introduced. Then, a solution method based on Multi-valued Decision Diagrams is presented.

This chapter introduces the idea about abstractions and the first abstraction on the number of states of DDS presented in [Dennunzio et al. (2020), Dennunzio et al. (2022)].

4.1 Polynomial Equations over Dynamical Systems

This chapter focus on polynomial equations with a constant right-hand term on $\mathcal{D}$. Given the results presented in Chapter 1 regarding the equations on DDS, the aim is to introduce a new technique for finding solutions by exploring the solution space efficiently. As a starting point, the goal is to identify the solutions of all those equations with a multivariate polynomial that does not have products of several unknowns.

The aim is to express a certain DDS (contained in the constant term of the equation) as the sum and product of other, smaller systems. In other words, an attempt is made to provide a finer structure of the DDS in order to answer the following question, which is the central issue of this work:

Question 4.1.1. *Is the dynamics that we observe the action of a single basic system or does it come from the cooperation between two or more simpler systems?*

In this sense, a DDS can be viewed as a complex object with a certain intrinsic structure, *i.e.*, the feature of resulting from cooperating basic components. It is crucial, of course, to precise the meaning of the word "cooperation" in Question 4.1.1. In Chapter 1, two forms of cooperation have been devised. The additive form, denoted by $+$, in which two DDS with independent dynamics provide together an observed system, and the product one, denoted by $\cdot$, in which a system results from the joint parallel action of two DDS.

Now, consider the semiring $\mathcal{D}[X_1, \ldots, X_\nu]$ of polynomials over $\mathcal{D}$ in the variables $X_1, \ldots, X_\nu$, naturally induced by $\mathcal{D}$. Polynomial equations of the form (4.1) allow us to investigate whether a dynamics B can be recreated through the cooperation of simpler systems.

$$A_1 \cdot X_1^{w_1} + A_2 \cdot X_2^{w_2} + \ldots + A_m \cdot X_m^{w_m} = B \tag{4.1}$$

The coefficients A_z (with $z \in \{1, \ldots, m\}$) are hypothetical sub-dynamical systems that should cooperate to produce the dynamics B. Finding valid values for the unknown terms in (4.1) provides a finer structure for B.

From a purely theoretical point of view, these equations are a good starting point for studying how one can solve equations on DDS. In fact, if we consider polynomials admitting monomials with several variables (such as $A \cdot X^w \cdot X'^{w'}$ for example), it would be possible to rewrite the monomial as $A \cdot Y$, identify the solutions and then study $Y = X^w \cdot X'^{w'}$. Furthermore, in the case of polynomials $P(X_1, \ldots, X_\nu)$ that contain constant terms, we could look for the components of the DDS B that are isomorphic to them, remove these components from B, and consider the equation $P(X_1, \ldots, X_\nu)' = B'$ in which $P(X_1, \ldots, X_\nu)'$ and B' consists of $P(X_1, \ldots, X_\nu)$ and B after removal of the constants of the polynomial[1]. This gives the intuition of why we choose to start from equations of the form (4.1). However, they are also interesting for the opportunity to model hypotheses.

4.1.1 Hypothesis Validation

Given equations of form (4.1), we may investigate the relation between partial knowledge about a phenomenon (contained in the coefficients) and the observed dynamics in an experimental phase

[1] If there are not components in B that are isomorphic to the known terms, one can conclude that the equation has no solution.

(contained in the right-hand term). We can thus investigate whether the right-hand term of an equation is a more complex dynamics that is based on the collaboration of some known simpler dynamics. In other words, we use an equation to model hypotheses about a certain dynamics deduced from experimental data. Thus, the known term B is the dynamical system deduced from experiments and solutions of the equation bring further knowledge about the phenomenon.

Also for these reasons, to face Question 4.1.1 it is quite natural to consider multivariate monomials of the type $A \cdot X^w$ to represent a hypothesis about a finer structure of a given DDS. Here, considering $A \cdot X^w = B$ means that in the first place the observed DDS B is supposed to result from the joint parallel action of a known DDS, *i.e.*, the coefficient A, and w copies of some yet unknown DDS X. Following this new point of view, a polynomial $P(X_1, \ldots, X_\nu)$ is hence a more complex and "realistic" hypothesis on the observed DDS B. Then Question 4.1.1 can rephrased as:

Question 4.1.2. *Does the dynamics that we observe (from experimental data for instance) result from several independent smaller systems, each of them having its dynamics determined by the joint parallel action of a known part and an unknown part to be computed? In other words, does the equation $P(X_1, \ldots, X_\nu) = B$ have a solution? If any, which are its solutions?*

Therefore, from now on, we will consider equations composed of a polynomial part (which can model the hypothesis) and a constant part (which is the object of study). The goal will be to introduce an approach to enumerate the solutions of the equation (up to isomorphism). In fact, as anticipated in Section 1.1.2, here we are not interested in the precise nature of the different states that characterise a given DDS. Enumerating allows us not only to check whether an equation has a solution (*i.e.*, whether a hypothesis modelled in a polynomial is true) but also to obtain as much information as possible about the dynamics B.

4.2 Abstractions over DDS

Answering to Question 4.1.2 is always possible since the constant right-hand side bounds the space of admissible solutions. However, it might be highly non trivial to find the precise answer, as illustrated in Chapter 1 for distinct classes of polynomial equations. Then, given the complexity of the problem, the main idea is to study independently the different characteristics of a solution to provide a solid and effective analysis tool to be used in applications that answers Question 4.1.2. Three aspects of DDS can be investigated: the number of states, the asymptotic behaviour (*i.e.*, the cyclic part of the dynamics) and the transient behaviour (what happens before the system stabilises). For each of these characteristics, we want to reduce the solution space of the original equation, to have an idea of the features of the solutions.

The approach here is that for each of these characteristics, an *abstraction* is introduced, *i.e.* an equation corresponding to the original DDS equation but focusing only on a specific property. The term abstraction arises from the fact that one wants to focus only on a specific property and thus ignore other aspects of the dynamics. One therefore abstracts the dynamics of the coefficients and the known term of an equation to determine some information about the dynamics of the variables. This allows one to reduce the solution space as one is sure that the original solutions of the DDS equation respect the characteristics found through the abstractions. The idea of abstractions can be found in model checking [Clarke et al. (1994)], as well as in other areas of computer science [Knoblock (1990)].

In this work, three abstractions are introduced to solve equations of type (4.1): the *c-abstraction* (Section 4.3), the *a-abstraction* (Chapter 5) and the *t-abstraction* (Chapter 6). Each one provides specific properties of the solutions of an equation of type (4.1). Namely, the c-abstraction focus on the cardinalities of the set of states of the variables. Then, it allows to reduce the solution space to all those DDS that satisfy the condition on the cardinality of the state set induced by the original equation. The a-abstraction provides the cyclic (or asymptotic) behaviour of the dynamical systems $X_1, \ldots, X_\nu$. Thus, it allows to reduce the solution space to all those DDS having a set of attractors that satisfies the equation. Finally, the t-abstraction provides the transient behaviours in the variables. We stress that the set of the solutions of the equation (4.1) just turns out be the intersection of the sets of DDS selected by these three abstractions. For this reason and given the fact that the goal is the enumeration of solutions of an Equation (4.1) (according to Question 4.1.2), the enumeration of all solutions of each abstraction is needed to reach the goal.

However, it must be emphasised that for an efficient search, it is best to solve the abstractions in the order in which they are presented to reduce the search space for each abstraction. In this way, for each variable, one first identifies its feasible number of states, then its cyclic behaviour and finally reconstructs its transient behaviour. This is especially important given the fact that the cost in terms of time and space increases from the first abstraction to the last one.

4.3 The c-abstraction

Given a polynomial equation over DDS, a natural abstraction concerns the number of states of the systems which are involved in it. Performing such an abstraction leads to a new equation in which the coefficients of the polynomial, the variables, and the constant term become those natural numbers corresponding to the cardinalities of the sets of states of the DDS involved in the original equation. This is the simplest of the three abstractions but allows us to introduce ideas that will be useful later on.

Definition 4.3.1 (c-abstraction)**.** The *c-abstraction* of a DDS S is the cardinality of its set of states. With an abuse of notation, the c-abstraction of S is denoted by $|S|$.

The following lemma links c-abstractions with the operations over DDS described in Chapter 1 (see Section 1.1.3).

Lemma 4.3.1. *For any pair of DDS S_1 and S_2, it holds that $|S_1 + S_2| = |S_1| + |S_2|$ and $|S_1 \cdot S_2| = |S_1| \cdot |S_2|$.*

The lemma is derived directly from the definition of sum and product over DDS (Definitions 1.1.2 and 1.1.3). The first operation requires a disjoint union of the sets of states, while the second requires a Cartesian product.

Using the notion of c-abstraction and the previous lemma, Equation (4.1) turns into the following *c-abstraction equation*

$$|A_1| \cdot |X_1|^{w_1} + |A_2| \cdot |X_2|^{w_2} + \ldots + |A_m| \cdot |X_m|^{w_m} = |B| \ . \tag{4.2}$$

Recall that, to reach the overall goal, we need to enumerate all solutions of Equation (4.2) to identify all cardinalities of the set of states of the variables in the original equation.

The following example illustrates the concepts just introduced.

Example 4.3.1 – Consider the equation on DDS presented in Figure 4.1. Its c-abstraction is $5 \cdot |X_1|^2 + 4 \cdot |X_2| = 293$ since $|A_1| = 5$, $|A_2| = 4$, and $|B| = 293$.

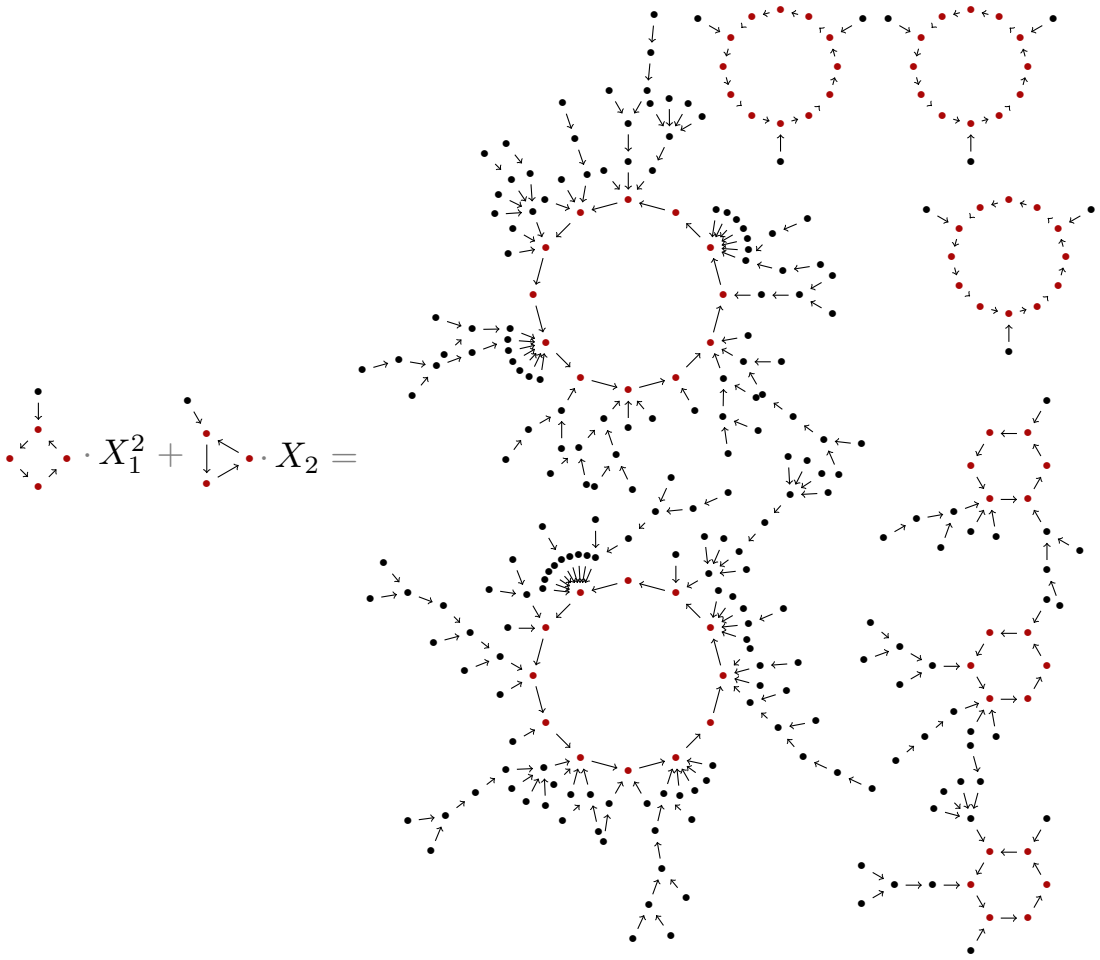


Figure 4.1 – An example of Equation (4.1). The coefficients A_1, A_2 and the know term B are depicted by their dynamics graphs.

Let us proceed as follows. First of all, we present this problem from a combinatorial point of view. Then, we will provide an algorithmic approach (based on Multi-valued Decision Diagrams) allowing the enumeration of the solutions of an Equation (4.2) in an efficient way.

Let us consider the case with just one monomial (*i.e.*, $m = 1$) corresponding to a simpler equation of the form $|A| \cdot |X|^w = |B|$ (*basic case*). It is clear that:

- if $w = 0$, then X^w consists of a system with one cycle of length one[2] (a fixed point). Then, $|X|^w = 1$ and $|A| = |B|$, while the equation is impossible, otherwise;
- if $w \neq 0$, the equation admits a (unique) solution iff $\sqrt[w]{|B|/|A|}$ is an integer number.

[2]For w equal to 0, we assume that S^0 is equal to the single fixed point system, the neutral element for the product operation.

Given now an equation with $m > 1$ monomials, it is clear that each state of the DDS B must come from one of the monomials. Remind that the sums between the different monomials correspond to disjoint unions between the sets of states obtained by the product operations. Therefore, we have to consider all the ways of arranging $|B|$ states among m monomials. The number of such arrangements is $\binom{|B|+m-1}{m-1}$ and the Stars and Bars (SB) is a graphical interpretation of this binomial value [Jongsma (2019)].

Example 4.3.2 – The Stars and Bars concept takes its name from the graphical representation. For example, the divisions of 4 elements in 3 boxes are $\binom{4+3-1}{3-1} = 15$.

$$
\begin{array}{lll}
||\star\star\star\star & |\star|\star\star\star & |\star\star|\star\star \\
|\star\star\star|\star & |\star\star\star\star| & \star|\star\star\star| \\
\star\star|\star\star| & \star\star\star|\star| & \star\star\star\star|| \\
\star\star\star||\star & \star\star||\star\star & \star||\star\star\star \\
\star|\star\star|\star & \star\star|\star|\star & \star|\star|\star\star
\end{array}
$$

Let B_z be the number of states generated by the z-th monomial. Any arrangement consisting of $B_1, \ldots, B_m$ states (*i.e.*, any weak composition[3]) gives rise to a system as follows

$$
\begin{cases}
|A_1| \cdot |X_1|^{w_1} & = B_1 \\
|A_2| \cdot |X_2|^{w_2} & = B_2 \\
& \vdots \\
|A_m| \cdot |X_m|^{w_m} & = B_m
\end{cases}, \tag{4.3}
$$

where $\sum_{z=1}^{m} B_z = |B|$ and each equation falls into the basic case.

To enumerate the solutions of the abstraction, we need an efficient method that solves all feasible Systems (4.3) (*i.e.*, those systems admitting a solution). Since any System (4.3) consists of equations that are basic cases, and establishing whether each of them admits a solution is easy, the method can be designed in such a way that the solution space to be explored is reduced.

Due to the combinatorial nature of the problem, we will use Multi-valued Decision Diagrams (MDD) (presented in Chapter 3) to enumerate the solutions of equations over c-abstractions.

Consider an Equation (4.2) with m monomials and a number ν of distinct variables. We associate such an equation with an MDD $(N, E, \ell ab, val)$ in which there are $\nu + 1$ layers (one for each variable and one final layer for the *tt* node). Recall that, as explained in Chapter 3, the structure is built according to a certain order L_o of variables. Let us start to define the structure.

The nodes of the structure are $N = N_1 + (\sum_{i=2}^{\nu} N_i) + N_{\nu+1}$,[4] where $N_1 = \{root\}$, $N_i = \bigcup_{j=0}^{|B|} N_{i,j}$ and $N_{\nu+1} = \{tt\}$. Hence, each N_i is a level of the MDD.

[3] A weak composition of an integer $|B|$ is a way of writing it as the sum of a sequence of m integers $B_1, \ldots, B_m$ greater than or equal to 0.

[4] It is important to note that to define the set of nodes in the structure, we used the sum and not the union. When working with multisets, the union preserves the elements in the maximum multiplicity between the two sets, whereas the sum adds the multiplicities.

For any node $\alpha \in N$, let $val(\alpha) = 0$ if $\alpha = root$, $val(\alpha) = |B|$ if $\alpha = tt$, and $val(\alpha) = j$ if $\alpha = N_{i,j}$. In fact, the value $val(\alpha)$ associated with a node α represents the amount of states obtained from a partial set of variable assignments, *i.e.*, a set of assignments involving the variables corresponding to the layers up to the one of α. Each of them corresponds to a path from *root* to α. Indeed, the c-abstraction equation admits no solution if there is no path from *root* to *tt* on the associated MDD.
The set of edges is

$$E = \{(\alpha, \beta) \in N \times N \mid \exists i \in \{1, \dots, \nu\} \text{ s.t. } \alpha \in N_i, \beta \in N_{i+1}, \\ val(\beta) = val(\alpha) + \sum_{z=1}^{m} var(i,z) \cdot |A_z| \cdot d^{w_z}, \text{ and } d \in \{0, \dots, |B|\}\}$$

where $var(i,z) = 1$ if X_z is the i-th variable in the order L_o (*i.e.* if it corresponds to the i-th layer of the structure), 0 otherwise. Lastly, the labelling function $\ell ab\colon E \to \{0, \dots, |B|\}$ is defined as $\ell ab((\alpha, \beta)) = d$ where $val(\beta) = val(\alpha) + \sum_{z=1}^{m} var(i,z) \cdot |A_z| \cdot d^{w_z}$.

This definition is formally correct and allows us to describe the set of nodes, arcs as well as the ℓab and val functions. However, it leads to the potential creation of several isolated nodes and unfeasible variables assignments which would be certainly deleted later in the reduction phase (see Figure 4.2). Hence, we introduce a second definition of the structure by induction (*i.e.* level by level) which is also more effective since it leads to avoiding unfeasible parts of the solutions space considering the previous variable assignments (the algorithmic approach to building the structure will be then based on it).

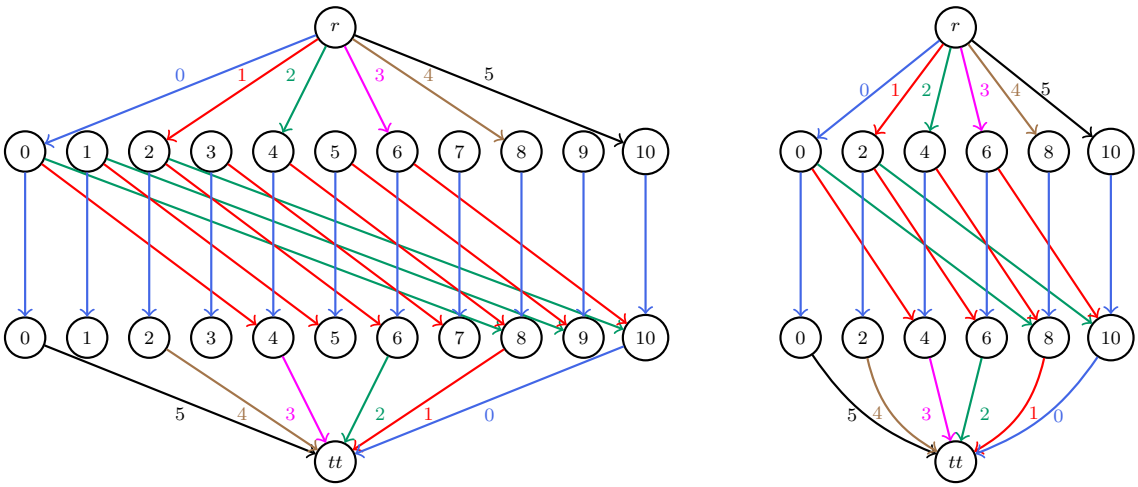


Figure 4.2 – Let us consider the equation $2 \cdot |X_1| + 4 \cdot |X_2| + 2 \cdot |X_3| = 10$. On the left, we see the (unreduced) MDD constructed according to the first definition. The (unreduced) MDD built according to the second definition is on the right. Both MDD are constructed by considering the variables in order of appearance in the equation, hence $|X_1|$, $|X_2|$ and then $|X_3|$. One can see how the former explores non-feasible parts of the solution space that the latter does not. For the sake of clarity, the labels of the arcs in the second layer are expressed through colours (the colour-value correspondence is expressed in the first layer).

The first level of the structure contains the *root* (*i.e.* $N_1 = \{root\}$) with $val(root) = 0$. For any $d \in \{0, \ldots, |B|\}$ such that

$$\sum_{z=1}^{m} var(1, z) \cdot |A_z| \cdot d^{w_z} \leq |B|$$

we create a node in the following level. Hence, we have $\beta \in N_2$ with $val(\beta) = \sum_{z=1}^{m} var(1, z) \cdot |A_z| \cdot d^{w_z}$, and $(root, \beta) \in E$ with $\ell ab((root, \beta)) = d$. The outgoing labels represent the values (*i.e.* the number of states) for the variable corresponding to the first layer.

Now, let us consider a level i (with $i \in \{2, \ldots, \nu - 1\}$) already defined. For any $\alpha \in N_i$ and any $d \in \mathbb{N}$ such that

$$val(\alpha) + \sum_{z=1}^{m} var(i, z) \cdot |A_z| \cdot d^{w_z} \leq |B|$$

we potentially create a node β (with $val(\beta) = \sum_{z=1}^{m} var(1, z) \cdot |A_z| \cdot d^{w_z}$) in the following level. In fact, if a node β' such that $val(\beta) = val(\beta')$ exists, just a new edge is created. Indeed, it represents the same amount of states obtained from a partial set of variable assignments. More formally, if $\beta \notin N_{i+1}$ and $\beta = \sum_{z=1}^{m} var(i, z) \cdot |A_z| \cdot d^{w_z} + val(\alpha)$, then $N_{i+1} = N_{i+1} \cup \{\beta\}$ and $(\alpha, \beta) \in E$ with $\ell ab((\alpha, \beta)) = d$. However, if $\beta \in N_{i+1}$, we just add $(\alpha, \beta) \in E$ with $\ell ab((\alpha, \beta)) = d$.

To conclude the definition of the structure, let us consider the level ν. For any $\alpha \in N_\nu$ and any $d \in \mathbb{N}$ such that

$$val(\alpha) + \sum_{z=1}^{m} var(\nu, z) \cdot |A_z| \cdot d^{w_z} = |B|$$

we create an arc between α and *tt*. Then, we have $(\alpha, tt) \in E$ with $\ell ab((\alpha, tt)) = d$.

According to this second definition, the nodes of the structure (at the end) are the multiset $N = \sum_{i \in \{1, \ldots, \nu+1\}} N_i$ where $N_1 = \{root\}$, $N_i \subseteq \{0, ..., |B|\}$ with $i \in \{2, \ldots, \nu\}$, and $N_{\nu+1} = \{tt\}$.

Finally, the MDD is reduced by performing a pReduction (see Section 3.2) to merge equivalent nodes and delete all nodes (and the corresponding edges) which are not on a path from *root* to *tt*.

An implementation of this construction is provided by Algorithm 6. It takes in input all the necessary information about the equation: the coefficients, variables and exponents of the monomials, as well as the order chosen for the variables in order to construct the structure, and the number of states of B.

It is important to note that we did not require that the c-abstraction equation is in simplified form. Indeed, the construction process presented is invariant with respect to whether the equation is simplified or not.

The algorithm exploits the procedure $NewArcNode$ which implements the allocation technique explained before. Indeed, a new node β is created if and only if another node with the same value does not already exist, otherwise, just a new arc is created.

It is important to emphasise that as indicated in the pseudocode (lines 17 and 18) we consider the values of d in ascending order and we stop at the first value that results in the creation of more than $|B|$ states.

Let us illustrates this MDD-based approach with an example (see Example 4.3.3).

Algorithm 6: c-abstraction

Input : L_c list of coefficients, L_v list of variables, L_e list of exponents, L_o order of variables, and $rterm$ corresponding to $|B|$

Output: M MDD with the solutions

1 $M[1] \leftarrow \{root\}$; ▷ M[i] are the nodes of M in level i
2 $val(root) \leftarrow 0$;
3 $d \leftarrow 0$; ▷ Current value to test
4 **forall** $i \in \{1, \ldots, \nu\}$ **do**
5 **forall** $\alpha \in M[i]$ **do**
6 $creation \leftarrow true$;
7 **while** $creation$ **do**
8 $\beta \leftarrow \left(\sum_{z=1}^{m} var(L_o[i], L_v[z]) \cdot L_c[z] \cdot d^{\,L_o[z]}\right) + val(\alpha)$;
9 **if** $i \neq \nu$ *and* $\beta \leq rterm$ **then**
10 NewArcNode$(M, i+1, \alpha, d, \beta)$;
11 $d \leftarrow d + 1$;
12 **else**
13 **if** $i == \nu$ *and* $\beta == rterm$ **then**
14 NewArc$(M, \nu, \alpha, d, \beta)$;
15 $d \leftarrow d + 1$;
16 **else**
17 $creation \leftarrow false$;
18 $d \leftarrow 0$;
19 **return** pReduce(M);

Example 4.3.3 – Consider the following equation:

$$2 \cdot |X_3| + 5 \cdot |X_1|^2 + 4 \cdot |X_2| + 4 \cdot |X_1|^4 + 4 \cdot |X_3|^2 = 593.$$

By using SB, one can check that there are $\binom{593+5-1}{5-1} = 5.239776465 \times 10^9$ ways of arranging $|B| = 593$ states among $m = 5$ monomials. However, not all of them are give rise to solutions. The MDD produced by Algorithm 6 is presented in Figure 4.3. The first level of the structure represents the feasible values of the variable $|X_1|$ associated to the inequality $\sum_{z=1}^{5} var(1,1) \cdot |A_z| \cdot |X_z|^{w_z} = 5 \cdot |X_1|^2 + 4 \cdot |X_1|^4 \leq 593$. For this variable, different values of d are tested.

$$\begin{aligned}
d = 0, &\quad 5 \cdot 0^2 + 4 \cdot 0^4 \leq 593,\\
d = 1, &\quad 5 \cdot 1^2 + 4 \cdot 1^4 = 9 \leq 593,\\
d = 2, &\quad 5 \cdot 2^2 + 4 \cdot 2^4 = 84 \leq 593,\\
d = 3, &\quad 5 \cdot 3^2 + 4 \cdot 3^4 = 369 \leq 593,\\
d = 4, &\quad 5 \cdot 4^2 + 4 \cdot 4^4 = 1104 > 593.
\end{aligned}$$

In the second layer, the red edges along with the corresponding labels represent the values for $|X_2|$, when $|X_1| = 1$. The blue ones are the assignments for $|X_2|$, when $|X_1| = 3$. The last layer represents the feasible assignments of $|X_3|$.

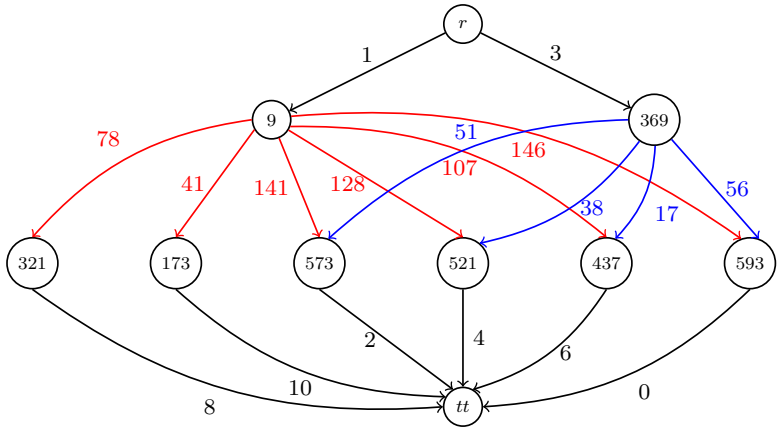


Figure 4.3 – The reduced MDD representing all the solutions of $2 \cdot |X_3| + 5 \cdot |X_1|^2 + 4 \cdot |X_2| + 4 \cdot |X_1|^4 + 4 \cdot |X_3|^2 = 593$. There are $\nu = 3$ variables, which are represented in the structure in the following order: $|X_1|$, $|X_2|$, and $|X_3|$. Here, the val of the nodes is the value reported inside them.

We stress that the MDD allows the exploration of the solution space of the equation in a efficient way. In fact, at each level only a part of the values for a variable are considered depending on the feasible assignments of the variables of the previous levels. Moreover, the MDD can gain up to an exponential factor in representation space through the reduction process (as explained in Chapter 3).

The worst case time complexity is $O(|B|^2\nu)$, for the construction part, and $O(|B|\nu + \delta + D)$, for the reduction one, where $\delta = \sum_{i=1}^{\nu} |D_i| \cdot |B|$ and D is the bigger domain for a variable which can be at most $\{0, \ldots, |B|\}$ (see Chapter 3). The worst case space complexity is $O(|B|\nu + \delta)$, in terms of number of nodes and edges. The pReduction reduces the total number of edges to $\delta' \ll \delta$ and the bound of the number of nodes of any level to $\mu \leq |B|$, giving rise to a lower complexity $O(\mu\nu + \delta')$. Actually, this bound is never reached in our experiments. As an illustrative case, consider Example 4.3.3. The MDD could have up to 1188 nodes and 352835 edges, but its reduced version has only 10 nodes and 18 edges (see Figure 4.3).

Example 4.3.4 – For the sake of completeness, consider the equation of the Example 4.3.1. Figure 4.4 depicts the MDD returned by Algorithm 6. There are $\nu = 2$ variables, and hence 2 layers. The first level of the structure represents the assignments of the variable $|X_1|$ and the second one $|X_2|$. Then, the c-abstraction provides the following solutions:

$$|X_1| = 7, \quad |X_2| = 12$$

$$|X_1| = 5, \quad |X_2| = 42$$

$$|X_1| = 3, \quad |X_2| = 62$$

$$|X_1| = 1, \quad |X_2| = 72$$

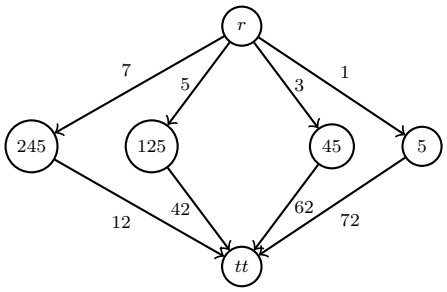


Figure 4.4 – The reduced MDD representing all the solutions of $5 \cdot |X_1|^2 + 4 \cdot |X_2| = 293$.

In conclusion, the approach just presented permits the enumeration of all the solutions of a c-abstraction equation and, hence, to understand the feasible cardinalities of the set of states for the variables in an Equation (4.1).

CHAPTER 5

Asymptotic behaviour of DDS

This chapter introduces the a-abstraction which aims to enumerate all possible cyclic behaviours of the variables of multi-variate polynomial equations with constant right-hand term on $\mathcal{D}$. This is done by introducing a specific notation to express any DDS as a function of its periodic part, studying how these parts of the systems are involved in sums and products, but also by proving that we can relate back to a simpler problem. In fact, it is shown that one can see any equation as a finite set of systems made by a finite number of simpler equations, called basic equations. *This result is achieved thanks to some algebraic transformations, called contraction steps. The complexity of deciding whether a basic equation has a solution, as well as the complexity of listing all its solutions are also investigated. The chapter provides two algorithmic techniques to enumerate the solutions of basic equations and, starting from them, it develops a complete MDD-based pipeline to enumerate the solutions of any equation are introduced. Key points are: the use of MDD to enumerate the solutions of the basic equations as it allows them to be combined efficiently to find the solutions of the starting equation, and the introduction of a technique to calculate roots on asymptotic behaviour of DDS.*

This chapter focus on the a-abstraction, the colored-tree [Dennunzio et al. (2020)] and the MDD-based approach [Formenti et al. (2021)], and the complete pipeline proposed in [Dennunzio et al. (2022)].

5.1 The a-abstraction

In this chapter we deal with a further abstraction, namely, the asymptotic one, describing the long-term behaviour of a DDS, *i.e.*, its periodic behaviour. In fact, we introduce the version of Equation (4.1), called *a-abstraction equation*, obtained from the original one by removing all transients states and their edges in coefficients, variables and constant right-hand term. This chapter develops an algorithmic approach to enumerate the solutions of this new abstraction. We thus obtain a method to validate hypotheses about the asymptotic behaviour of a phenomenon modelled through a DDS. Before going into the details of the a-abstraction, let us provide the notation that we will use to represent the cyclic behaviour of a DDS.

Notation. *In the sequel, for any pair of positive integers n and p, C_p^n will stand for the union of any n disjoint cycles of length p of a DDS S. To stress that we are dealing with sets consisting of union of disjoint cycles, each of them identifying a subsystem of S, the operations of disjoint union and product of two of such sets, or, by identification, the sum and product of the corresponding dynamical (sub)systems, will be denoted by $\oplus$ and $\odot$ instead of $+$ and $\cdot$, respectively. According to this notation, it is clear that $C_p^{n_1} \oplus C_p^{n_2} = C_p^{n_1+n_2}$ for any pair of positive integers n_1, n_2. Moreover, kC_p^n is a shortcut for C_p^{kn} for any positive integer k and n.*

Definition 5.1.1 (a-abstraction)**.** The *a-abstraction* of a DDS S, denoted by $\mathring{S}$, is the dynamical subsystem of S induced by the set $\mathcal{P}$ of all its periodic points, or, by identification, the set $\mathcal{P}$ itself.

Then, the a-abstraction of a DDS S can be written as

$$\mathring{S} = \bigoplus_{i=1}^{l} C_{p_i}^{n_i} \ ,$$

for some positive naturals l, $n_1, \ldots, n_l$, and pairwise distinct positive naturals $p_1, \ldots, p_l$, where, for each $i \in \{1, ..., l\}$, n_i is the number of disjoint cycles of length p_i (see Figure 5.1 for an illustrative example).

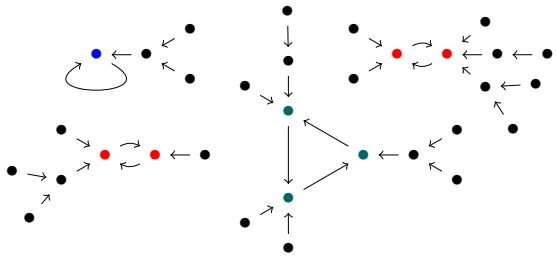


Figure 5.1 – A DDS S with four cycles. Its a-abstraction $\mathring{S}$ is $(C_1^1 \oplus C_2^2 \oplus C_3^1)$ in our notation.

It immediately follows from the previous definition that the a-abstraction of the sum, resp., the product, of two DDS, is the sum, resp., the product of the a-abstractions of the two DDS. In fact, we know that the periodic points of a DDS, resulting from a product operation, are all those points arising from a Cartesian product between two periodic points of the DDS involved in the product.

Let us formalise these operations between a-abstractions. Let us consider $p_{S,j}$ the j^{th} period into the system $\mathring{S}$, and $n_{S,j}$ the number of cycles of length $p_{S,j}$.

Proposition 5.1.1. *Given $S_1 = (\mathcal{X}_1, f_1)$ (resp., $S_2 = (\mathcal{X}_2, f_2)$), let $\mathring{S}_1 = \bigoplus_{i=1}^{l_1} C_{p_{1,i}}^{n_{1,i}}$ (resp., $\mathring{S}_2 = \bigoplus_{j=1}^{l_2} C_{p_{2,j}}^{n_{2,j}}$). The following equivalence holds for $\mathring{S}_1 \oplus \mathring{S}_2$*

$$\bigoplus_{i=1}^{l_1} C_{p_{1,i}}^{n_{1,i}} \oplus \bigoplus_{j=1}^{l_2} C_{p_{2,j}}^{n_{2,j}} = C_{p_{1,1}}^{n_{1,1}} \oplus \ldots \oplus C_{p_{1,l_1}}^{n_{1,l_1}} \oplus C_{p_{2,1}}^{n_{2,1}} \oplus \ldots \oplus C_{p_{2,l_2}}^{n_{2,l_2}}.$$

By Definition 1.1.2, $(\mathcal{X}_1, f_1) + (\mathcal{X}_2, f_2) = (\mathcal{X}_1 \sqcup \mathcal{X}_2, f_1 \sqcup f_2)$ and each cycle of $(\mathcal{X}_1 \sqcup \mathcal{X}_2, f_1 \sqcup f_2)$ comes from either S_1 or S_2. The sum of the a-abstractions of $(\mathcal{X}_1, f_1)$ and $(\mathcal{X}_2, f_2)$ is then by definition the a-abstraction of $(\mathcal{X}_1 \sqcup \mathcal{X}_2, f_1 \sqcup f_2)$.

Proposition 5.1.2. *Given $S_1 = (\mathcal{X}_1, f_1)$ (resp., $S_2 = (\mathcal{X}_2, f_2)$), let $\mathring{S}_1 = \bigoplus_{i=1}^{l_1} C_{p_{1,i}}^{n_{1,i}}$ (resp., $\mathring{S}_2 = \bigoplus_{j=1}^{l_2} C_{p_{2,j}}^{n_{2,j}}$). Then, $\mathring{S}_1 \odot \mathring{S}_2$ is $\bigoplus_{i=1}^{l_1} C_{p_{1,i}}^{n_{1,i}} \odot \bigoplus_{j=1}^{l_2} C_{p_{2,j}}^{n_{2,j}} = \bigoplus_{i=1}^{l_1} \bigoplus_{j=1}^{l_2} C_{p_{1,i}}^{n_{1,i}} \odot C_{p_{2,j}}^{n_{2,j}}$.*

As stated in Chapter 1, the product distributes over the sum. However, it is now necessary to be able to compute $C_{p_{1,i}}^{n_{1,i}} \odot C_{p_{2,j}}^{n_{2,j}}$ (see Equation (5.2)). Let us consider directly the case in which we need to multiply more than two sets of cycles.

Proposition 5.1.3. *For any natural $z > 1$ and any positive naturals $n_1, \ldots, n_z$, $p_1, \ldots, p_z$, it holds that*

$$\bigodot_{i=1}^{z} C_{p_i}^{n_i} = C_{\lambda_z}^{\prod_{i=1}^{z} n_i \cdot \prod_{i=2}^{z} \gcd(\lambda_{i-1}, p_i)}, \tag{5.1}$$

where $\lambda_i = \mathrm{lcm}(p_1, \ldots, p_i)$.

Proof. We proceed by finite induction over z. First of all, we prove that the statement is true for $z = 2$, *i.e.*,

$$C_{p_1}^{n_1} \odot C_{p_2}^{n_2} = C_{\lambda_2}^{n_1 \cdot n_2 \cdot \gcd(p_1, p_2)}. \tag{5.2}$$

Let us consider the case $n_1 = n_2 = 1$. Since $C_{p_1}^1$ and $C_{p_2}^1$ can be viewed as finite cyclic groups of order p_1 and p_2, respectively, each element of the product of such cyclic groups has order $\mathrm{lcm}(p_1, p_2)$ or, in other words, each element of $C_{p_1}^1 \odot C_{p_2}^1$ belongs to some cycle of length λ_2. So, $C_{p_1}^1 \odot C_{p_2}^1$ just consists of $(p_1 \cdot p_2)/\lambda_2 = \gcd(p_1, p_2)$ cycles, all of length λ_2, and therefore

$$C_{p_1}^1 \odot C_{p_2}^1 = C_{\lambda_2}^{\gcd(p_1, p_2)}.$$

In the case $n_1 \neq 1$ or $n_2 \neq 1$, since the product is distributive over the sum, we get

$$C_{p_1}^{n_1} \odot C_{p_2}^{n_2} = \bigoplus_{i=1}^{n_1} C_{p_1}^1 \odot \bigoplus_{j=1}^{n_2} C_{p_2}^1 = \bigoplus_{i=1}^{n_1} \bigoplus_{j=1}^{n_2} (C_{p_1}^1 \odot C_{p_2}^1) = \bigoplus_{i=1}^{n_1} \bigoplus_{j=1}^{n_2} C_{\lambda_2}^{\gcd(p_1, p_2)} = C_{\lambda_2}^{n_1 \cdot n_2 \cdot \gcd(p_1, p_2)}.$$

Assume now that the equality holds for any $z > 2$. Then, we get

$$\bigodot_{i=1}^{z+1} C_{p_i}^{n_i} = C_{\lambda_z}^{\prod_{i=1}^{z} n_i \cdot \prod_{i=2}^{z} \gcd(\lambda_{i-1}, p_i)} \odot C_{p_{z+1}}^{n_{z+1}} =$$

$$= C_{\mathrm{lcm}(\lambda_z, p_{z+1})}^{\prod_{i=1}^{z+1} n_i \cdot \prod_{i=2}^{z} \gcd(\lambda_{i-1}, p_i) \gcd(\lambda_z, p_{z+1})} = C_{\mathrm{lcm}(\lambda_z, p_{z+1})}^{\prod_{i=1}^{z+1} n_i \cdot \prod_{i=2}^{z+1} \gcd(\lambda_{i-1}, p_i)}.$$

Therefore, the equality also holds for $z + 1$ and this concludes the proof. □

Remark 5.1.1. For any natural $z > 1$ and any positive naturals $n_1, \ldots, n_z$, $p_1, \ldots, p_z$, it holds that

$$\bigodot_{i=1}^{z} C_{p_i}^{n_i} = C_{\lambda_z}^{\frac{1}{\lambda_z} \cdot \prod_{i=1}^{z}(p_i n_i)} \quad , \tag{5.3}$$

where $\lambda_i = \mathrm{lcm}(p_1, \ldots, p_i)$.

Proof. According to Proposition 5.1.3,

$$\bigodot_{i=1}^{z} C_{p_i}^{n_i} = C_{\lambda_z}^{\prod_{i=1}^{z} n_i \cdot \prod_{i=2}^{z} \gcd(\lambda_{i-1}, p_i)}.$$

However, one find that

$$\frac{1}{\lambda_z} \cdot \prod_{i=1}^{z}(p_i n_i) = \prod_{i=1}^{z} n_i \cdot \prod_{i=2}^{z} \gcd(\lambda_{i-1}, p_i).$$

In fact, we get

$$\frac{\prod_{i=1}^{z} p_i}{\lambda_z} = \gcd(p_1, p_2) \cdot \gcd(\lambda_2, p_3) \cdot \gcd(\lambda_3, p_4) \cdot \ldots \cdot \gcd(\lambda_{z-1}, p_z)$$

that can be rewritten as follows

$$\frac{\prod_{i=1}^{z} p_i}{\lambda_z} = \gcd(p_1, p_2) \cdot \frac{\lambda_2 \cdot p_3}{\lambda_3} \cdot \frac{\lambda_3 \cdot p_4}{\lambda_4} \cdot \ldots \cdot \frac{\lambda_{z-1} \cdot p_z}{\lambda_z}.$$

Indeed,

$$\prod_{i=1}^{z} p_i = \frac{p_1 \cdot p_2}{\lambda_2} \cdot \lambda_2 \cdot p_3 \cdot p_4 \cdot \ldots \cdot p_z.$$

□

We now consider the w-th power of a set of cycles of a certain lengths and the w-th power of a generic set of cycles. These propositions will be fundamental to introduce the approach that will permit us to calculate the $\mathring{X}$-value of a $\mathring{X}^w$ that appears in the equation. Before proceeding, for any DDS S, we naturally define $\mathring{S}^0$ as C_1^1, *i.e.*, the neutral element of the product operation.

Lemma 5.1.4. *For any natural numbers $w \geq 1$, $n > 0$, and $p \geq 1$, it holds that:*

$$(C_p^n)^w = C_p^{p^{w-1} n^w}.$$

Proof. Given $w = 1$, $(C_p^n)^1 = C_p^{n^1}$. Considering the lemma true for a generic $w > 1$, we show that it is true also in the case of $w + 1$. Indeed, $(C_p^n)^{w+1} = (C_p^n)^w \odot C_p^n = C_p^{p^{w-1} n^w} \odot C_p^n$, and according to Equation (5.2), $C_p^{p^{w-1} n^w} \odot C_p^n = C_p^{p^{w-1} n^w \cdot n \cdot p} = C_p^{p^w n^{w+1}}$. □

Proposition 5.1.5. *For any positive naturals $l > 1$, $w > 1$, $n_1, \ldots, n_l$, and $p_1, \ldots, p_l$, it holds that*

$$\left(\bigoplus_{i=1}^{l} C_{p_i}^{n_i} \right)^w = \bigoplus_{\substack{k_1 + \ldots + k_l = w \\ 0 \leq k_1, \ldots, k_l \leq w}} \binom{w}{k_1, \ldots, k_l} C_{\lambda_l^*}^{\prod_{1 \leq i \leq l : k_i \neq 0} p_i^{k_i - 1} n_i^{k_i} \cdot \prod_{2 \leq i \leq l : k_i \neq 0} \gcd(\lambda_{i-1}^*, p_i)}$$

where, for any tuple $k_1, \ldots, k_i$, λ_i^* is the lcm *of those p_j (with $j \in \{1, \ldots, i\}$) such that $k_j \neq 0$ (while $\lambda_i^* = 1$ iff all $k_j = 0$).*

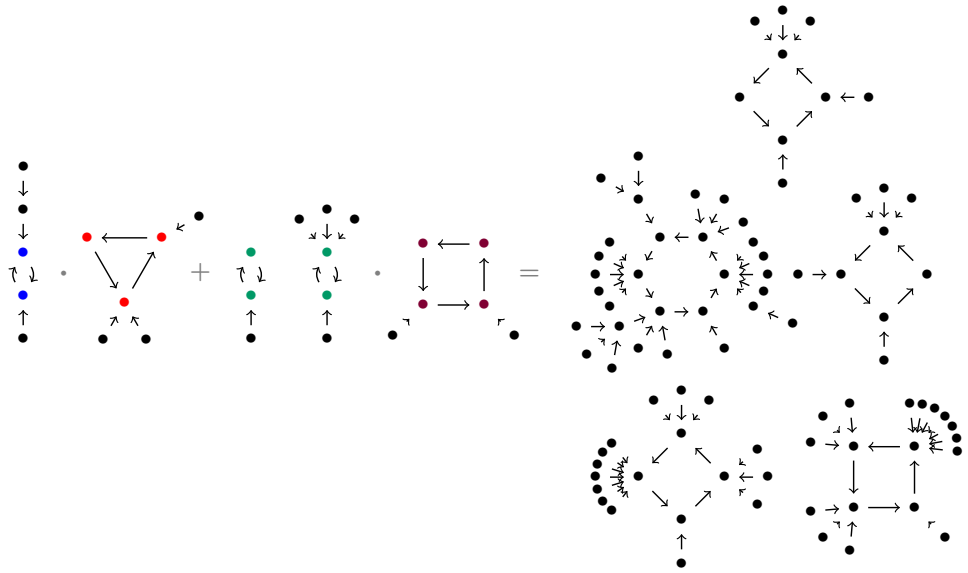


Figure 5.2 – Operations of sums and products between DDS. Considering the a-abstractions of the systems involved, one can see that the operations between a-abstractions gives the abstraction of the result above. In fact, $C_2^1 \odot C_3^1 \oplus C_2^2 \odot C_4^1 = C_6^1 \oplus C_4^4$ (the colours here refer to the colours of the periodic points of the systems above).

Proof. By Proposition 1.1.1, Proposition 5.1.3, and Lemma 5.1.4, we get

$$\begin{aligned}
\left(\bigoplus_{i=1}^{l} C_{p_i}^{n_i}\right)^w &= \bigoplus_{\substack{k_1+\ldots+k_l=w \\ 0\le k_1,\ldots,k_l\le w}} \binom{w}{k_1,\ldots,k_l} \bigodot_{i=1}^{l} (C_{p_i}^{n_i})^{k_i} \\
&= \bigoplus_{\substack{k_1+\ldots+k_l=w \\ 0\le k_1,\ldots,k_l\le w}} \binom{w}{k_1,\ldots,k_l} \bigodot_{i=1,k_i\neq 0}^{l} C_{p_i}^{{p_i}^{k_i-1}{n_i}^{k_i}} \\
&= \bigoplus_{\substack{k_1+\ldots+k_l=w \\ 0\le k_1,\ldots,k_l\le w}} \binom{w}{k_1,\ldots,k_l} C_{\lambda_l^*}^{(\prod_{\substack{i=1\\k_i\neq 0}}^{l} {p_i}^{k_i-1}{n_i}^{k_i})(\prod_{\substack{i=2\\k_i\neq 0}}^{l} \gcd(\lambda_{i-1}^*,p_i))}.
\end{aligned}$$

□

Remark 5.1.2. According to Remark 5.1.1, for any positive naturals $l > 1$, $w > 1$, $n_1, \ldots, n_l$, and $p_1, \ldots, p_l$, it holds that

$$\left(\bigoplus_{i=1}^{l} C_{p_i}^{n_i}\right)^w = \bigoplus_{\substack{k_1+\ldots+k_l=w \\ 0\le k_1,\ldots,k_l\le w}} \binom{w}{k_1,\ldots,k_l} C_{\lambda_l^*}^{\frac{1}{\lambda_l^*}\cdot\prod_{i=1}^{l}(p_i n_i)^{k_i}}$$

where, for any tuple $k_1, \ldots, k_i$, λ_i^* is the lcm of those p_j (with $j \in \{1, \ldots, i\}$) such that $k_j \neq 0$ (while $\lambda_i^* = 1$ iff all $k_j = 0$).

Now that we know how the operations of sum, product and exponentiation act on the cyclic parts of systems, let us go back to Equation (4.1) which is the problem that we want to solve. By considering just the asymptotic behaviour of all constants and variables, Equation (4.1) can be rewritten as follows to obtain the *a-abstraction equation*

$$(\bigoplus_{i=1}^{l_1} C_{p_{1,i}}^{n_{1,i}} \odot \mathring{X}_1^{\,w_1}) \oplus (\bigoplus_{i=1}^{l_2} C_{p_{2,i}}^{n_{2,i}} \odot \mathring{X}_2^{\,w_2}) \oplus \ldots \oplus (\bigoplus_{i=1}^{l_m} C_{p_{m,i}}^{n_{m,i}} \odot \mathring{X}_m^{\,w_m}) = \bigoplus_{j=1}^{l_B} C_{p_j}^{n_j} \tag{5.4}$$

where, for each $z \in \{1, \ldots, m\}$ the a-abstraction of the coefficient A_z and the a-abstraction of the known term B are

$$\mathring{A}_z = \bigoplus_{i=1}^{l_z} C_{p_{z,i}}^{n_{z,i}} \qquad \text{and} \qquad \mathring{B} = \bigoplus_{j=1}^{l_B} C_{p_j}^{n_j} \ .$$

Then, let l_z be the number of different length of cycles in the system A_z with $z \in \{1, \ldots, m\}$ (of Equation (4.1)), $p_{z,j}$ be the j^{th} different length of period in the system A_z, and $n_{z,j}$ is the number of cycles of length $p_{z,j}$. Each monomial contains a variable X_z and its exponent w_z. In the right term B, there are l_B different periods, where for the j^{th} different period there are n_j cycles of period p_j.

To solve the a-abstraction equation, we first carry out some simplifications. Indeed, we can imagine a polynomial in which we have two monomials $\mathring{A}_z \cdot \mathring{X}_z^{w_z}$ and $\mathring{A}_{z'} \cdot \mathring{X}_{z'}^{w_{z'}}$ such that $\mathring{X}_z = \mathring{X}_{z'}$ and $w_z = w_{z'}$ (with $z \in \{1, \ldots, m\}$). Then, we consider the actual number $\boldsymbol{m} \leq m$ of distinct pairs $(\mathring{X}_z, w_z)$ appearing in such an equation. In this way, Equation (5.4) can be rewritten as

$$\bigoplus_{i=1}^{\ell_1} C_{p_{1,i}}^{n_{1,i}} \odot \mathring{\mathbb{X}}_1 \oplus \bigoplus_{i=1}^{\ell_2} C_{p_{2,i}}^{n_{2,i}} \odot \mathring{\mathbb{X}}_2 \oplus \ldots \oplus \bigoplus_{i=1}^{\ell_{\boldsymbol{m}}} C_{p_{\boldsymbol{m},i}}^{n_{\boldsymbol{m},i}} \odot \mathring{\mathbb{X}}_{\boldsymbol{m}} = \bigoplus_{j=1}^{l_B} C_{p_j}^{n_j} \ , \tag{5.5}$$

where, for each $z \in \{1, \ldots, \boldsymbol{m}\}$, $\mathring{\mathbb{X}}_z$ denotes the z-th different variable-exponent pair $\mathring{X}_z^{w_z}$, and ℓ_z is the number of the distinct cycle lengths involved in a monomial with $\mathring{\mathbb{X}}_z$. At this point, in this simplified version, with an abuse of notation, $n_{z,i}$ is the number of cycles of the i^{th} cycle length $p_{z,i}$ involved in a monomial with $\mathring{\mathbb{X}}_z$.

In contrast to Chapter 4, here the a-abstraction equation is simplified before studying the solutions because this allows us to limit the combinatorial complexity of the problem. This will be evident in a moment.

Equation (5.5) is still hard to solve in this form. We can further simplify it by performing a **contraction step**. This permits us to identify a set of systems such that the union of their solutions is the set of solutions of Equation (5.5). In fact, this step consists in rewriting the simplified equation as a set of Systems (5.6), one for each vector $(n_1^{1,1}, \ldots, n_{l_B}^{1,1})$ obtained by varying each $n_j^{1,1} \in \{0, \ldots, n_j\}$ with $j \in \{1, \ldots, l_B\}$. Hence, the idea is to create a system for every possible subset of the set of cycles in the right term.

$$\begin{cases} C_{p_{1,1}}^{n_{1,1}} \odot \mathring{\mathbb{X}}_1 = \bigoplus_{j=1}^{l_B} C_{p_j}^{n_j^{1,1}} & \text{(5.6a)} \\ C_1^1 \odot \mathring{\mathbb{Y}} = \bigoplus_{j=1}^{l_B} C_{p_j}^{n_j - n_j^{1,1}} & \text{(5.6b)} \end{cases}$$

where $\mathring{\mathbb{Y}} = (\bigoplus_{i=2}^{\ell_1} C_{p_{1,i}}^{n_{1,i}} \odot \mathring{\mathbb{X}}_1) \oplus (\bigoplus_{i=1}^{\ell_2} C_{p_{2,i}}^{n_{2,i}} \odot \mathring{\mathbb{X}}_2) \oplus \ldots \oplus (\bigoplus_{i=1}^{\ell_m} C_{p_{m,i}}^{n_{m,i}} \odot \mathring{\mathbb{X}}_m)$. Note that we use $n_j^{1,1}$ to indicate the number of cycles of length p_j that are associated with the monomial $C_{p_{1,1}}^{n_{1,1}} \odot \mathring{\mathbb{X}}_1$.

At this point, let us repeat as long as possible the application of the contraction step over the last equation of each system obtained by the previous contraction step. We stress that such an application essentially consists in:

i) updating $\mathring{\mathbb{Y}}$ by removing a term $C_{p_{z,i}}^{n_{z,i}} \odot \mathring{\mathbb{X}}_z$ with $z \in \{1, \ldots, m\}$ and $i \in \{1, \ldots, \ell_z\}$,

ii) considering all possible vectors $(n_1^{z,i}, \ldots, n_{l_B}^{z,i})$ obtained varying each $n_j^{z,i}$ with $j \in \{1, \ldots, l_B\}$ from 0 to the remaining number of cycles of length p_j of the right-hand side,

iii) introducing, for each of the above mentioned vectors, a new system obtained by adding the following equation

$$C_{p_{z,i}}^{n_{z,i}} \odot \mathring{\mathbb{X}}_z = \bigoplus_{j=1}^{l_B} C_{p_j}^{n_j^{z,i}}$$

to the considered initial system just before the equation involving $\mathring{\mathbb{Y}}$.

iv) updating the right-hand side of the equation involving $\mathring{\mathbb{Y}}$ by removing $n_j^{z,i}$ cycles from the unions of cycles of length p_j.

In this way, an Equation (5.5) can be equivalently rewritten as a set of systems, each of them having the following form

$$\begin{cases} C_{p_{1,1}}^{n_{1,1}} \odot \mathring{\mathbb{X}}_1 = \bigoplus_{j=1}^{l_B} C_{p_j}^{n_j^{1,1}} \\ C_{p_{1,2}}^{n_{1,2}} \odot \mathring{\mathbb{X}}_1 = \bigoplus_{j=1}^{l_B} C_{p_j}^{n_j^{1,2}} \\ \vdots \\ C_{p_{1,\ell_1}}^{n_{1,\ell_1}} \odot \mathring{\mathbb{X}}_1 = \bigoplus_{j=1}^{l_B} C_{p_j}^{n_j^{1,\ell_1}} \\ C_{p_{2,1}}^{n_{2,1}} \odot \mathring{\mathbb{X}}_2 = \bigoplus_{j=1}^{l_B} C_{p_j}^{n_j^{2,1}} \\ \vdots \\ C_{p_{m,\ell_m}}^{n_{m,\ell_m}} \odot \mathring{\mathbb{X}}_m = \bigoplus_{j=1}^{l_B} C_{p_j}^{n_j^{m,\ell_m}} \end{cases} . \tag{5.7}$$

Intuitively, the system links each monomial to a specific subset of the cycles contained in the right-hand term of the equation. Then, we stress that, for each $j \in \{1, ..., l_B\}$, it holds that $n_j = \sum_{z=1}^{m} \sum_{i=1}^{\ell_z} n_j^{z,i}$, where $n_j^{z,i}$ represents the number of cycles of length p_j that the monomial $C_{p_{z,i}}^{n_{z,i}} \odot \mathring{\mathbb{X}}_z$ contributes to form. As a consequence, solving Equation (5.4) boils down to solve this finite set of Systems (5.7).

For the sake of completeness, let us specify the number of different systems that exist. For each different cycle length in the right-hand term, each cycle must have been generated by one of

the monomials. By a similar reasoning to the one made for the c-astraction, we have a number of

$$\prod_{j=1}^{l_B} \binom{n_j + (\sum_{z=1}^{\boldsymbol{m}} \ell_z) - 1}{(\sum_{z=1}^{\boldsymbol{m}} \ell_z) - 1}$$

different systems.

Now, to solve any equation $C_{p_{z,i}}^{n_{z,i}} \odot \mathring{\mathbb{X}}_z = \bigoplus_{j=1}^{l_B} C_{p_j}^{n_j^{z,i}}$ involved in a System (5.7), it is enough to solve the following l_B equations

$$\begin{aligned} C_{p_{z,i}}^{n_{z,i}} \odot \mathring{\mathbb{X}}_z &= C_{p_1}^{n_1^{z,i}}, \\ C_{p_{z,i}}^{n_{z,i}} \odot \mathring{\mathbb{X}}_z &= C_{p_2}^{n_2^{z,i}}, \\ &\vdots \\ C_{p_{z,i}}^{n_{z,i}} \odot \mathring{\mathbb{X}}_z &= C_{p_{l_B}}^{n_{l_B}^{z,i}} \end{aligned}$$

and compute the Cartesian product among their solutions. We are in fact looking to enumerate all possible ways to generate the cycles of each different length, and combine them (in all possible ways) to enumerate the possible values of $\mathring{\mathbb{X}}_z$.

According to Equation (5.2), an equation $C_{p_{z,i}}^{n_{z,i}} \odot \mathring{\mathbb{X}}_z = C_{p_j}^{n_j^{z,i}}$ can be rewritten as

$$C_{p_{z,i}}^{1} \odot \mathring{\mathbb{X}}_z = C_{p_j}^{n_j^{z,i}/n_{z,i}}. \tag{5.8}$$

Obviously, an equation $C_{p_{z,i}}^{n_{z,i}} \odot \mathring{\mathbb{X}}_z = C_{p_j}^{n_j^{z,i}}$ can have a solution only if $n_j^{z,i}/n_{z,i}$ is an integer.

In addition, to find the solutions of a System (5.7), we must also calculate the intersections between the solutions found by the different equations for the same variable.

In conclusion, solving Equation (5.5) corresponds to identify all the Systems (5.7) and perform the Cartesian products and intersections of the solutions of a certain number of simpler equations, called **basic equations**, of the following form:

$$C_p^1 \odot \mathring{\mathbb{X}} = C_q^n \ , \tag{5.9}$$

where $\mathring{\mathbb{X}}$ is some $\mathring{\mathbb{X}}_z$, $p \in \{p_{1,1}\ , p_{1,2}\ , \ldots\ , p_{\boldsymbol{m},\ell_{\boldsymbol{m}}}\}$, $q \in \{p_1, \ldots, p_{l_B}\}$, and, making reference to the right-hand side, n is smaller or equal to n_j, *i.e.*, the number of cycles of length $q = p_j$.

Example 5.1.1 – Consider the equation

$$(C_5^1 \oplus C_2^1) \odot \mathring{\mathbb{X}}_1 \oplus C_3^1 \oplus \mathring{\mathbb{X}}_2 = C_6^2 \oplus C_{15}^1.$$

With an initial application of the contraction steps, we obtain the following systems

$$\begin{cases} C_5^1 \odot \mathring{\mathbb{X}}_1 = C_6^1 \\ C_1^1 \odot \mathring{\mathbb{Y}} = C_6^1 \oplus C_{15}^1 \end{cases} \quad \begin{cases} C_5^1 \odot \mathring{\mathbb{X}}_1 = C_6^2 \\ C_1^1 \odot \mathring{\mathbb{Y}} = C_{15}^1 \end{cases} \quad \begin{cases} C_5^1 \odot \mathring{\mathbb{X}}_1 = C_{15}^1 \\ C_1^1 \odot \mathring{\mathbb{Y}} = C_6^2 \end{cases}$$

$$\begin{cases} C_5^1 \odot \mathring{\mathbb{X}}_1 = C_6^1 \oplus C_{15}^1 \\ C_1^1 \odot \mathring{\mathbb{Y}} = C_6^1 \end{cases} \quad \begin{cases} C_5^1 \odot \mathring{\mathbb{X}}_1 = C_6^2 \oplus C_{15}^1 \end{cases} \quad \begin{cases} C_1^1 \odot \mathring{\mathbb{Y}} = C_6^2 \oplus C_{15}^1 \end{cases}$$

where $\mathring{\mathbb{Y}} = C_2^1 \odot \mathring{\mathbb{X}}_1 \oplus C_3^1 \oplus \mathring{\mathbb{X}}_2$. Let us specify that a $n_j^{z,i}$ equal to 0 implies that that the monomial does not contribute to the creation of cycles of length p_j.

To continue with the contraction steps, let us consider the first system shown above. The monomial $C_2^1 \odot \mathring{\mathbb{X}}_1$ is removed from $\mathring{\mathbb{Y}}$. Considering all possible values of $n_j^{1,2}$ with $j \in \{1, 2\}$ we obtain the systems below (remark that the number of components remaining to be generated for the monomial $C_1^1 \odot \mathring{\mathbb{Y}}$ is updated accordingly).

$$\begin{cases} C_5^1 \odot \mathring{\mathbb{X}}_1 = C_6^1 \\ C_2^1 \odot \mathring{\mathbb{X}}_1 = C_{15}^1 \\ C_1^1 \odot \mathring{\mathbb{Y}} = C_6^1 \end{cases} \quad \begin{cases} C_5^1 \odot \mathring{\mathbb{X}}_1 = C_6^1 \\ C_2^1 \odot \mathring{\mathbb{X}}_1 = C_6^1 \\ C_1^1 \odot \mathring{\mathbb{Y}} = C_{15}^1 \end{cases} \quad \begin{cases} C_5^1 \odot \mathring{\mathbb{X}}_1 = C_6^1 \\ C_1^1 \odot \mathring{\mathbb{Y}} = C_6^1 \oplus C_{15}^1 \end{cases} \quad \begin{cases} C_5^1 \odot \mathring{\mathbb{X}}_1 = C_6^1 \\ C_2^1 \odot \mathring{\mathbb{X}}_1 = C_6^1 \oplus C_{15}^1 \end{cases}$$

Iterating the contraction steps on the systems above again, we find that the monomial $C_1^1 \odot \mathring{\mathbb{Y}}$ is simply replaced by the monomial $C_3^1 \oplus \mathring{\mathbb{X}}_2$. We note how the same process must be applied iteratively to all the first six systems found (in the beginning) in order to have the systems that yield all the solutions of the original equation.

Contraction steps allow to find solutions to equations of type (5.5) at the price of solving a huge number of equations of type (5.9). The enumeration of the solutions is necessary according to the goal of this work. Therefore, we need to consider all the possible contraction steps and all solutions of basic equations (5.9). Then, to solve Equation (5.5), we need an efficient method that:

- enumerates the solutions of all Equations (5.9),
- computes all Systems (5.7) and enumerates the solutions of each system (through Cartesian products and intersections of the solutions of the basic equations),
- computes the solutions of $\mathring{X}_z$ from the set of solutions found for $\mathring{X}_z^{w_z}$.

5.2 Basic equations for the a-abstraction

In this chapter, we will see how we will enumerate the solutions of an equation on a-abstractions. However, since we have realised that to solve an Equation (5.4) we must solve a finite number of basic equations (5.9), the following section begins with the problem of enumerating the solutions of a basic equation.

5.2.1 The SOBFID and EnumSOBFID problems

Let us formally introduce the problem of solving Equations (5.9) and investigating its computational complexity.

Definition 5.2.1 (SOBFID)**.** $SOBFID$ (*SOlve equation on Bijective FInite DDS*) is the decision problem which takes in input three positive integers $p, q, n \in \mathbb{N} \setminus \{0\}$ and returns true iff Equation (5.9) admits a solution.

To prove the complexity of $SOBFID$, we need the following result which relates the existence of solutions of any basic equation to the prime factorisation of p, q, and n.

Theorem 5.2.1. *Consider the equation*

$$C_p^1 \odot \bigoplus_{i=1}^{s} C_{p_i'}^1 = C_q^n$$

where $\bigoplus_{i=1}^{s} C_{p_i'}^1$ *is a set of cycles of possibly equivalent lengths, and the prime factorisations of* p*,* n *and* q *are:*

$p = \mathsf{p}_1^{h_1}\mathsf{p}_2^{h_2}, \ldots, \mathsf{p}_\psi^{h_\psi}$ *with factors*$(p) = \{\mathsf{p}_1, \mathsf{p}_2, \ldots, \mathsf{p}_\psi\}$,
$n = \mathsf{n}_1^{t_1}\mathsf{n}_2^{t_2}, \ldots, \mathsf{n}_\tau^{t_\tau}$ *with factors*$(n) = \{\mathsf{n}_1, \mathsf{n}_2, \ldots, \mathsf{n}_\tau\}$, *and*
$q = \mathsf{q}_1^{o_1}\mathsf{q}_2^{o_2}, \ldots, \mathsf{q}_\iota^{o_\iota}$ *with factors*$(q) = \{\mathsf{q}_1, \mathsf{q}_2, \ldots, \mathsf{q}_\iota\}$.

The equation has a solution if and only if p *divides* q*, and* $\forall i \in \{1, \ldots, \psi\}$, $\mathsf{p}_i^{h_i} \in \{\mathsf{q}_1^{o_1}, \mathsf{q}_2^{o_2}, \ldots, \mathsf{q}_\iota^{o_\iota}\}$ *or* $\mathsf{p}_i^{h_i}$ *divides* n*.*

Proof. According to Proposition 5.1.3, to have a solution, p must divide q. This means that factors$(p) \subseteq$ factors(q) and the exponents of the elements in factors(p) will be less than or equal to the corresponding exponents of the elements in factors(q).

($\Leftarrow$) Given the factorisation of q and Proposition 5.1.3, we can rewrite C_q^n as follows.

$$C_q^n = C_q^1 \odot C_1^n = C_{\mathsf{q}_1^{o_1}\mathsf{q}_2^{o_2},\ldots,\mathsf{q}_\iota^{o_\iota}}^1 \odot C_1^n = C_{\mathsf{q}_1^{o_1}}^1 \odot C_{\mathsf{q}_2^{o_2}}^1 \odot \ldots \odot C_{\mathsf{q}_\iota^{o_\iota}}^1 \odot C_1^n$$

Suppose now that one of $\mathsf{p}_i^{h_i}$ is not contained in $\{\mathsf{q}_1^{o_1}\mathsf{q}_2^{o_2}, \ldots, \mathsf{q}_\iota^{o_\iota}\}$. Then, it must be true that $\mathsf{p}_i = \mathsf{q}_j$ (for a certain $j \in \{1, \ldots, \iota\}$) with $h_i < o_j$, and if $\mathsf{p}_i^{h_i}$ divides n then we have the following

$$C_{\mathsf{q}_1^{o_1}}^1 \odot C_{\mathsf{q}_2^{o_2}}^1 \odot \ldots \odot C_{\mathsf{q}_\iota^{o_\iota}}^1 \odot C_1^n = C_{\mathsf{q}_1^{o_1}}^1 \odot C_{\mathsf{q}_2^{o_2}}^1 \odot \ldots \odot C_{\mathsf{q}_\iota^{o_\iota}}^1 \odot C_{\mathsf{p}_i^{h_i}}^1 \odot C_1^{\frac{n}{(\mathsf{p}_i^{h_i})}}.$$

By repeating for all $\mathsf{p}_i^{h_i}$ not included in $\{\mathsf{q}_1^{o_1}\mathsf{q}_2^{o_2}, \ldots, \mathsf{q}_\iota^{o_\iota}\}$, we reach the goal since $\bigoplus_{i=1}^{s} C_{p_i'}^1$ will contain a cycle $C_{p_i'}^1$ (with p_i' equals $\mathsf{q}_j^{o_j}$) for any $\mathsf{p}_i^{h_i}$ not included, and the set of self-loops.

($\Rightarrow$) Let us suppose that the equation $C_p^1 \odot \bigoplus_{i=1}^{s} C_{p_i'}^1 = C_q^n$ holds for some $p_1', p_2', \ldots, p_s'$. Let us assume that there exists i such that $\mathsf{p}_i^{h_i} \notin \{\mathsf{q}_1^{o_1}\mathsf{q}_2^{o_2}, \ldots, \mathsf{q}_\iota^{o_\iota}\}$. Then $\mathsf{p}_i^{h_i}$ will be equal to a certain q_j^y with $y < o_j$. This implies that all $p_1', p_2', \ldots, p_s'$ must contain exactly $\mathsf{q}_j^{o_j}$ as a factor, since the lcm with p must be q. Thus we can write:

$$\bigoplus_{i=1}^{s} C_{p_i'}^1 = C_{\mathsf{q}_j^{o_j}}^1 \odot \bigoplus_{i=1}^{s} C^1{}_{\frac{p_i'}{(\mathsf{q}_j^{o_j})}}$$

which leads us to the following.

$$\begin{aligned} C_p^1 \odot \bigoplus_{i=1}^{s} C_{p_i'}^1 &= C_p^1 \odot C_{\mathsf{q}_j^{o_j}}^1 \odot \bigoplus_{i=1}^{s} C^1{}_{\frac{p_i'}{(\mathsf{q}_j^{o_j})}} \\ &= C^1{}_{\frac{p}{(\mathsf{q}_j^{y})}} \odot C_{\mathsf{q}_j^{y}}^1 \odot C_{\mathsf{q}_j^{o_j}}^1 \odot \bigoplus_{i=1}^{s} C^1{}_{\frac{p_i'}{(\mathsf{q}_j^{o_j})}} \\ &= C^1{}_{\frac{p}{(\mathsf{q}_j^{y})}} \odot C^{(\mathsf{q}_j^{y})}{}_{(\mathsf{q}_j^{o_j})} \odot \bigoplus_{i=1}^{s} C^1{}_{\frac{p_i'}{(\mathsf{q}_j^{o_j})}} \end{aligned}$$

Then, according to Proposition 5.1.3, $\mathsf{p}_i^{h_i}$ (*i.e.* q_j^{y}) must divide n. □

This theorem allows us to provide a condition to decide whether a solution can exist. To determine the complexity of verifying whether or not the condition is fulfilled by a certain SOBFID instance, we introduce two important remarks. We emphasise that these will be applied to the problem we are studying but they are true for any pair of integers.

Remark 5.2.1. Let $q = \mathsf{q}_1^{o_1}\mathsf{q}_2^{o_2}, \ldots, \mathsf{q}_\iota^{o_\iota}$ with factors$(q) = \{\mathsf{q}_1, \mathsf{q}_2, \ldots, \mathsf{q}_\iota\}$, and $p = \mathsf{p}_1^{h_1}\mathsf{p}_2^{h_2}, \ldots, \mathsf{p}_\psi^{h_\psi}$ with factors$(p) = \{\mathsf{p}_1, \mathsf{p}_2, \ldots, \mathsf{p}_\psi\}$ be two positive integers, and let $\mathbf{F}$ be the set of all $\mathsf{q}_j^{o_j}$ such that $\mathsf{q}_j \in$ factors(q) and $\mathsf{q}_j \notin$ factors(p). Algorithm 7 computes $\Pi_{\mathbf{F}} = \prod_{\mathbf{f}\in\mathbf{F}} \mathbf{f}$ without knowing the factorisations of q and p.

Algorithm 7: $\Pi_{\mathbf{F}}$

Input : q and p integers
Output: $\prod_{\mathbf{f}\in\mathbf{F}} \mathbf{f}$ integer

```
if gcd(q, p) == 1 then
    return q;
else
    return Π_F(q / gcd(q,p), gcd(q, p));
```

Remark 5.2.2. Let $q = \mathsf{q}_1^{o_1}\mathsf{q}_2^{o_2}, \ldots, \mathsf{q}_\iota^{o_\iota}$ with factors$(q) = \{\mathsf{q}_1, \mathsf{q}_2, \ldots, \mathsf{q}_\iota\}$, and $p = \mathsf{q}_1^{h_1}\mathsf{q}_2^{h_2}, \ldots, \mathsf{q}_\psi^{h_\psi}$ with factors$(n) = \{\mathsf{q}_1, \mathsf{q}_2, \ldots, \mathsf{q}_\psi\}$ be two positive integers with $h_i \leq o_j$ for all $\mathsf{q}_i = \mathsf{q}_j$, and let $\mathbf{E}$ be the set of all $\mathsf{q}_i^{h_i}$ such that $h_i < o_j$. Algorithm 8 computes $\Pi_{\mathbf{E}} = \prod_{\mathbf{e}\in\mathbf{E}} \mathbf{e}$ without knowing the factorisations of q and p.

Algorithm 8: $\Pi_{\mathbf{E}}$

Input : q and p integers
Output: $\prod_{\mathbf{e}\in\mathbf{E}} \mathbf{e}$ integer

```
i ← 1;
div ← q / p;
g ← gcd(div^i, p);
g' ← gcd(div^(i+1), p);
while g_i ≠ g_(i+1) do
    i ← i + 1;
    g ← g';
    g' ← gcd(div^(i+1), p);
return g_i;
```

Theorem 5.2.2. *SOBFID is in P.*

Proof. We can solve SOBFID ,without knowing the factorisations of q, p, and n, by means of Algorithm 9. In fact, according to Proposition 5.1.3, p must divide q (Algorithm 9 line 1). Then, factors$(p) \subseteq$ factors(q) and the exponents of the elements in factors(p) will be less than or equal to the corresponding exponents of the elements in factors(q). However, to get lcm equals q, all factors contained in q, such that p has a strictly lower exponent, must be contained in the solution. For this reason, $\Pi_{\mathbf{F}}(q,p)$ looks for the factors which are only present in q and then finds, by doing $\frac{q}{\Pi_{\mathbf{F}}(q,p)}$, all those common between q and p. However, by doing this, we find the factors with the exponent they have in q. It is then the function $\Pi_{\mathbf{E}}(\frac{q}{\Pi_{\mathbf{F}}(q,p)}, p)$ which finds all the factors $\mathfrak{q}_i^{h_j}$ which will have to be contained in the solution and which will have to divide n according to Proposition 5.1.3 (Algorithm 9 line 4). □

Algorithm 9: DecisionSOBFID

Input : p, n, and q integers
Output: $true$ iff $C_p^1 \odot \mathring{\mathbb{X}} = C_q^n$ has solution, $false$ otherwise

```
1 if p ∤ q then
2 |   return false;
3 else
4 |   if Π_E(q/Π_F(q,p), p) | n then
5 |   |   return true;
6 |   else
7 |   |   return false;
```

Example 5.2.1 – Consider the following equation

$$C_{8400}^1 \odot \mathring{\mathbb{X}} = C_{8316000}^{6000}$$

where $p = 8400$, $q = 8316000$ and $n = 6000$. Let us first see how by means of the presented algorithms we can verify the condition of Theorem 5.2.1 without knowing the factorisations of the numbers. Later we will see how the methods act at the level of the factorisations.

In this case p divides q, so we want to calculate $\Pi_{\mathbf{E}}(\frac{8316000}{\Pi_{\mathbf{F}}(8316000,8400)}, 8400)$. Let us begin by considering $\Pi_{\mathbf{F}}(8316000, 8400)$. Since 8316000 and 8400 are not coprime, the method is interated, *i.e.* $\Pi_{\mathbf{F}}(\frac{8316000}{\gcd(8316000,8400)}, \gcd(8316000, 8400)) = \Pi_{\mathbf{F}}(990, 8400)$. Again, 990 and 8400 are not coprime, we call recursively the function $\Pi_{\mathbf{F}}(\frac{990}{\gcd(990,8400)}, \gcd(990, 8400)) = \Pi_{\mathbf{F}}(33, 30)$ which brings us to $\Pi_{\mathbf{F}}(\frac{33}{\gcd(33,30)}, \gcd(33, 30)) = \Pi_{\mathbf{F}}(11, 3)$. Since 11 and 3 are coprime, the method returns 11.

Let us therefore study $\Pi_{\mathbf{E}}(\frac{8316000}{11}, 8400) = \Pi_{\mathbf{E}}(756000, 8400)$. With $i = 1$, $div = \frac{756000}{8400} = 90$, $g = \gcd(90, 8400) = 30$ and $g' = \gcd(90^2, 8400) = \gcd(8100, 8400) = 300$. Since $g \neq g'$, i becomes 2, g takes value 300 and g' becomes $\gcd(90^3, 8400) = \gcd(729000, 8400) = 600$. Hence, since g and g' are still not equivalent, the method continues

as follows.

$$i = 3, g = 600, g' = \gcd(90^4, 8400) = \gcd(65610000, 8400) = 1200$$

$$i = 4, g = 1200, g' = \gcd(90^5, 8400) = \gcd(5904900000, 8400) = 1200$$

Since g and g' are equivalent, the method returns 1200. Finally, since 1200 divides $n = 6000$ we know that the equation admits of solution.

Let us now see what happens, from the point of view of the factorisations, by applying these methods. Considering the values in input of this example, the factorisations are:

$$p = 2^4 \cdot 3 \cdot 5^2 \cdot 7, \quad q = 2^5 \cdot 3^3 \cdot 5^3 \cdot 7 \cdot 11, \quad n = 2^4 \cdot 3 \cdot 5^3$$

The goal of $\Pi_{\mathbf{F}}$ is to calculate the product of all $\mathsf{q}_j^{o_j}$ such that $\mathsf{q}_j \in \text{factors}(q)$ and $\mathsf{q}_j \notin \text{factors}(p)$, and in fact the method calculates the following.

$$\begin{aligned}
&\Pi_{\mathbf{F}}(2^5 \cdot 3^3 \cdot 5^3 \cdot 7 \cdot 11, 2^4 \cdot 3 \cdot 5^2 \cdot 7) = \\
&\quad = \Pi_{\mathbf{F}}(\frac{2^5 \cdot 3^3 \cdot 5^3 \cdot 7 \cdot 11}{\gcd(2^5 \cdot 3^3 \cdot 5^3 \cdot 7 \cdot 11, 2^4 \cdot 3 \cdot 5^2 \cdot 7)}, \gcd(2^5 \cdot 3^3 \cdot 5^3 \cdot 7 \cdot 11, 2^4 \cdot 3 \cdot 5^2 \cdot 7)) \\
&\quad = \Pi_{\mathbf{F}}(2 \cdot 3^2 \cdot 5 \cdot 11, 2^4 \cdot 3 \cdot 5^2 \cdot 7)
\end{aligned}$$

$$\begin{aligned}
&\Pi_{\mathbf{F}}(2 \cdot 3^2 \cdot 5 \cdot 11, 2^4 \cdot 3 \cdot 5^2 \cdot 7) = \\
&\quad = \Pi_{\mathbf{F}}(\frac{2 \cdot 3^2 \cdot 5 \cdot 11}{\gcd(2 \cdot 3^2 \cdot 5 \cdot 11, 2^4 \cdot 3 \cdot 5^2 \cdot 7)}, \gcd(2 \cdot 3^2 \cdot 5 \cdot 11, 2^4 \cdot 3 \cdot 5^2 \cdot 7)) \\
&\quad = \Pi_{\mathbf{F}}(3 \cdot 11, 2 \cdot 3 \cdot 5)
\end{aligned}$$

$$\begin{aligned}
&\Pi_{\mathbf{F}}(3 \cdot 11, 2 \cdot 3 \cdot 5) = \\
&\quad = \Pi_{\mathbf{F}}(\frac{3 \cdot 11}{\gcd(3 \cdot 11, 2 \cdot 3 \cdot 5)}, \gcd(3 \cdot 11, 2 \cdot 3 \cdot 5)) \\
&\quad = \Pi_{\mathbf{F}}(11, 3) = 11.
\end{aligned}$$

Once $\Pi_{\mathbf{F}}$ has been calculated, through $\frac{q}{\Pi_{\mathbf{F}}(q,p)}$ we obtain the product of all $\mathsf{q}_j^{o_j}$ such that $\mathsf{q}_j \in \text{factors}(q)$ and $\mathsf{q}_j \in \text{factors}(p)$, *i.e.* $2^5 \cdot 3^3 \cdot 5^3 \cdot 7$. Then, the goal of $\Pi_{\mathbf{E}}$ is to calculate the product of all $\mathsf{q}_i^{h_j}$ such that $\mathsf{p}_j = \mathsf{q}_i$ $h_j < o_i$. Note that $90 = 2 \cdot 3^2 \cdot 5$.

$$i = 1, g = \gcd(2 \cdot 3^2 \cdot 5, 2^4 \cdot 3 \cdot 5^2 \cdot 7) = 2 \cdot 3 \cdot 5, g' = \gcd(2^2 \cdot 3^4 \cdot 5^2, 2^4 \cdot 3 \cdot 5^2 \cdot 7) = 2^2 \cdot 3 \cdot 5^2,$$

$$\begin{aligned}
i = 2, g = 2^2 \cdot 3 \cdot 5^2, g' = \gcd(2^3 \cdot 3^6 \cdot 5^3, 2^4 \cdot 3 \cdot 5^2 \cdot 7) = 2^3 \cdot 3 \cdot 5^2, \\
i = 3, g = 2^3 \cdot 3 \cdot 5^2, g' = \gcd(2^4 \cdot 3^8 \cdot 5^4, 2^4 \cdot 3 \cdot 5^2 \cdot 7) = 2^4 \cdot 3 \cdot 5^2, \\
i = 4, g = 2^4 \cdot 3 \cdot 5^2, g' = \gcd(2^5 \cdot 3^{10} \cdot 5^5, 2^4 \cdot 3 \cdot 5^2 \cdot 7) = 2^4 \cdot 3 \cdot 5^2.
\end{aligned}$$

Let us now turn our attention to the enumeration version of the problem.

Definition 5.2.2 (EnumSOBFID)**.** EnumSOBFID is an enumeration problem which takes in input $p, q, n \in \mathbb{N} \setminus \{0\}$ and returns the list of all solutions to Equation (5.9).

The following result classifies our enumeration problem in EnumP *i.e.*, the complexity class of enumeration problems for which a solution can be verified in polynomial time. EnumP can be seen as the enumeration counterpart of the NP complexity class. For more on the complexity classes of enumeration problems, the reader may refer to [Mary and Strozecki (2019)].

Proposition 5.2.3. *EnumSOBFID is in EnumP.*

Proof. Given an equation $C_p^1 \odot \mathring{\mathbb{X}} = C_q^n$ and a solution $\mathring{\mathbb{X}} = \bigoplus_{i=1}^{l} C_{p_i}^{n_i}$, the verification requires the following computations:

$$\sum_{i=1}^{l} \gcd(p, n_i) = n \text{ and } \forall i \in \{1, \ldots, l\} \operatorname{lcm}(p, p_i) = q$$

All those computations can be made in polynomial time, hence EnumSOBFID $\in EnumP$. □

5.2.2 The Colored-tree method and the Change-Making problem

In this section, we present a first technique to enumerate all the solutions of an EnumSOBFID instance. As explained before, this is the first essential step in introducing a technique to list the solutions of an Equation (5.4). The method is based on two principal phases: tree building and solutions aggregation. During the first phase, we explore the solution space of interest using a tree, and in the second one, the solutions of the equation represented in the tree are computed. This is a very simple algorithmic technique based on the connection between the enumeration of the solutions of interest and the *Change-Making problem* (CMP). We will see that one can improve this method from a memory usage point of view, as well as, from a time point of view. However, it allows us to introduce the main idea and concepts that will be used in the MDD-based technique.

The CMP is known as an optimisation problem which takes in input a sequence $\$ = \{c_1, c_2, \ldots, c_t\}$ of positive integers, named the *coin system*, and an integer T. In these inputs, each c_i represents a coin denomination and T is a sum which has to be expressed as a combination (or sum) of the given coins. This means that we have a number of coins of different denominations at our disposal but CMP requires to use the least number of coins to meet the target total T. Remark that CMP is a special case of the integer knapsack problem [Martello and Toth (1990)]. It is like having t item types and a knapsack, where c_i is the weight of an item of type i and T the capacity of the knapsack. CMP has been demonstrated NP-hard by a polynomial reduction to the knapsack problem, but may be solved in pseudo-polynomial time by dynamic programming [Wright (1975)]. It is well known that a greedy approach provides an easier algorithm but it may fail to find the optimal solution (consider for example $T = 11$ and $\$ = \{9, 6, 5, 1\}$, the solution computed by a greedy approach is $[9, 1, 1]$ but the optimal one is $[6, 5]$).

In the literature, we also find other versions of this problem. For example, the enumeration version of the CMP consists in finding all the possible representations of a certain T under a certain $\$$.

The colored-tree method aims at enumerating solutions for equations of type $C_p^1 \odot \mathring{\mathbb{X}} = C_q^n$. It needs as input three integer values n, p and q (which are the quantities involved in the equation).

Algorithm 10: DynamicCMP

Input : T integer, \$ increasingly ordered coin system
Output: sol an optimal solution for the CMP

```
1  many ← ∅;
2  last ← ∅;
3  sol ← ∅;
4  many[0] ← 0;
5  last[0] ← 0;
6  forall i ∈ {1, ..., T} do
7      c ← $[0];
8      j ← 1;
9      while $[j] ≤ i and j < |$| do
10         c ← $[j];
11     many[i] ← many[i − c] + 1;
12     last[i] ← c;
13 add(sol, last[T]);
14 l ← last[T];
15 while l ≠ 0 do
16     add(sol, last[T − l]);
17     l ← last[T − l];
18 return sol;
```

The connection with the CMP starts here. We have to generate n cycles of length q. Thus, T corresponds to n while q gives us some information to construct the coins system.

According to Equation 5.2 and Proposition 5.1.2, we know that

$$C_p^1 \odot \bigoplus_{i=1}^{s} C_{p'_i}^1 = C_{\mathrm{lcm}(p,p'_1)}^{\gcd(p,p'_1)} \oplus \ldots \oplus C_{\mathrm{lcm}(p,p'_l)}^{\gcd(p,p'_l)} = C_q^n$$

where, for a certain value s, $\mathrm{lcm}(p, p'_i) = q$ for all p'_i with $i \in \{1, \ldots, s\}$. If we want to search for a p'_i, an intuitive search space is, in fact, the divisors of q. Knowing that $\gcd(p, p'_i) \leq n$, we consider only those smaller than or equal to n. Let us call the set of such divisors $div(q, n)$. Moreover, both p and p'_i must be divisors of q and then $\gcd(p, p'_i)$ is a divisor of q too. In other words, we need to decompose n as a sum of divisors of q. Then, the set $div(q, n)$ is the coin system here.

In this method, a data structure is generated to explore the feasible solutions space and to compute all the possible solutions of the equation. This structure is a table (for an easier representation and for saving memory) which can be seen as colored tree. The table contains, for each node r of the tree, the set of CMP solutions computed for $T = r$ and $\$ = div(q, n)$, the solution $\mathring{\mathbb{X}}$ containing only one cycle (such that $C_p^1 \odot \mathring{\mathbb{X}} = C_q^r$) and the solutions for the same equation where $\mathring{\mathbb{X}}$ consists of more cycles (called subtree solutions set).

Algorithm 11: CTM

Input : p, n, q integers
Output: solutions of $C_p^1 \odot \mathring{\mathbb{X}} = C_q^n$

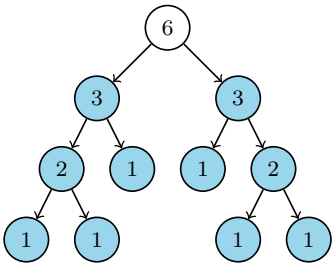
```
1  $ ← div(q, n);
2  Node, toDecompose ← {n};
3  CMPsolutions, NodeSolution, SubtreeSol ← ∅;
4  while toDecompose ≠ ∅ do
5      r ← toDecompose[0];
6      dec ← DynamicCMP(r,$);
7      remove r from toDecompose;
8      add dec to CMPsolutions[r];
9      if ListToSet(dec) ≠ {1} then
10         forall c ∈ ListToSet(dec) do
11             if Node.contains(c) == false then
12                 add c in Node;
13                 if c ≠ 1 then
14                     add c to toDecompose;
15                 else
16                     CMPsolutions[1] ← ∅;
17     else
18         if Node.contains(1)== false then
19             add c to Node;
20             CMPsolutions[1] ← ∅;
21 CompletenessCheck(n, $, CMPsolutions);
22 forall r ∈ Node do
23     if gcd(p, q/p · r) = r and lcm(p, q/p · r) = q then
24         NodeSolution[r] ← C^1_{q/p · r};
25     else
26         NodeSolution[r] ← ∅;
27 SubtreeSol[1] ← {NodeSolution[1]};
28 forall r ∈ Node.ascendingOrder \ {1} do
29     sol ← ∅;
30     forall dec ∈ CMPsolutions[r] do
31         CartesianElements ← ∅;
32         forall elem ∈ dec do
33             add SubtreeSol[elem] to CartesianElements;
34         add CartesianProduct(CartesianElements) to sol;
35     add sol and NodeSolution[r] to SubtreeSol[r];
36 return SubtreeSol[n];
```

The idea is to find all possible solutions increasing, step by step, the number of cycles in the variable $\mathring{\mathbb{X}}$. For this reason, starting from the root n (first line of the table), an optimal solution of the CMP with $T = n$ is computed with a dynamical programming approach (an example is shown in Algorithm 10) [Wright (1975)]. The dynamical approach is necessary because, as anticipated, a greedy approach may fail to obtain an optimal split.

According to the optimal solution, the set of cycles C_q^n is divided into smaller sets, one for each j-th coin involved into the optimal CMP solution computed. This corresponds to the fact that the solutions of equation $C_p^1 \odot \mathring{\mathbb{X}} = C_q^n$ will be identified by solving equations $C_p^1 \odot \mathring{\mathbb{X}} = C_q^{c_j}$, one for each coin c_j. For each chosen coin, a child node is created and the process is iterated. The method stops when the CMP when all coins have value 1. The tree is represented in the table with a line for each possible subset of the n cycles and the edges are represented by the solution of the CMP at every step (see Example 5.2.2). In this way, even if there are duplicate nodes in the tree (with the same r value), they are represented in the table only once.

After this first phase, a check of completeness is applied to ensure the completeness of the exploration carried out. This procedure checks if all the possible ways to make the change of the node n are represented in the structure, if a combination is not represented, it is added in the CMP solution set of the node n and a new subtree is computed (*i.e.* the decomposition is iterated). This new subtree can be denoted with a new color to recall that is a new independent part of the solution space that must be explored.

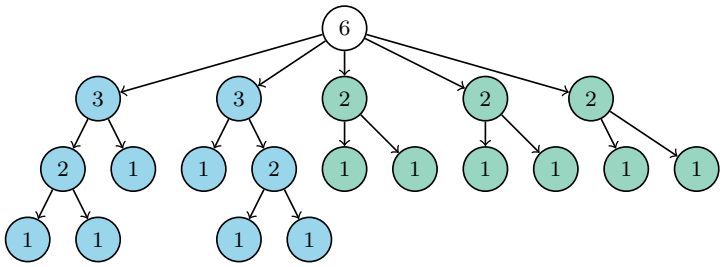
Example 5.2.2 – Consider the equation $C_6^1 \odot \mathring{\mathbb{X}} = C_6^6$. Algorithm 11 consists of two distinct phases: tree building and solutions aggregation. In the first phase, the algorithm computes the coin system $div(q, n) = \{6, 3, 2, 1\}$. Then, it solves an instance of the CMP problem in which the total sum is 6 and the allowed set of coins is $div(q, n) \setminus \{6\}$. It decomposes 6 as $3 + 3$. Then, the same idea is applied recursively (always using $div(q, n) \setminus \{3\}$ as the set of coins). We obtain that 3 is decomposed as $2 + 1$, and later 2 as $1 + 1$.



Node	CMP solutions	Node solution	Subtree solutions set
6	[3,3]		
3	[2,1]		
2	[1,1]		
1	$\emptyset$		

Figure 5.3 – The tree resulting from the CMP solutions for $\$ = \{3, 2, 1\}$ and $T = 6$ together with the corresponding table used to model the structure.

At this point, a check is performed to ensure that all possible ways of decomposing 6 using $div(q, n)$ are present in the structure (this is necessary to explore the whole solutions space). In our case, we already have $[3, 3]$ found by the first run and we also recursively found: $[3, 2, 1]$, $[2, 2, 1, 1]$, $[1, 1, 2, 1, 1]$, $[1, 1, 1, 1, 1, 1]$. By performing the check, we discover that the decomposition of 6 as $[2, 2, 2]$ is not represented. For this reason, $[2, 2, 2]$ is added to the set of decompositions of 6. As illustrated in Figure 5.4, the missing decomposition is assigned a new color and a recursive application of the CMP problem is started on the newly added nodes. This is made only if it is necessary, in other words, only if they are not already represented in the structure. In this case, a node of value 2 (and its possible decompositions) is already represented in the structure, so the process stops. A new check ensures that all decompositions are present. This ends the building phase. The resulting structure is reported in Figure 5.4.



Node	CMP solutions	Node solution	Subtree solutions set
6	[3,3] [2, 2, 2]		
3	[2,1]		
2	[1,1]		
1	$\emptyset$		

Figure 5.4 – The tree resulting at the end of the construction phase and the corresponding table able to model the structure for $\$ = \{3, 2, 1\}$ and $T = 6$.

When also the check procedure is terminated, the computation of the solutions starts (the solution aggregation phase). In this phase, the first step consists in the computation of the single-cycle solution, called the *node solution*, for each node r. According to Equation (5.2), we aim at finding a p' such that $C^1_p \odot C^1_{p'} = C^r_q$, then $p' = \frac{q}{p} \cdot r$. As a consequence, for each node r, $C^1_{\frac{q}{p} \cdot r}$ is the node solution if and only if $\gcd(p, \frac{q}{p} \cdot r) = r$ and $\mathrm{lcm}(p, \frac{q}{p} \cdot r) = q$.

To enumerate all solutions, the last part of the method consists in the computation, for each node, of the set of solutions represented in the subtrees. In this method, the aggregation of the solutions is divided into two steps: the first one computes the Cartesian product of the subtree solution sets between nodes of the same color (in the same CMP solution), where each combination of the product corresponds to a sum between the components, and the second one builts in the union of the solutions provided by different colored subtrees (different CMP solutions), if more than one exists. This procedure is applied in each node of the structure (or, equivalently, in each line of the table). At the end of the execution, the set of solutions is contained in the *subtree solutions set* of the node n.

Example 5.2.2 – Continuing the previous example, after the of construction of the tree, the aggregation of solutions starts. Remark that each node r represents the equation $C_p^1 \odot \mathring{\mathbb{X}} = C_q^r$. According to Equation (5.2), the single-cycle solution is $C^1_{\frac{q}{p}\cdot r}$ iff $\gcd(p, \frac{q}{p} \cdot r) = r$ and $\mathrm{lcm}(p, \frac{q}{p} \cdot r) = q$. For example, for $r = 3$ one finds $\mathring{\mathbb{X}} = C_3^1$.

Node	CMP solutions	Node solution	Subtree solutions set
6	[3,3][2,2,2]	C_6^1	
3	[2,1]	C_3^1	
2	[1,1]	C_2^1	
1	$\emptyset$	C_1^1	

Figure 5.5 – Table status after the calculation of the single-cycle solutions.

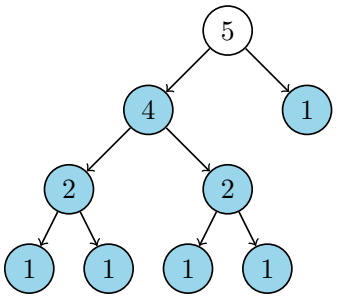
Moreover, for each equation $C_p^1 \odot \mathring{\mathbb{X}} = C_q^r$, one needs to compute the solutions represented in the subtree that has root r. To find all the solutions for the current node, it is necessary to take the Cartesian product of the solutions sets of nodes in the subtrees of the same color and then the union of the solution sets of different CMP solutions. For example, considering the node 3, its subtree solutions set is composed by: the node solution C_3^1, and the solutions coming from the subtree solutions set of 2 and 1 (*i.e.* $C_1^1 \oplus C_2^1$ or $C_1^1 \oplus C_1^2$). All the solutions can be found in Table 5.6.

Node	CMP solutions	Node solution	Subtree solutions set
6	[3,3][2,2,2]	C_6^1	$\{C_6^1, C_3^2, C_1^1 \oplus C_2^1 \oplus C_3^1, C_3^1 \oplus C_1^3, C_2^1 \oplus C_1^4, C_1^6, C_2^3, C_1^2 \oplus C_2^2\}$
3	[2,1]	C_3^1	$\{C_3^1, C_1^1 \oplus C_2^1, C_1^3\}$
2	[1,1]	C_2^1	$\{C_1^2, C_2^1\}$
1	$\emptyset$	C_1^1	$\{C_1^1\}$

Figure 5.6 – Final data-structure storing the result of the CMP execution.

The impossibility of an equation can be detected with a subtree solutions set empty. This happens if there exists a node (table line) such that there is no single solution and the subtree cannot represent valid solutions. Moreover, in this case, another valid independent subspace of solutions does not exist. If this property is propagated from a node to the root, then the equation has no solutions.

Example 5.2.3 – Consider the equation $C_2^1 \odot \mathring{\mathbb{X}} = C_4^5$. In the first phase, the algorithm enumerates all the divisors $\{4, 2, 1\}$ and constructs the corresponding CMP-tree like in the previous example. The method starts by decomposing 5 as $[4, 1]$ (which is the optimal decomposition). The procedure is then applied recursively always using the $div(q, n)$ as the set of coins to decompose each node. We obtain $5 = 4 + 1$, $4 = 2 + 2$ and $2 = 1 + 1$ as reported in Figure 5.7.



Node	CMP solutions	Node solution	Subtree solutions set
5	[4,1]	$\emptyset$	$\emptyset$
4	[2,2]	$\emptyset$	$\{C_4^2\}$
2	[1,1]	C_4^1	$\{C_4^1\}$
1	$\emptyset$	$\emptyset$	$\emptyset$

Figure 5.7 – Final data-structure storing the result of the CMP execution.

By performing the check, we discover that all the possible decompositions of 5 are represented in the current tree. This ends the building phase. The resulting tree is reported in Figure 5.7.

After this first phase of construction of the tree, the aggregation of solutions starts. Remark that if in the Cartesian product an empty set is involved, the result of the operation is the empty set. For $r = 5$, the final solution set is the empty set, since $r = 1$ is involved (with empty subtree solutions set) and we do not have a node solution. Then, the equation has no solution.

If we denote, for any $n, p, q \in \mathbb{N}^\star$, $\mathrm{EQ}_{p,q,n}$ the set of solutions of Equation (5.9) and $\mathrm{CTM}_{p,q,n}$ the set of solutions returned by the colored-tree method, then the following propositions ensure the soundness, completeness and termination of the colored-tree method.

Proposition 5.2.4 (Soundness)**.** *For all* $n, p, q \in \mathbb{N}^\star$, $CTM_{p,q,n} \subseteq EQ_{p,q,n}$.

Proof. Let us prove the soundness by induction on the depth of the tree from leaves to root. *Induction base*: if there is only one node, we know by Equation (5.2), that the solution found is the node solution, if it exists. *Induction hypothesis*: let us assume that we have $\mathrm{CTM}_{p,q,r} \subseteq \mathrm{EQ}_{p,q,r}$ for each node r into a certain level. Let us show that we can obtain real solutions at upper layer. *Induction step*: A solution of a node r' in the upper layer can be a single-cycle solution or a solution that is generated from the subtrees. In the first case, the solution will be also into $\mathrm{CTM}_{p,q,r'}$ according to Equation (5.2). In the second one, this came from a certain split, and this means that the solution comes from a Cartesian product between the set of solutions of some nodes (between nodes of the same color). Since all decompositions are represented in the structure, we know that the solution will be also into $\mathrm{CTM}_{p,q,r'}$. □

Proposition 5.2.5 (Completeness)**.** *For all* $n, p, q \in \mathbb{N}^\star$, $EQ_{p,q,n} \subseteq CTM_{p,q,n}$.

Proof. By contradiction, let us assume that there exists a solution that is in $\mathrm{EQ}_{p,q,n}$ but not in $\mathrm{CTM}_{p,q,n}$. This means that the colored-tree method does not return it. This implies that there exists a decomposition of n, which leads to the solution, such that this decomposition is not in the tree.

This is impossible since, an exhaustive check is performed to assure that all the decompositions are there. Therefore, all solutions are returned. □

Proposition 5.2.6 (Termination)**.** *The colored-tree method always terminates.*

This last proposition follows from the fact that the building phase always terminates since the dimension of the structure is related to the solutions set of the CMP. Moreover, the aggregation phase always terminates since it performs a finite number of operations per each node of the structure.

This method has hence allowed us to introduce the connection with the Change-Making problem. It represents the first method introduced to enumerate the solutions of a basic equation (5.9) [Dennunzio et al. (2020)] but, as pointed out, it has important weaknesses. Some important parameters that affect the computational time of the algorithm are the number of nodes in the colored tree (gives an idea of the number of Cartesian products and unions that are necessary to find the solutions set), and the check that ensures that all the decompositions are present (this is particularly time- and memory-consuming). We therefore present a more efficient technique based on Multi-valued Decision Diagrams.

5.2.3 A SB-MDD-based method

The previous method exploits a connection between EnumSOBFID and the well-known Change-making problem coupled with a completeness-check (running in exponential time) to explore the feasible solutions space. An efficient enumeration of solutions for Equation (5.4) is essential for solving more complex a-abstraction equations. In order to achieve this, we use MDD to boost up the enumeration. Recall that we are interested to find all the possible ways to generate C_q^n cycles starting with one cycle of length p. A cycle $C_{p'}^1$ generates C_q^r cycles in the right part iff r divides q, $p' = \frac{q}{p} \cdot r$, $\gcd(p, \frac{q}{p} \cdot r) = r$ and $\operatorname{lcm}(p, \frac{q}{p} \cdot r) = q$. From now on, a cycle $C_{p'}^1$ with the previous properties will be called *feasible* and r a *feasible divisor* of q.

Let $D_{p,q} = \{d_1, \ldots, d_t\}$ be the set of feasible divisors (*w.r.t.* Equation (5.9) of course). There is a solution to (5.9) iff there exist $y_1, \ldots, y_t$ such that $\sum_{i=1}^{t} d_i \cdot y_i = n$ (*i.e.* there is a solution to the Change-making problem for a total amount n and a coins system $D_{p,q}$). To solve EnumSOBFID we need to enumerate all solutions of the previous Change-making problem. At this point MDD come into play. We are going to use them to have a compact and handful representation of the set of all solutions to Equation (5.9). Based on what we saw in Chapter 4, let us define the structure level by level.

The MDD $M_{p,q,n}$, containing all solutions to Equation (5.9), is a labelled digraph $(N, E, \ell ab, val)$ with $Z = \lfloor \frac{n}{\min D_{p,q}} \rfloor + 1$ levels. In fact, the number of levels depends on the maximum number of coins one can use and on the fact of having a final level for the *tt* node. Graphically, each layer N_i in $M_{p,q,n}$ represents the choice of a new coin, and each node α (or $val(\alpha)$) represents the sum of the coins chosen from the *root* to the node. As a consequence, for this construction we do not need a variable order in input.

The first level of the structure contains the *root* (*i.e.* $N_1 = \{root\}$) with $val(root) = 0$. For any $d \in D_{p,q}$,

- if $d < n$, we create a new node $\beta \in N_2$ with $val(\beta) = d$ and $(root, \beta) \in E$ with $\ell ab((root, \beta)) = d$;
- if $d = n$, then $(root, tt) \in E$ with $\ell ab((root, tt)) = d$.

Note that since we need to enumerate all ways to reach the total amount n, $val(tt) = n$ and the basic equation has solution just if there is a path from the *root* to the *tt*.

Now, let us consider a level i (with $i \in \{2, \ldots, Z-2\}$) already defined. For any $\alpha \in N_i$ and any $d \in D_{p,q}$, we have:

- if $val(\alpha) + d < n$ and $\nexists \beta' \in N_{i+1}$ such that $val(\beta') = val(\alpha) + d$, a new node β is added to N_{i+1} (*i.e.* $N_{i+1} = N_{i+1} \cup \{\beta\}$) with $val(\beta) = val(\alpha) + d$, and $(\alpha, \beta) \in E$ with $\ell ab((\alpha, \beta)) = d$;
- if $val(\alpha) + d < n$ and $\exists \beta' \in N_{i+1}$ such that $val(\beta') = val(\alpha) + d$, then $(\alpha, \beta') \in E$ with $\ell ab((\alpha, \beta')) = d$;
- if $val(\alpha) + d = n$, then $(\alpha, tt) \in E$ with $\ell ab((\alpha, tt)) = d$.

To conclude the definition of the structure, let us consider the level $Z-1$. For any $\alpha \in N_{Z-1}$ and any $d \in D_{p,q}$, if $val(\alpha) + d = n$, then $(\alpha, tt) \in E$ with $\ell ab((\alpha, tt)) = d$.

According to this definition, the nodes of the structure (at the end) are the multiset $N = \sum_{i=1}^{Z} N_i$, where $N_1 = \{root\}$, $N_i \subseteq \{1, \ldots, n-1\}$ for $i \in \{2, \ldots, Z-1\}$, and $N_Z = \{tt\}$.

Note that in this structure we authorise direct arcs from a generic level to the final node *tt*. This choice arises from the fact that we are representing in each level the i-th coin chosen and the number of coins used to arrive at n is variable among the solutions. An alternative choice would be to insert a 0-value coin into the coin system. We choose the former to limit (even if only slightly) the size of the structure to be built and reduced.

Once $M_{p,q,n}$ is built and reduced according to the pReduction algorithm (Section 3.2), all the solutions of the considered basic equation can be computed. Each solution corresponds to the sequence of the edge labels of a path from *root* to *tt* consisting of possibly repeated values of $D_{p,q}$ with sum equal to n. To obtain the corresponding solution for $\mathring{\mathbb{X}}$, it is sufficient, for each label d belonging to a path from the *root* to *tt*, to consider the corresponding $C^1_{\frac{q}{p} \cdot d}$. It is important to emphasise that authorising direct arcs to the *tt* does not affect the functionality of the reduction procedure.

Remark that $M_{p,q,n}$ contains duplicated solutions, indeed, it contains all the permutations of a solution but according to Equation (5.9) different permutations lead to the same solution. For this reason, we impose a *symmetry breaking constraint*: for any node α (different from *tt*), let $e(\alpha)$ be the label of the incoming edge; the only allowed outgoing edges of α are those with labels of value smaller or equal to $e(\alpha)$. In this way all the paths of the MDD will be ordered, each solution is represented in the structure only once, and the size of the MDD will be smaller. An *SB-MDD* is an MDD which satisfies the symmetry breaking constraint. It should therefore be noted that the nodes belonging to a level are therefore multisets since two nodes with the same value but different incoming arcs can no longer be considered equivalent (in the construction of

the structure) since they give rise to different possible outgoing arcs. Obviously, if during the reduction phase, one finds that they have equivalent outgoing arcs they can be merged. Let us present the formal definition of the SB-MDD, and then let us illustrate the construction by an example.

The definition of the first level (i.e. the choice of the first coin) remains unchanged, but for each node β that is created we define $e(\beta) = d$. However, let us consider a level i (with $i \in \{2, \ldots, Z-2\}$) already defined. For any $\alpha \in N_i$ and any $d \in D_{p,q}$ where $d \leq e(\alpha)$, we have:

- if $val(\alpha) + d < n$ and $\nexists \beta' \in N_{i+1}$ such that $val(\beta') = val(\alpha) + d$ or $\exists \beta' \in N_{i+1}$ such that $val(\beta') = val(\alpha) + d$ and $e(\beta') \neq d$, a new node β is added to N_{i+1} (*i.e.* $N_{i+1} = N_{i+1} + \{\beta\}$) with $val(\beta) = val(\alpha) + d$ and $e(\beta) = d$, and $(\alpha, \beta) \in E$ with $\ell ab((\alpha, \beta)) = d$;
- if $val(\alpha) + d < n$ and $\exists \beta' \in N_{i+1}$ such that $val(\beta') = val(\alpha) + d$ and $e(\beta') = d$, then $(\alpha, \beta') \in E$ with $\ell ab((\alpha, \beta')) = d$;
- if $val(\alpha) + d = n$, then $(\alpha, tt) \in E$ with $\ell ab((\alpha, tt)) = d$.

To conclude the definition of the structure, for any $\alpha \in N_{Z-1}$ and any $d \in D_{p,q}$ where $d \leq e(\alpha)$, if $val(\alpha) + d = n$, then $(\alpha, tt) \in E$ with $\ell ab((\alpha, tt)) = d$.

According to this definition, the nodes of a SB-MDD structure (at the end) are the multiset $N = \sum_{i=1}^{Z} N_i$, where $N_1 = \{root\}$, N_i is a multiset of $\{1, \ldots, n-1\}$ for $i \in \{2, \ldots, Z-1\}$, and $N_Z = \{tt\}$.

Let us see through an example, the impact of choosing coins in an increasing way.

Example 5.2.4 – Consider the simple equation $C_2^1 \odot \mathring{\mathbb{X}} = C_6^6$. The set of divisors of q (smaller or equal to n) is $\{6, 3, 2, 1\}$. However, $D_{p,q} = \{1, 2\}$. Indeed, we have

$$\begin{aligned} r = 6, p' = 18 \rightarrow \gcd(2, 18) \neq 6 \text{ and } \operatorname{lcm}(2, 18) \neq 6 \\ r = 3, p' = 9 \rightarrow \gcd(2, 9) \neq 3 \text{ and } \operatorname{lcm}(2, 9) \neq 6 \\ r = 2, p' = 6 \rightarrow \gcd(2, 6) = 2 \text{ and } \operatorname{lcm}(2, 6) = 6 \\ r = 1, p' = 3 \rightarrow \gcd(2, 3) = 1 \text{ and } \operatorname{lcm}(2, 3) = 6 \end{aligned}$$

Let us look at the construction of the MDD both with and without the constraint to break the symmetries of the problem. Starting from the root, we evaluate which can be the first coin. As we start from $val(root) = 0$, both values contained in $D_{p,q}$ are candidate to be the first chosen coin. Two nodes are then created in the next level. One with a partial sum of value 2 and the other with a value of 1. Without the additional constraint of ordering paths, the same process is iterated on the two child nodes, hence creating three nodes in the next level (with values $4, 3$ and 2). In the case of an SB-MDD, on the other hand, only after choosing a coin of value 2, we can choose between a coin of value 2 or one of value 1. However, in the case that the first coin chosen was 1, we can only choose a second coin of the same value to have an ordered path. Hence, we obtain the same set of nodes but with different arcs. This idea is repeated until a total sum of 6 is reached, i.e. until *tt* is reached.

Figure 5.8 shows $M_{2,6,6}$ in its classic form (left) and in its SB-MDD form (right). Remark that in Figure 5.8 (left) many solutions are duplicated. For example, the solution $[2, 2, 1, 1]$ (in red) is represented (more than) twice.

Once $M_{2,6,6}$ is built and reduced, reading solutions[a] correspond to the sequence of labels of paths from *root* to *tt*: $\{[2, 2, 2], [2, 2, 1, 1], [2, 1, 1, 1, 1], [1, 1, 1, 1, 1, 1]\}$. Each coin corresponds to a cycle of a certain length p'. For example, the solution $[2, 2, 2]$ says that 6 is changed with 3 coins of value $r = 2$. Recalling that $p' = \frac{q}{p} \cdot r$ we can express the solution in term of dynamics graphs as C_6^3. Operating similarly for all the other solutions, we find $\{C_6^3, C_6^2 \oplus C_3^2, C_6^1 \oplus C_3^4, C_3^6\}$.

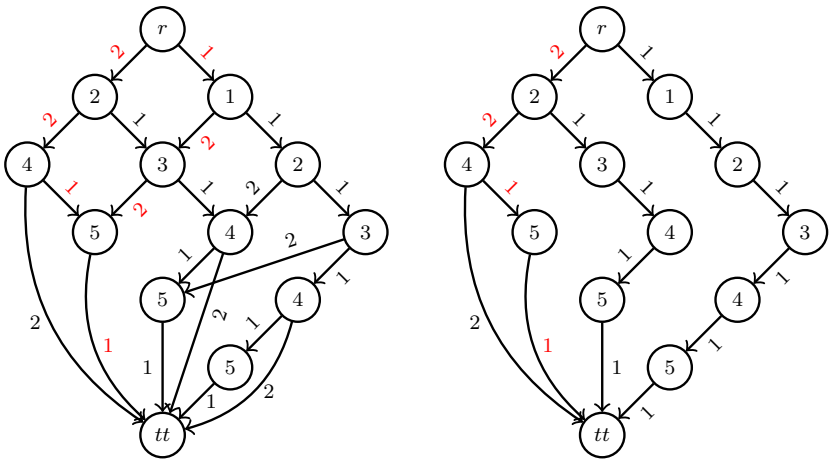


Figure 5.8 – The MDD representing all the solutions of $C_2^1 \odot \mathring{\mathbb{X}} = C_6^6$ in its classic form (left) and in its SB-MDD form (right).

[a] Solution reading consists of traversing the Z levels of the structure to enumerate the solutions represented. In general, one can upper bound the cost of reading to $O(S^{\#} \cdot Z)$ (where $S^{\#}$ is the number od solutions), but the actual cost also depends on the shape of the structure. For this reason, we will see later that we will leave solution reading as one of the final steps when solving a-abstraction equations.

The method detects the impossible instances by exploiting some properties:

- if p cannot divide q,
- if $D_{p,q}$ is an empty set (recall that if n is a feasible divisor, it is involved in $D_{p,q}$),
- if, after the reduction process, the SB-MDD structure has no valid path from the *root* to the *tt* node.

The following example shows how the method can detect the impossibility of an equation.

Example 5.2.5 – Consider the simple equation $C_2^1 \odot \mathring{\mathbb{X}} = C_4^5$. The set of divisors of q (smaller or equal to n) is $\{4, 2, 1\}$. However, $D_{p,q} = \{2\}$. Indeed, we have

$$\begin{aligned} r = 4, p' = 8 \rightarrow \gcd(2, 8) = 2 \text{ and } \operatorname{lcm}(2, 8) = 8 \\ r = 2, p' = 4 \rightarrow \gcd(2, 4) = 2 \text{ and } \operatorname{lcm}(2, 4) = 4 \\ r = 1, p' = 2 \rightarrow \gcd(2, 2) = 2 \text{ and } \operatorname{lcm}(2, 2) = 2 \end{aligned}$$

Figure 5.9 shows $M_{5,4,2}$ before the reduction procedure. The red part is deleted into the reduction phase. The SB-MDD has no paths from the *root* to *tt* node and this is one of the characterisations of an impossible instance.

Figure 5.9 – The SB-MDD (before reduction) for $C_2^1 \odot \mathring{\mathbb{X}} = C_4^5$. The red part will be deleted by the pReduction algorithm.

Algorithm 12: SB-MDD

Input : p, n, q integers
Output: M MDD with the solutions of $C_p^1 \odot \mathring{\mathbb{X}} = C_q^n$

$M[1] \leftarrow \{root\}$; ▷ M[i] are the nodes of M in level i
$val(root) \leftarrow 0$;
$e(root) \leftarrow \infty$;
$M[Z] \leftarrow \{tt\}$;
$val(tt) \leftarrow n$;
$D_{p,q} \leftarrow \emptyset$;
forall $r \in div(q, n)$ **do**
  if $\gcd(p, \frac{q}{p} \cdot r) = r$ *and* $\mathrm{lcm}(p, \frac{q}{p} \cdot r) = q$ **then**
    add r to $D_{p,q}$;
forall $i \in \{1, \ldots, Z-1\}$ **do**
  forall $\alpha \in M[i]$ **do**
    forall $d \in D_{p,q}$ *s.t.* $d \leq e(\alpha)$ **do**
      if $val(\alpha) + d < n$ *and* $i \neq Z-1$ **then**
        $\beta \leftarrow val(\alpha) + d$; ▷ NewArcNode verifies also $e(\beta)$
        NewArcNode$(M, i+1, \alpha, d, \beta)$;
      if $val(\alpha) + d == n$ **then**
        NewArc(M, Z, α, d, n);
return pReduce(M);

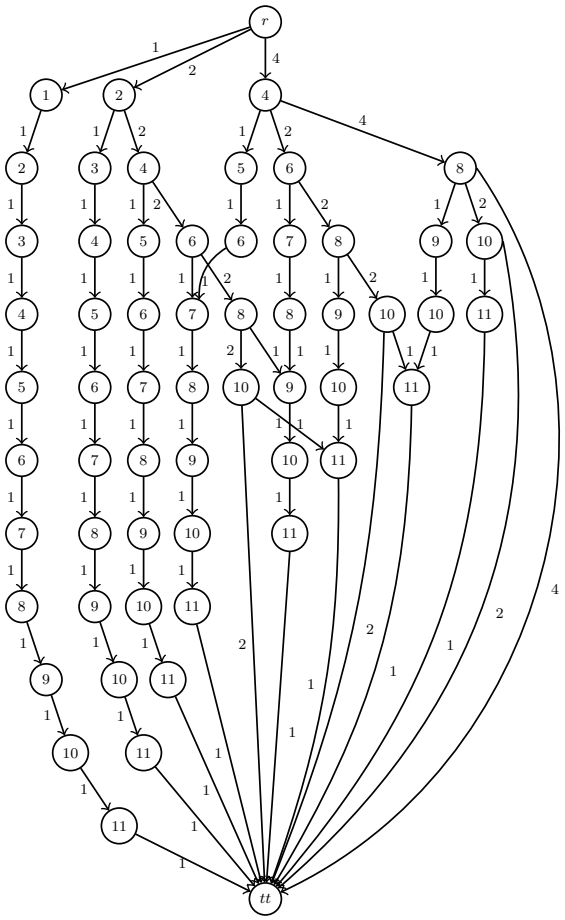


Figure 5.10 – The reduced SB-MDD that represents the solutions of $C_4^1 \odot \mathring{\mathbb{X}} = C_{12}^{12}$ with $D_{p,q} = \{4, 2, 1\}$. Note how nodes with equal values but different incoming arc labels give rise to different coins choices.

Experimental evaluations show how the new method can achieve interesting performance in time and memory. Concerning equations of the form $C_p^1 \odot \mathring{\mathbb{X}} = C_q^n$, in our experiments, we set $p = q$ since this grants the existence of at least a solution. Using the MDD, it is possible to outperform the Colored-Tree method (CTM) *w.r.t.* both memory and time. If we compare the dimension (in terms of nodes) of a colored-tree with the corresponding SB-MDD for a given equation, the second one is smaller. CTM presents some out of memory cases even for equations with n, q, and p smaller than 30 and memory limit of 30GB (see Figure 5.11 (left)). Using MDD, we solved equations with n, q, and p up to 100 without any out of memory case and only 6GB RAM limit (see Figure 5.11 (right)).

Analysing the time to solve equations with parameters up to 30, it turns out that the new technique is already faster than the previous one (see Tables 5.1, 5.2, and 5.3). The reason is that CTM requires a time consuming check procedure to ensure the completeness of the solutions which is not necessary in the MDD case.

Due to too high memory and time costs, CTM is unsuitable to solve simple equations coming from contractions steps. The new method [Formenti et al. (2021)] fixes these issues allowing the

introduction of a complete pipeline to solve abstractions over the asymptotic behaviour of generic polynomial equations.

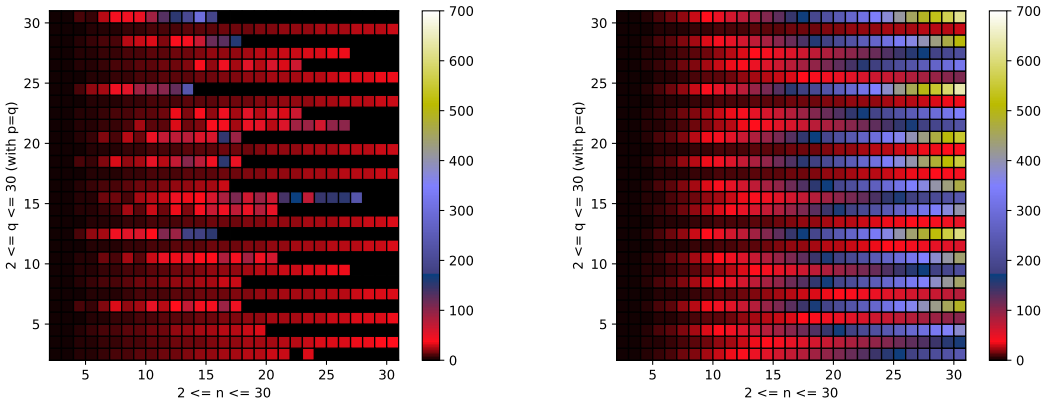


Figure 5.11 – The number of nodes for the colored-tree with memory limit of 30GB (left) and for the SB-MDD with memory limit of 4GB (right) in the case of equation for type $C_q^1 \odot \mathring{\mathbb{X}} = C_q^n$. Remark that the black square in the rightmost part of the left diagram are out of memory cases.

q,p \ n	2	3	4	5	6	7	8	9	10
2	56|0	56|0	54|0	54|0	53|0	54|0	54|0	54|0	53|0
3	54|1	54|0	55|1	54|0	55|1	54|0	55|1	54|0	55|1
4	55|2	54|1	56|2	53|0	57|2	54|0	55|2	55|1	55|2
5	55|2	56|2	59|2	54|0	61|30	54|0	58|2	55|3	56|2
6	58|2	56|2	60|2	58|1	63|9	55|0	60|2	56|2	62|2
7	60|3	58|2	63|19	56|1	78|21	54|0	63|29	60|2	61|2
8	63|2	59|2	96|10	60|2	107|20	56|1	97|9	61|2	65|2
9	66|3	60|3	106|21	57|2	153|22	57|1	168|21	60|3	76|20
10	84|3	62|2	140|11	58|2	185|21	57|2	369|11	70|2	120|17

Table 5.1 – Computation times (millisec) of CTM (left) and MDD (right) over different input parameters. Symbols '-' represent out of memory cases.

q,p \ n	11	12	13	14	15	16	17	18	19	20
11	55|0	824|25	55|0	116|3	74|21	406|22	55|0	1071|22	57|0	1334|23
12	58|1	17678|26	57|0	132|4	88|21	4105|12	56|0	6022|22	56|0	3672|22
13	61|2	177894|27	56|0	246|21	92|21	4163|27	55|0	5967|24	56|0	3332|24
14	59|2	1277979|26	61|1	900|11	116|22	19895|24	56|0	27381|27	56|0	96997|26
15	60|2	-|28	60|2	3721|22	169|22	19711|25	56|0	637457|26	59|0	419000|25
16	62|2	-|29	61|2	19900|12	502|23	-|13	57|0	1185947|26	60|0	759365|26
17	62|2	-|30	62|2	25908|24	554|23	-|26	57|0	-|27	57|0	-|27
18	64|2	-|46	62|2	164167|13	1102|22	-|26	61|2	-|28	61|0	-|30
19	66|2	-|32	63|2	226315|25	950|24	-|27	62|2	-|34	57|0	-|39
20	68|2	-|32	65|2	1707299|25	2749|24	-|16	63|2	-|31	62|2	-|29

Table 5.2 – Computation times (millisec) of CTM (left) and MDD (right) over different input parameters. Symbols '-' represent out of memory cases.

q,p \ n	21	22	23	24	25	26	27	28	29	30
21	2712|23	343542|26	60|0	-|35	95|4	389971|7	2034|12	-|28	58|0	-|37
22	23399|24	-|14	62|0	-|36	103|4	381929|7	2711|24	-|30	59|0	-|35
23	27430|24	-|27	59|0	-|38	134|4	-|7	2712|24	-|31	59|0	-|37
24	149296|25	-|16	64|2	-|39	149|4	-|8	20641|14	-|42	59|0	-|41
25	162413|25	-|28	65|2	-|40	160|4	-|26	24632|24	-|34	59|0	-|39
26	212277|25	-|16	66|2	-|42	403|5	-|15	24177|25	-|33	60|0	-|40
27	-|25	-|27	69|2	-|45	454|4	-|33	-|15	-|34	59|0	-|45
28	-|26	-|17	70|2	-|47	488|5	-|16	-|25	-|35	60|0	-|44
29	-|28	-|28	69|2	-|66	506|5	-|28	-|27	-|36	61|1	-|46
30	-|26	-|27	69|3	-|51	838|6	-|17	-|17	-|38	65|2	-|48

Table 5.3 – Computation times (millisec) of CTM (left) and MDD (right) over different input parameters. Symbols '-' represent out of memory cases.

5.3 An MDD-based pipeline to solve a-abstraction equations

We have shown how MDD can be used to compute the solutions set of basic equations. In this section and the next one, we introduce the missing steps to obtain a pipeline to enumerate the solutions of Equations (5.4), *i.e.* to validate hypotheses over the long-term behaviour of DDS. Recall that the goal is the enumeration of the solutions. The pipeline consists in the following steps:

- simplification of an Equation (5.4) to an Equation (5.5) as explained in Section 5.1;
- identification and resolution of the necessary equations (*i.e.* basic equations with at least a solution);
- enumeration of the contractions steps (with an MDD);
- computation of the solutions of each System (5.7) with a particular technique to compute the intersection between SB-MDD;
- computation of the $\mathring{X}$ values for all $\mathring{X}^w$.

The algorithmic pipeline illustrated in Figure 5.12 just performs all these tasks.

5.3.1 Necessary equations

As we have already seen, to solve a generic equation over a-abstractions, we need to enumerate the solutions of a finite number of systems obtained with the contraction steps (Systems (5.7)). Each system is based on a finite number of basic equations. Since many contraction steps and a finite (but potentially large) number of basic equations have to be solved, the pipeline is designed in order that first of all the basic equations admitting solution are identified. In this way, Systems (5.7) involving basic equations without solutions are avoided, or, in other words, only feasible Systems (5.7) generated by contraction steps are considered. Moreover, note that a single basic equation appears several times while searching for all Systems (5.7) solutions.

Therefore, the method starts with the computation of the set of basic equations that can be involved in the result of a contraction step. This amounts to compute all the SB-MDD $M_{p_{z,i}\,,\,p_j\,,\,\frac{n}{n_{z,i}}}$ with $p_{z,i} \in \{p_{1,1}, p_{1,2}, \ldots, p_{m,\ell_m}\}$, $p_j \in \{p_1, \ldots, p_{l_B}\}$, for all $n \in \{1, \ldots, n_j\}$. Recall that $\frac{n}{n_{z,i}}$ comes from Equation (5.8). In this way, even if a basic equation is involved in many contraction

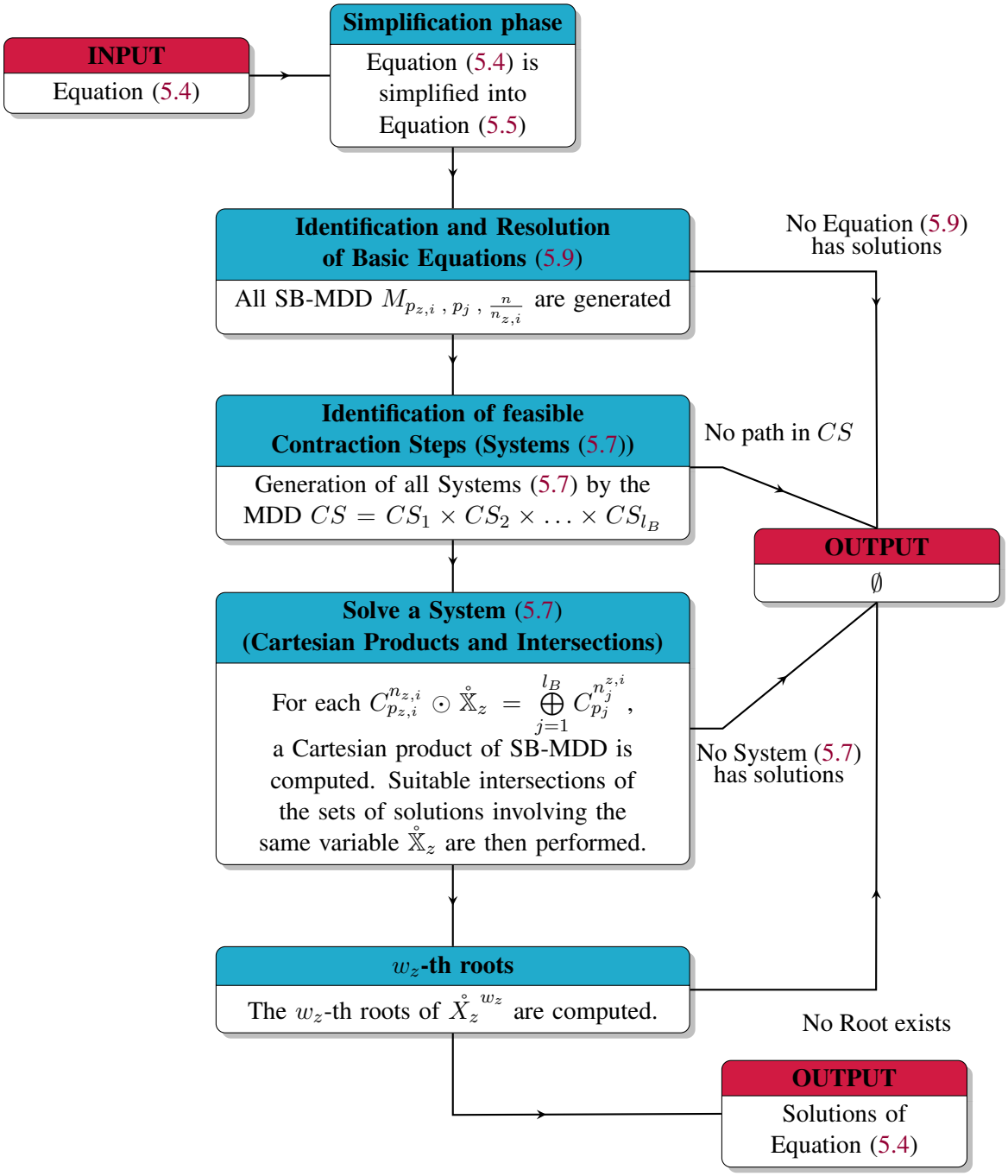


Figure 5.12 – The MDD-based algoritmic pipeline for solving a-abstraction equations.

steps, it is solved only once. Then, there are $\sum_{z=1}^{m} \ell_z \cdot \sum_{j=1}^{l_B} n_j$ potential necessary equations for an Equation.

An Equation (5.9) with at least one solution is called *necessary equation*. It is important to notice that it is not necessary to explore an SB-MDD to decide if a solution exists. Indeed, by construction, an SB-MDD exists if and only if there is at least one path from the *root* to the *tt* after the reduction process.

Example 5.3.1 – Consider the following equation:

$$C_4^1 \odot \mathring{\mathbb{X}}_1 \oplus C_2^1 \odot \mathring{\mathbb{X}}_2 = C_2^4 \oplus C_4^4 \oplus C_6^7 \oplus C_{12}^7$$

with $m = 2$, $l_B = 4$. There are 44 distinct basic equations.

$$C_4^1 \odot \mathring{\mathbb{X}}_1 = C_2^{n_1^{1,1}} \text{ with } n_1^{1,1} \in \{1,2,3,4\},$$

$$C_2^1 \odot \mathring{\mathbb{X}}_2 = C_2^{n_1^{2,1}} \text{ with } n_1^{2,1} \in \{1,2,3,4\},$$

$$C_4^1 \odot \mathring{\mathbb{X}}_1 = C_4^{n_2^{1,1}} \text{ with } n_2^{1,1} \in \{1,2,3,4\},$$

$$C_2^1 \odot \mathring{\mathbb{X}}_2 = C_4^{n_2^{2,1}} \text{ with } n_2^{2,1} \in \{1,2,3,4\},$$

$$C_4^1 \odot \mathring{\mathbb{X}}_1 = C_6^{n_3^{1,1}} \text{ with } n_3^{1,1} \in \{1,2,3,4,5,6,7\},$$

$$C_2^1 \odot \mathring{\mathbb{X}}_2 = C_6^{n_3^{2,1}} \text{ with } n_3^{2,1} \in \{1,2,3,4,5,6,7\},$$

$$C_4^1 \odot \mathring{\mathbb{X}}_1 = C_{12}^{n_4^{1,1}} \text{ with } n_4^{1,1} \in \{1,2,3,4,5,6,7\},$$

$$C_2^1 \odot \mathring{\mathbb{X}}_2 = C_{12}^{n_4^{2,1}} \text{ with } n_4^{2,1} \in \{1,2,3,4,5,6,7\}.$$

However, only 27 of them are necessary equations. Indeed, besides the basic equations with $p = 4$ and $q \in \{2,6\}$ (which have no solution because q does not divide p), the following ones have no solutions: $C_2^1 \odot \mathring{\mathbb{X}}_2 = C_4^1$, $C_2^1 \odot \mathring{\mathbb{X}}_2 = C_4^3$, $C_2^1 \odot \mathring{\mathbb{X}}_2 = C_{12}^1$, $C_2^1 \odot \mathring{\mathbb{X}}_2 = C_{12}^3$, $C_2^1 \odot \mathring{\mathbb{X}}_2 = C_{12}^5$, and $C_2^1 \odot \mathring{\mathbb{X}}_2 = C_{12}^7$.

5.3.2 Contractions steps

Once the set of necessary equations has been computed, we create an MDD to enumerate all the contractions steps that must be taken into account to enumerate the solutions of an Equation (5.5). Such an MDD is $CS = CS_1 \times \ldots \times CS_{l_B}$, *i.e.*, the Cartesian product of l_B MDD, where each CS_j aims at providing, according to the set of the necessary equations, all the feasible ways by which the monomials can concur to form the n_j cycles of length p_j of the known term $\mathring{B}$. As a consequence, by definition, the whole MDD CS will provide all the feasible ways by which all the cycles of $\mathring{B}$ can be formed according to the necessary equations computed.

Each CS_j is a labelled digraph $(N_j, E_j, \ell ab_j, val_j)$ with $m+1$ layers (one for each monomial $C_{p_{z,i}}^{n_{z,i}} \odot \mathring{\mathbb{X}}_z$ from the left-hand side of Equation (5.5), and one final layer for the *tt* node). To define

the outgoing edges of any level, we associate each monomial (z,i) with the set $D_{p_{z,i},p_j} = \{d \in \mathbb{N} \mid 1 \leq d \leq n_j$ and $M_{p_{z,i}\,,\,p_j\,,\,\frac{d}{n_{z,i}}}$ corresponds to a necessary equation$\} \cup \{0\}$, *i.e.* the number of cycles of length p_j that the monomial is able to create. Now, we can define the structure level by level.

Consider $N_{j,(1,1)} = \{root\}$ the first layer of the MDD. For any $d \in D_{p_{1,1},p_j}$, we create a node in the next layer $N_{j,i'}$ where $i' = (1,2)$ iff $\ell_1 > 1$, and $i' = (2,1)$ otherwise. Then, $N_{j,i'} = N_{j,i'} \cup \{\beta\}$ where $val_j(\beta) = d$, and $(root, \beta) \in E_j$ with $\ell ab_j((root,\beta)) = d$.

Also for this MDD, the value $val_j(\alpha)$ associated with a node α of a path from *root* to *tt* is the number of cycles of length p_j formed by the monomials encountered on the subpath from *root* to α.

Now, let us consider a generic level $N_{j,(z,i)}$ (with $(z,i) \neq (m, \ell_m)$ such that $z \in \{1,\ldots,m\}$, $i \in \{1,\ldots,\ell_z\}$) already created, and its following layer $N_{j,i'}$ where $i' = (z, i+1)$ iff $i < \ell_z$, and $i' = (z+1, 1)$ otherwise. For any $\alpha \in N_{j,(z,i)}$ and any $d \in D_{p_{z,i},p_j}$:

- if $val_j(\alpha) + d \leq n_j$ and $\nexists \beta' \in N_{j,i'}$ such that $val_j(\beta') = val_j(\alpha) + d$, we create a new node β in $N_{j,i'}$ (*i.e.* $N_{j,i'} = N_{j,i'} \cup \{\beta\}$) with $val_j(\beta) = val_j(\alpha) + d$, and $(\alpha,\beta) \in E_j$ with $\ell ab_j((\alpha,\beta)) = d$;
- if $val_j(\alpha) + d \leq n_j$ and $\exists \beta' \in N_{j,i'}$ such that $val_j(\beta') = val_j(\alpha) + d$, then $(\alpha,\beta') \in E_j$ with $\ell ab_j((\alpha,\beta')) = d$.

To conclude the definition of the structure, let us consider the level $N_{j,(m,\ell_z)}$. For any node $\alpha \in N_{j,(m,\ell_z)}$ and any $d \in D_{p_{m,\ell_z},p_j}$, if and only if $val_j(\alpha) + d = n_j$, $(\alpha, tt) \in E_j$ with $\ell ab_j((\alpha,tt)) = d$ and $tt \in N_{j,(m+1,1)}$.

According to this definition, the nodes of an MDD CS_j are the multiset $N_j = (\sum_{z\in\{1,\ldots,m\}\ \ i\in\{1,\ldots,\ell_z\}} N_{j,(z,i)}) + N_{j,((m+1),1)}$ where $N_{j,(1,1)} = \{root\}$, $N_{j,((m+1),1)} = \{tt\}$, and all others $N_{j,(z,i)} \subseteq \{0,\ldots,n_j\}$.

We stress that any edge outgoing from vertexes of the level (z,i) represents the cycles of length p_j that the monomial $C^{n_{z,i}}_{p_{z,i}} \odot \mathring{\mathbb{X}}_z$ can contribute to form together with the monomials corresponding to the other edges encountered on a same path from *root* to *tt*. Then, it is just the number $n_j^{z,i}$. Remark that a label 0 means that the monomial is not involved in the generation of the cycles of length p_j and that the sum of the labels of each path from the *root* to the *tt* node will be equal to n_j, because n_j cycles of length p_j must be generated.

Example 5.3.2 – Resuming from the previous example, let us illustrate the construction of one CS_j. Let us consider $j = 2$, or, in other words, the MDD providing all the possible ways by which the two monomials of the given equation can concur to form C^4_4. Thus, CS_2 has 3 levels, one for each monomial and one for the final terminal node. Any edge outgoing from a level represents the number of cycles of length 4 that the monomial corresponding to that level can contribute to form. The first level, corresponding to the monomial $C^1_4 \odot \mathring{\mathbb{X}}_1$, only contains the node *root* (which with its value at 0 represents the fact that no cycle of length 4 has yet been generated). According to the necessary equations defined $p = 4$ and $q = 4$ (see Example 5.3.1), the first monomial is able by itself to form $n_2^{1,1}$ cycles of length 4 where $n_2^{1,1} \in \{1,2,3,4\}$. Regarding the second monomial, it is able by itself to form either $n_2^{2,1} = 2$ or $n_2^{2,1} = 4$ cycles of length 4. As Figure 5.13 shows, the MDD CS_2 also represents the cases $n_2^{1,1} = 0$ and/or $n_2^{2,1} = 0$, *i.e.*, where at least one of the two monomials does not contribute to the generation of

such cycles at all. Any path from *root* to *tt* provides a feasible way by which the two monomials concur to form $n_2 = 4$ cycles of length $p_2 = 4$.

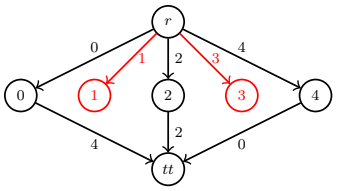


Figure 5.13 – The MDD CS_2 represents all the possible ways by which, according to the set of necessary equations, the two monomials can concur to form C_4^4. The red part is deleted by the pReduction procedure. The value $val(\alpha)$ associated with each node α is also reported inside the nodes. Recall that $val(root) = 0$ and $val(tt) = n_j$.

Figure 5.14 illustrates the MDD $CS = CS_1 \times CS_2 \times CS_3 \times CS_4$ associated with the given equation and obtained by stacking each CS_j on top of CS_{j+1}. Any path from the *root* to the terminal node of CS represents a possible way by which the monomials of the left-hand side of the given equation can concur to form all the cycles of its known term.

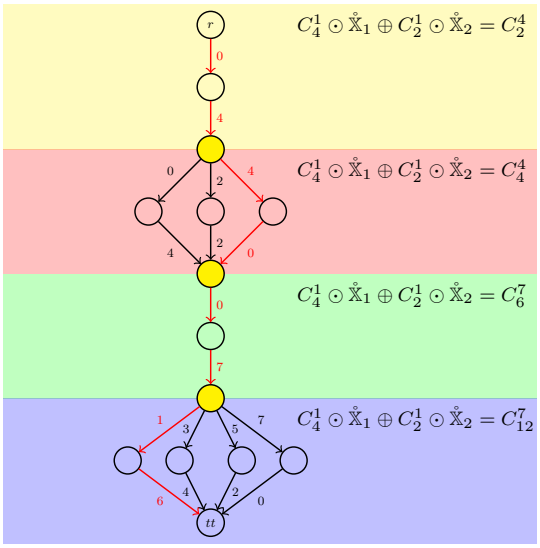


Figure 5.14 – The MDD CS represents all the feasible ways by which, according to the set of necessary equations, the monomials of the equation can concur to form its right-hand side. According to the Cartesian product of MDD, the yellow nodes are at the same time the *tt* node of a CS_j and the *root* node of CS_{j+1}. The four MDD are depicted by different colours (the red MDD corresponds to that from Figure 5.13). In each CS_j, the first (resp., second) level corresponds to the monomial $C_4^1 \odot \mathring{\mathbb{X}}_1$ (resp., $C_2^1 \odot \mathring{\mathbb{X}}_2$). The values $val(\alpha)$ associated to nodes are omitted for simplicity.

In conclusion, let us recall that each path from the *root* of CS to its final node represents a system obtainable with the contraction steps which does not contain impossible basic equations. For example, the path highlighted in red in Figure 5.14 corresponds to the following system:

$$\begin{cases} C_4^1 \odot \mathring{\mathbb{X}}_1 & = C_4^4 \oplus C_{12}^1 \\ C_2^1 \odot \mathring{\mathbb{X}}_2 & = C_2^4 \oplus C_6^7 \oplus C_{12}^6 \end{cases}.$$

At this point, we have now enumerated all the systems obtainable with the contraction steps (not containing impossible basic equations). As we have anticipated, enumerating the solutions of these systems corresponds to enumerating the solutions of the original equation over a-abstractions. However, we must now understand for each system how one can find all its possible solutions. This is the matter of the next section.

5.3.3 Solving a system of equations

Each path from the *root* to *tt* in CS corresponds to the result of a contraction step, *i.e.* a System (5.7). Solving any equation

$$C_{p_{z,i}}^{n_{z,i}} \odot \mathring{\mathbb{X}}_z = \bigoplus_{j=1}^{l_B} C_{p_j}^{n_j^{z,i}}$$

of a System (5.7) means computing the Cartesian product among the solutions of the l_B basic equations involved. In its turn, according to Section 5.2.3, solutions to a basic equation are represented by a SB-MDD. Therefore, we can be perform a Cartesian product between SB-MDD. As usual, such a Cartesian product is obtained by stacking the SB-MDD on top of each other. We will call *SB-Cartesian MDD* an MDD resulting from a Cartesian product between SB-MDD. Remark that, however, an SB-Cartesian MDD is not a SB-MDD. In fact, although it is satisfied by each of its component SB-MDD, the order constraint among the edge labels of any path from the root to the terminal node of an SB-Cartesian MDD does not necessarily hold (see Figure 5.15).

Example 5.3.3 – Consider the equation $C_2^1 \odot \mathring{\mathbb{X}}_2 = C_2^4 \oplus C_6^7 \oplus C_{12}^6$ where $D_{p,q} = D_{2,4} = \{1, 2\}$, $D_{2,6} = \{1, 2\}$, and $D_{2,12} = \{2\}$. Recall that each $d \in D_{p,q}$ corresponds to a $C_{p'}^1$ such that $p' = \frac{q}{p} \cdot d$. Then, the solutions of the equation are contained in the SB-Cartesian MDD of Figure 5.15 and they are:

$$\{C_1^4 \oplus C_3^7 \oplus C_{12}^3, C_1^4 \oplus C_3^5 \oplus C_6^1 \oplus C_{12}^3, C_1^4 \oplus C_3^3 \oplus C_6^2 \oplus C_{12}^3,$$
$$C_1^4 \oplus C_3^1 \oplus C_6^3 \oplus C_{12}^3, C_1^2 \oplus C_2^1 \oplus C_3^7 \oplus C_{12}^3, C_1^2 \oplus C_2^1 \oplus C_3^5 \oplus C_6^1 \oplus C_{12}^3,$$
$$C_1^2 \oplus C_2^1 \oplus C_3^3 \oplus C_6^2 \oplus C_{12}^3, C_1^2 \oplus C_2^1 \oplus C_3^1 \oplus C_6^3 \oplus C_{12}^3, C_2^2 \oplus C_3^7 \oplus C_{12}^3,$$
$$C_2^2 \oplus C_3^5 \oplus C_6^1 \oplus C_{12}^3, C_2^2 \oplus C_3^3 \oplus C_6^2 \oplus C_{12}^3, C_2^2 \oplus C_3^1 \oplus C_6^3 \oplus C_{12}^3\}$$

This example shows how solutions of an equation $C_{p_{z,i}}^{n_{z,i}} \odot \mathring{\mathbb{X}}_z = \bigoplus_{j=1}^{l_B} C_{p_j}^{n_j^{z,i}}$ are contained into a SB-Cartesian MDD.

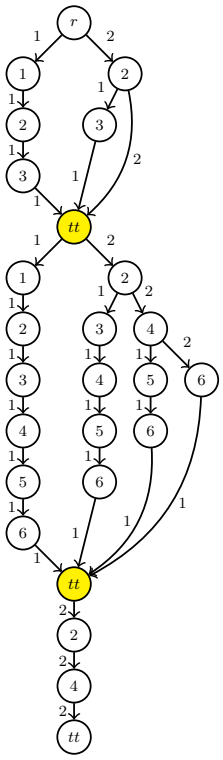


Figure 5.15 – SB-Cartesian MDD of equation $C_2^1 \odot \mathring{\mathbb{X}}_2 = C_2^4 \oplus C_6^7 \oplus C_{12}^6$. From top to bottom we find the SB-MDD of the equations: $C_2^1 \odot \mathring{\mathbb{X}}_2 = C_2^4$,$C_2^1 \odot \mathring{\mathbb{X}}_2 = C_6^7$, and $C_2^1 \odot \mathring{\mathbb{X}}_2 = C_{12}^6$. Nodes highlighted in yellow are those arising from the merge of a *root* node with a *tt* node.

Now that we know the solutions of each equation of a System (5.7), we need to compute the intersection between the different sets of solutions found for the same variable $\mathring{\mathbb{X}}_z$. Let us point out that at most ℓ_z equations of the system contain $\mathring{\mathbb{X}}_z$ with $z \in \{1, \ldots, m\}$, and that m is the number of different variable-power pairs that appear in the original equation. Here, the difficulty arises from the fact that each monomial containing $\mathring{\mathbb{X}}_z$ may be responsible for the generation of cycles of one or more different lengths. In the case of a monomial generating cycles all of the same length, the solutions are represented in an SB-MDD, but in the case of different lengths, we are dealing with an SB-Cartesian MDD. Hence, the intersection must be calculated between MDD that may at times be normal SB-MDD and at times SB-Cartesian MDD.

Notice that the classic algorithm to perform the intersection over MDD (presented in Section 3.3) cannot be used if the goal is the intersection between MDD issued by a Cartesian product (*i.e.* SB-Cartesian MDD) because the result depends on the order of the structures (see Figure 5.16). For this reason, we propose here a new algorithm to perform this task independently from the order.

Example 5.3.4 – This example shows the problem of the intersection algorithm on SB-Cartesian MDD. Consider the two SB-Cartesian MDD in Figure 5.16. The two structures contain exactly the same set of solutions. It is only the Cartesian product that is performed in a different order. By applying the classical top-down intersection algorithm (keeping just

common edges level by level), the only common solution is hence the red one. This is because the algorithm depends on the order in which the variables are represented in the structure.

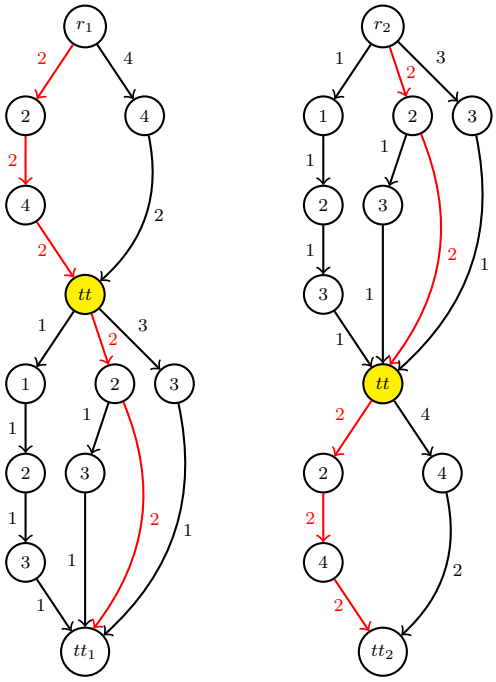


Figure 5.16 – Two SB-Cartesian MDD representing the same set of solutions.

SB-Cartesian Intersection Consider a set $\overline{M}$ of MDD in which some are SB-Cartesian MDD and others are not. We propose an algorithm (Algorithm 13) which computes the intersection of all the elements in $\overline{M}$. Our algorithm needs to be started with an initial set of candidate solutions $\overline{\mathcal{S}}$ (*initial guess*). If $\overline{M}$ contains at least a simple SB-MDD (*i.e.* one which is not a SB-Cartesian MDD), then $\overline{\mathcal{S}}$ is the set of solutions contained in the MDD resulting from the classical intersection between all SB-MDD in $\overline{M}$; otherwise $\overline{\mathcal{S}}$ is the set of solutions read in an arbitrarily chosen SB-Cartesian MDD of $\overline{M}$. We emphasise that the fact that SB-MDD have arcs directed towards the terminal node is not a problem when computing the intersection as presented in Section 3.3.

Using $\overline{\mathcal{S}}$, we will compute the intersection between the remaining SB-Cartesian MDD. The idea is to search all solutions into each SB-Cartesian MDD and update each time the remaining solutions. In fact, each candidate solution is ordered and recursively searched in a SB-Cartesian MDD in $\overline{M}$ (Algorithms 14 and 15). If it is not found, then it is removed from the set of candidate solutions. Given a SB-Cartesian element M of $\overline{M}$, a candidate solution $\mathcal{S}$ of $\overline{\mathcal{S}}$ is validated if it possible to visit each SB-MDD involved in M using a subset of the elements of the solution. Starting from the biggest elements of $\mathcal{S}$, we try to find a path from the *root* to the first *tt* node (recall that we are visiting a SB-Cartesian MDD). We proceed in this way because the paths of each SB-MDD are ordered. After having gone through the first SB-MDD, we need to recursively repeat the procedure with the remaining elements of $\mathcal{S}$.

Algorithm 13: SB-Cartesian Intersection

Input : $\overline{M}$, set of MDD (some SB-MDD and some SB-Cartesian MDD)
Output: $\overline{\mathcal{S}}$, solutions of the intersection in $\overline{M}$

```
1  Cartesian ← ∅;
2  Traditional ← ∅;
3  forall M ∈ M̄ do
4      if M is a SB-Cartesian MDD then
5          add M to Cartesian;
6      else
7          add M to Traditional;
8  S̄ ← ∅;
9  if |Traditional| > 0 then
10     if |Traditional| == 1 then
11         S̄ ← Traditional[0].readSolutions();
12     else
13         MddIntersected ←ClassicIntersection(Traditional[0], Traditional[1]);
14         forall M ∈Traditional \ {Traditional[0], Traditional[1]} do
15             MddIntersected ←ClassicIntersection(MddIntersected, M);
16         S̄ ← MddIntersected.readSolutions();
17 else
18     S̄ ← Cartesian[0].readSolutions();
19     remove 0 from Cartesian;
20 if |S̄| ≠ 0 then
21     forall M ∈Cartesian do
22         CartesianSearch(S̄, M);
23 return S̄
```

Two special cases need our attention:

- a generic node in which it is not possible to find a common element between the remaining elements of a solution and the outgoing edges;
- a node (different from the final *tt* node) in which there are no more elements of $\mathcal{S}$ to compare with the outgoing edges.

In both cases, it is necessary to backtrack in the visited nodes until there is another possible outgoing edge that can be taken into consideration. Once the search arrives at the last SB-MDD of M, a linear search is performed with the remaining elements, and $\mathcal{S}$ is one of the solutions contained in M if there exists a path from the last *root* to the final *tt* node following the remaining elements of $\mathcal{S}$ (Algorithm 16). If the linear search fails, then we return to the first node with a different feasible outgoing edge. If no different feasible edges exist, then $\mathcal{S}$ is not a solution. The previous procedure is performed over each candidate solution of the initial guess.

We stress that in the worst case, for a given candidate solution, our algorithm explores only the subgraph of a SB-Cartesian MDD made of feasible edges.

Algorithm 14: CartesianSearch

Input : $\overline{\mathcal{S}}$ set of solutions, M a SB-Cartesian MDD
Output: $\overline{\mathcal{S}}'$ solutions involved in M

```
1 S̄' ← ∅;
2 forall S ∈ S̄ do
3     S.order();
4     find ←FindSolution(S, 0, M);
5     if find then
6         add S to S̄';
7 return S̄';
```

Algorithm 15: FindSolution

Input : $\mathcal{S}$ solution, $0 \leq i < |M|$, M a SB-Cartesian MDD
Output: true if $\mathcal{S} \in M$, false otherwise

```
1 return FindSolutionNode(S, M[i].root, i, M);
```

Algorithm 16: FindSolutionNode

Data: $\mathcal{S}'$ sub-solution, α node, $0 \leq i < |M|$, M a SB-Cartesian MDD
Result: true if $\mathcal{S}' \in M$, false otherwise

```
1  find ← false;
2  if α is not a tt node then
3      forall s ∈ S'.removeDuplicates() and find==FALSE do
4          if α has an outgoing edge with label s then
5              newSubSolution ← S' \ {s} ;
6              find ←FindSolutionNode(newSubSolution, α.children(s),i);
7  else
8      i ← i + 1;
9      if i== |M|−1 then
10         valid ←LinearSearch(S',root(M[i]));
11         if valid then return true ;
12     else
13         return FindSolution(S',i);
14 return find;
```

The presented approach introduces a way to compute the intersection between MDD with ordered paths. It is suitable for our needs, and its improvement constitutes an interesting future research direction. However, we need to precise an additional aspect concerning our application.

As explained above, an SB-MDD (corresponding to a simple equation) has labels based on the feasible divisors set (or feasible coins) and each label corresponds to a certain cycle $C^1_{p'}$. However,

two different coins of two different simple equations can correspond to the same $C^1_{p'}$ as well as the same coin may correspond to different cycles lengths for different simple equations. Therefore, when searching for a solution in a SB-Cartesian MDD, we must take into account these cases. In our application, two divisors/coins are considered equivalent if they correspond to the same $C^1_{p'}$ both by computing the classical intersection between MDD and the SB-Cartesian intersection.

Example 5.3.5 – Consider a set $\overline{M}$ containing the two SB-Cartesian MDD of Figure 5.17. Since there are only SB-Cartesian MDD, reading the set of solutions represented in the first MDD gives us the initial guess:

$$\overline{\mathcal{S}} = \{[4,2,2,2],[4,2,2,1,1],[1,1,1,1,2,2,2],[1,1,1,1,2,2,1,1]\}$$

Each solution is then searched in the second SB-Cartesian MDD M, hence the CartesianSearch($\overline{\mathcal{S}}, M$) method is called (Algorithm 14). Each solution is sorted in descending order to be searched in the structure using the FindSolution method (Algorithm 16). Let us take the solution $\mathcal{S} = [2,2,2,1,1,1,1]$ as an example.

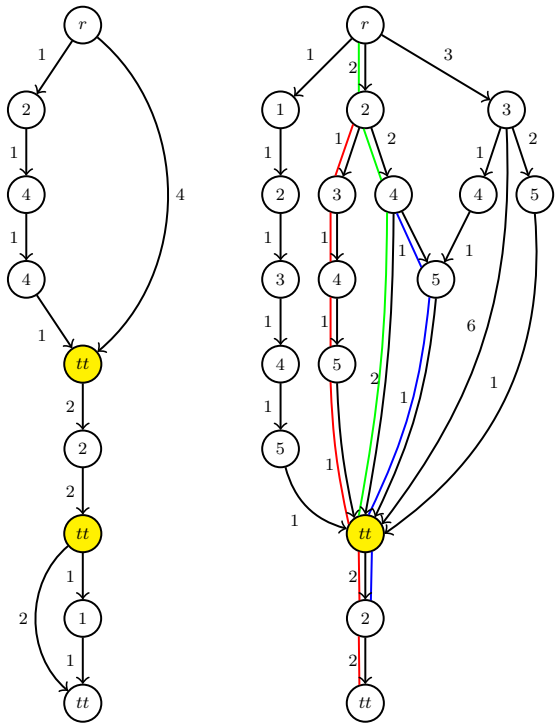


Figure 5.17 – Two SB-Cartesian MDD contained in a set $\overline{M}$.

Initially, the method tries to cross the first SB-MDD composing the SB-Cartesian MDD. Then, we search $\mathcal{S}' = [2,2,2,1,1,1,1]$ in the first (*i.e.* $i = 0$) SB-MDD of M (the SB-Cartesian MDD on the right in Figure 5.17) starting from the r root node. The biggest element s of $\mathcal{S}$ is 2. Hence, since the root has an outgoing arc with label value 2, the solution to be searched for is updated to $\mathcal{S}' = [2,2,1,1,1,1]$ and the procedure is iterated (*i.e.* FindSolutionNode($[2,2,1,1,1,1]$, 2, 0, M) where 2 is the node reachable from r by following the arc labelled 2). The process is iterated and brings us through the structure following

the path highlighted in green in Figure 5.17. Once the root of the second SB-MDD of M is reached, LinearSearch over $[1, 1, 1, 1]$ is called and hence false is returned. Going back in the recursive calls, another possible value for s is then considered. The first choice is to consider $s = 1$ and to take the arc outgoing from the node of value 4 in the third level of the structure. FindSolutionNode($[2, 1, 1, 1]$, 4, 0, M) is then calculated. The process is iterated and brings us through the structure following the path highlighted in blue in Figure 5.17. However, by reaching the node of value 2 of the second SB-MDD we find ourselves in the situation where $\mathcal{S}' = [1, 1]$ and the only arc exiting the node has a 2-value label. As a consequence, false is returned. Going back again, one tries to consider $s = 1$ and to take the arc outgoing from the node of value 2 in the second level of the structure. The process is iterated and brings us through the structure following the path highlighted in red in Figure 5.17. This leads us to the final node of the structure and thus to decide that $\mathcal{S} = [2, 2, 2, 1, 1, 1, 1]$ is one of the solutions contained in M.

Once a set of solutions is assigned to a $\mathring{\mathbb{X}}_z$ (which we recall to be the z-th variable-power pair), we need to compute the w_z-th root for each value of $\mathring{\mathbb{X}}_z$ (*i.e.*, $\mathring{X}_z$ such that $\mathring{X}_z^{\,w_z} = \mathring{\mathbb{X}}_z$). The root procedure is not a trivial step. Indeed, the inverse operations of sum and product are not definable in the commutative semiring of DDS. Therefore, we need an algorithmic technique to compute the result of the w-th root of $\mathring{\mathbb{X}}_z$. We will therefore present how the root of a a-abstraction can be calculated in the final part of this chapter.

The root procedure tries to calculate the w_z-root of all solutions found for a $\mathring{\mathbb{X}}_z$ and hence returns a set containing the roots found. Recall that each variable $\mathring{X}_z$ may appear in the original equation at different degrees. To conclude, it then becomes necessary to calculate an intersection between the solutions sets of the different roots calculated to obtain the set of solutions for each variable $\mathring{X}_z$.

We conclude this section with a few considerations about the performance of the pipeline. If the computation of the roots is not performed then everything depends on the number of monomials and the number of distinct sizes of cycles in the right-hand side of the equation. If these quantities grow also the number of contraction steps to consider grows. The solutions space is limited by considering only contraction steps that are feasible according to the necessary equations. For example, if we consider the equation $C_5^1 \odot \mathring{X}_2 \oplus C_4^1 \odot \mathring{X}_2^{\,2} \oplus C_3^1 \odot \mathring{X}_1^{\,2} \oplus C_2^1 \odot \mathring{X}_1^{\,3} = C_{10}^3 \oplus C_4^{68} \oplus C_3^9 \oplus C_6^9 \oplus C_{12}^{136}$, the number of contraction steps is $\approx 2,42 \cdot 10^{16}$, but only 6665400 are considered. Furthermore, each basic equation is solved only once and the solutions of the systems are calculated from the solutions of the basic equations. For each system, Cartesian product operations, classical and SB-Cartesian intersections between SB-MDD and SB-Cartesian MDD are required. The former have reduced computational and memory cost (see Chapter 3) and the latter operation is defined to limit the search space, and therefore its computational cost.

The pipeline introduced above is the first complete technique to solve equations and validate hypotheses over the long-term behaviour of dynamics graphs showing an interesting interaction between MDD and DDS, and adding one more item to the growing list of successful applications of MDD [Formenti et al. (2021)].

5.4 Roots over a-abstractions

We now deal with the problem of retrieving the value of each DDS $\mathring{X}_z$ once the possible values of $\mathring{X}_z^{w_z}$ have been computed. We are going to introduce the concept of w-th root in the semiring of DDS and provide an algorithm for computing the w-th roots of the a-abstractions of DDS.

First of all, let us formally define the notion of w-root of a general DDS.

Definition 5.4.1. Let $w \geq 2$ be a natural number. The w-th root of a DDS S is a DDS having w-th power equal to S.

Clearly, the a-abstraction of the w-th root of a DDS is the w-th root of the a-abstraction of that system. The goal is now to compute the w-th root of the a-abstraction of any DDS. Namely, for any given a-abstraction

$$C_{q_1}^{o_1} \oplus \ldots \oplus C_{q_t}^{o_t} \ ,$$

with $0 < q_1 < q_2 < \ldots < q_t$, we want to solve the equation

$$\mathring{X}^w = C_{q_1}^{o_1} \oplus \ldots \oplus C_{q_t}^{o_t} \ , \tag{5.10}$$

where the unknown is expressed as

$$\mathring{X} = C_{p_1}^{n_1} \oplus \ldots \oplus C_{p_l}^{n_l},$$

for some naturals $l, p_1, \ldots, p_l, n_1, \ldots, n_l$ (with $p_1 < \ldots < p_l$) to be determined. From now on, without loss of generality, we will assume $p_1 < \ldots < p_l$, and $q_1 < \ldots < q_t$.

Since providing a closed formula for $\mathring{X}$ seems quite complicated, we are going to compute the sets $C_{p_i}^{n_i}$ one by one starting from $i = 1$. Such a computation will be iteratively performed by comparing the $C_{q_1}^{o_1} \oplus \ldots \oplus C_{q_t}^{o_t}$ with the w-th power of the sum of sets $C_{p_i}^{n_i}$ found up to a certain i.

Proposition 5.4.1. *If $\mathring{X} = C_{p_1}^{n_1} \oplus \ldots \oplus C_{p_l}^{n_l}$ is a solution of the equation $\mathring{X}^w = C_{q_1}^{o_1} \oplus \ldots \oplus C_{q_t}^{o_t}$, then the following facts hold:*

- (i) *$l \leq t$ and $\{p_1, \ldots, p_l\} \subseteq \{q_1, \ldots, q_t\}$*
- (ii) *$p_1 = q_1$ and $p_2 = q_2$;*
- (iii) *n_1 is the integer solution of $\sqrt[w]{\frac{o_1}{q_1^{w-1}}}$;*
- (iv) *n_2 is the integer solution of*

$$\begin{cases} \frac{\sqrt[w]{q_2 o_2 + q_1 o_1} - \sqrt[w]{q_1 o_1}}{q_2} \in \mathbb{N}, & \text{if } \operatorname{lcm}(q_1, q_2) = q_2, \\ \sqrt[w]{\frac{o_2}{q_2^{w-1}}} \in \mathbb{N}, & \text{otherwise.} \end{cases} \tag{5.11}$$

Proof.

(i): According to Proposition 5.1.5, for each $i \in \{1, \ldots, l\}$, a set $C_{\lambda_l^*}^{o}$ with $\lambda_l^* = p_i$ appears in $\mathring{X}^w$ when the tuple $(k_1, \ldots, k_l)$ with $k_i = w$ and $k_{i'} = 0$ (for all $i' \neq i$) is involved in the sum. Hence, $\{p_1, \ldots, p_l\} \subseteq \{q_1, \ldots, q_t\}$ and $l \leq t$.

(ii): Since p_1 is the smallest value among all possible lcm λ_l^* from Proposition 5.1.5 and q_1 is the smallest among the lengths $q_1, \ldots, q_t$ of the cycles to be generated when the w-th power of $\mathring{X}$ is performed, it must necessarily hold that $p_1 = q_1$ in order that cycles of length q_1 are generated.

Moreover, since p_2 and q_2 follow in ascending order p_1 and q_1, respectively, and p_2 is also the successor of p_1 among all the above mentioned lcm, it must also hold that $p_2 = q_2$ in order that cycles of length q_2 are generated too.

(iii): It holds that $(C_{q_1}^{n_1})^w = C_{q_1}^{o_1}$, which, by Lemma 5.1.4, is equivalent to $C_{q_1}^{q_1^{w-1}n_1^w} = C_{q_1}^{o_1}$. This implies that $q_1^{w-1}n_1^w = o_1$ and, hence, n_1 is equal to $\sqrt[w]{\frac{o_1}{q_1^{w-1}}}$ iff integer.

(iv): If $\gcd(q_1, q_2) \neq q_1$, then $\mathrm{lcm}(q_1, q_2) > q_2$. Thus, when computing the w-th power of $\mathring{X}$, by Proposition 1.1.1, $C_{p_1}^{n_1}$ does not contribute to form $C_{q_2}^{o_2}$ and, necessarily, it holds that $(C_{q_2}^{n_2})^w = C_{q_2}^{q_2^{w-1}n_2^w} = C_{q_2}^{o_2}$. So, we get $q_2^{w-1}n_2^w = o_2$, the latter implying that $n_2 = \sqrt[w]{\frac{o_2}{q_2^{w-1}}}$ iff integer. Otherwise, it follows that $\mathrm{lcm}(q_1, q_2) = q_2$ and both $C_{p_1}^{n_1}$ and $C_{p_2}^{n_2}$ contribute to $C_{q_2}^{o_2}$. In particular, it holds that $(C_{p_1}^{n_1} \oplus C_{p_2}^{n_2})^w = C_{q_1}^{o_1} \oplus C_{q_2}^{o_2}$. By Proposition 1.1.1, one finds

$$(C_{p_1}^{n_1})^w \oplus \left(\bigoplus_{k=1}^{w-1} \binom{w}{k} (C_{p_1}^{n_1})^k \odot (C_{p_2}^{n_2})^{w-k} \right) \oplus (C_{p_2}^{n_2})^w = C_{q_1}^{o_1} \oplus C_{q_2}^{o_2}.$$

Since $(C_{p_1}^{n_1})^w = C_{q_1}^{o_1}$ and by Remark 5.1.2, the last equation can be seen as follows

$$C_{p_2}^{{p_2}^{w-1}{n_2}^w} \oplus \left(\bigoplus_{k=1}^{w-1} \binom{w}{k} C_{\mathrm{lcm}(p_1,p_2)}^{\frac{1}{\mathrm{lcm}(p_1,p_2)} \cdot {p_1}^i {n_1}^i {p_2}^{w-i} {n_2}^{w-i}} \right) = C_{q_2}^{o_2}.$$

Recalling that $\gcd(q_1, q_2) = q_1$, $\mathrm{lcm}(q_1, q_2) = q_2$, $p_1 = q_1$ and $p_2 = q_2$, the latter equality is true iff

$${q_2}^{w-1}{n_2}^w + \sum_{k=1}^{w-1} \binom{w}{k} {q_1}^i {n_1}^i {q_2}^{w-i-1} {n_2}^{w-i} = o_2 \ ,$$

i.e., once both sides are first multiplied by q_2 and then added to the term $(q_1 n_1)^w$, iff

$$(q_1 n_1 + q_2 n_2)^w = q_2 o_2 + (q_1 n_1)^w \ .$$

By (i) and (iii), we get

$$n_2 = \frac{\sqrt[w]{q_2 o_2 + q_1 o_1} - \sqrt[w]{q_1 o_1}}{q_2} \ .$$

□

This last proposition provides some information on the possible solution. Note that if it is not possible to find integer values for n_1 or n_2, this means that there is no $\mathring{X} = C_{p_1}^{n_1} \oplus \ldots \oplus C_{p_l}^{n_l}$ solution of the equation $\mathring{X}^w = C_{q_1}^{o_1} \oplus \ldots \oplus C_{q_t}^{o_t}$.

The following theorem explains how to compute n_{i+1} and p_{i+1} once $n_1, \ldots, n_i$ and $p_1, \ldots, p_i$ are also known.

Theorem 5.4.2. *Let $\mathring{X} = C_{p_1}^{n_1} \oplus \ldots \oplus C_{p_l}^{n_l}$ be a solution of the equation $\mathring{X}^w = C_{q_1}^{o_1} \oplus \ldots \oplus C_{q_t}^{o_t}$. For any fixed natural i with $2 \leq i < l$, if $n_1, \ldots, n_i$, $p_1, \ldots, p_i$ are known, $t' \in \{i, \ldots, t\}$, and $o'_1, \ldots, o'_{t'}$, $q'_1, \ldots, q'_{t'}$ are positive integers such that $(C_{p_1}^{n_1} \oplus C_{p_2}^{n_2} \oplus \ldots \oplus C_{p_i}^{n_i})^w = C_{q'_1}^{o'_1} \oplus C_{q'_2}^{o'_2} \oplus \ldots \oplus C_{q'_{t'}}^{o'_{t'}}$, then the following facts hold:*

(1) $p_{i+1} = q_\xi$, where $\xi = \min\{j \in \{1, \ldots, t\}$ with $q_j > p_i \mid q_j > q'_{t'} \vee (q_j = q'_{z'}$ for some $1 \leq z' \leq t'$ with $o'_{z'} < o_j)\}$;

(2) n_{i+1} *is the solution of*

$$n_{i+1} = \begin{cases} \dfrac{\sqrt[w]{q_\xi o_\xi + Q_i^*} - \sum_{j=1}^{i} p_j n_j}{q_\xi}, & \text{if } \operatorname{lcm}(p_1, \ldots, p_{i+1}) = p_{i+1}, \\ \dfrac{\sqrt[w]{q_\xi o_\xi + Q_i^{**}} - \sum_{e=1}^{j-1} p_{i_e} n_{i_e}}{q_\xi}, & \text{otherwise} \end{cases} , \tag{5.12}$$

where

$$Q_i^* = \sum_{\substack{k_1+\ldots+k_i=w \\ 0\le k_1,\ldots,k_i\le w \\ \lambda_i^*\neq p_{i+1}}} \binom{w}{k_1, \ldots, k_i} \prod_{t=1}^{i} (p_t n_t)^{k_t} ,$$

with λ_i^* *as in Proposition 5.1.5,*

$$Q_i^{**} = \sum_{\substack{k_{i_1}+\ldots+k_{i_{j-1}}=w \\ 0\le k_{i_1},\ldots,k_{i_{j-1}}\le w \\ \lambda_{i_{j-1}}^{**}\neq p_{i+1}}} \binom{w}{k_{i_1}, \ldots, k_{i_{j-1}}} \prod_{t=1}^{j-1} (p_{i_t} n_{i_t})^{k_{i_t}} ,$$

and, regarding Q_i^{**}*, the set* $\{i_1, \ldots, i_j\}$ *is the maximal subset of* $\{1, \ldots, i+1\}$ *such that* $i_1 < \ldots < i_j$*,* $i_j = i+1$*, and* p_{i_e} *divides* p_{i+1} *for each* $1 \le e \le j$ *(*i.e.*,* $\operatorname{lcm}(p_{i_1}, \ldots, p_{i_j}) = p_{i+1}$*), and where, for each* $1 \le e \le j$ *and for any tuple* $k_{i_1}, \ldots, k_{i_e}$*,* $\lambda_{i_e}^{**}$ *denotes the* lcm *of those* p_{i_ε} *with* $\varepsilon \in \{1, \ldots, e\}$ *and* $k_{i_\varepsilon} \neq 0$ *(while* $\lambda_{i_e}^{**} = 1$ *iff all* $k_{i_\varepsilon} = 0$*).*

Proof.

(1) We deal with the following two mutually exclusive cases *a)* and *b)*.

Case a): for some $j \in \{1, \ldots, t\}$ the following condition holds: there exists $z' \in \{1, \ldots, t'\}$ such that $q_j = q'_{z'}$ and $o'_{z'} < o_j$. This means that, when the w-th power is performed, the cycles $(C_{p_1}^{n_1} \oplus \ldots \oplus C_{p_i}^{n_i})$ give rise to a number $o'_{z'}$ of cycles of length $q'_{z'} = q_j$ where $o'_{z'}$ is lower than the number o_j of cycles of length q_j that are expected once the w-th power of the whole solution is computed. Consider the minimum among all the indexes j satisfying the above introduced condition. It is clear that ξ is just such a minimum and q_ξ is the minimum among the values q_j corresponding to those indexes j. Since by item (i) of Proposition 5.4.1, p_{i+1} comes from the set $\{q_1, \ldots, q_t\}$, and it is the successor of p_i, we get that p_{i+1} can be nothing but q_ξ, or, equivalently, $i+1 = \xi$. Indeed, according to Proposition 5.1.5, if cycles of length greater than q_ξ were added to $(C_{p_1}^{n_1} \oplus \ldots \oplus C_{p_i}^{n_i})$, instead of cycles of length q_ξ, they would give rise to cycles of greater length, preventing the generation of the missing cycles of length q_ξ.

Case b): there is no index $j \in \{1, \ldots, t\}$ such that $q_j = q'_{z'}$ and $o'_{z'} < o_j$. Similar arguments from case *a)* over the values q_j and the corresponding indexes j such that $q_j > q'_t$ lead to the conclusion that ξ is the minimum of such indexes, $i+1 = \xi$, and $p_{i+1} = q_\xi$.

(2) We deal with the following two mutually exclusive cases:

Case 2.1): $\operatorname{lcm}(p_1, \ldots, p_{i+1}) = p_{i+1}$. By Remark 5.1.2, we can write

$$(C_{p_1}^{n_1} \oplus \ldots \oplus C_{p_{i+1}}^{n_{i+1}})^w = \bigoplus_{\substack{k_1+\ldots+k_{i+1}=w \\ 0\le k_1,\ldots,k_{i+1}\le w}} \binom{w}{k_1, k_2, \ldots, k_{i+1}} C_{\lambda_{i+1}^*}^{\frac{1}{\lambda_{i+1}^*}\prod_{t=1}^{i+1}(p_t n_t)^{k_t}}$$

Among all the addends of the latter sum, only the ones with a multinomial coefficient defined by $k_1, \ldots, k_{i+1}$ such that $\lambda^*_{i+1} = p_{i+1}$ give rise to cycles of length p_{i+1}, where $p_{i+1} = q_\xi$. In particular, it holds that

$$\bigoplus_{\substack{k_1+\ldots+k_{i+1}=w \\ 0\leq k_1,\ldots,k_{i+1}\leq w \\ \lambda^*_{i+1}=p_{i+1}}} \binom{w}{k_1,\ldots,k_{i+1}} C_{\lambda^*_{i+1}}^{\frac{1}{\lambda^*_{i+1}}\prod_{t=1}^{i+1}(p_t n_t)^{k_t}} = C_{q_\xi}^{o_\xi} \ ,$$

and, hence,

$$\sum_{\substack{k_1+\ldots+k_{i+1}=w \\ 0\leq k_1,k_2,\ldots,k_{i+1}\leq w \\ \lambda^*_{i+1}=p_{i+1}}} \binom{w}{k_1,\ldots,k_{i+1}} \cdot \frac{1}{\lambda^*_{i+1}} \cdot \prod_{t=1}^{i+1}(p_t n_t)^{k_t} = o_\xi \ . \tag{5.13}$$

Since $\lambda^*_{i+1} = q_\xi$, when both sides of Equation (5.13) are first multiplied by q_ξ and then summed to the quantity

$$\sum_{\substack{k_1+\ldots+k_{i+1}=w \\ 0\leq k_1,\ldots,k_{i+1}\leq w \\ \lambda^*_{i+1}\neq p_{i+1}\wedge k_{i+1}=0}} \binom{w}{k_1,\ldots,k_{i+1}} \prod_{t=1}^{i+1}(p_t n_t)^{k_t} = \sum_{\substack{k_1+\ldots+k_i=w \\ 0\leq k_1,\ldots,k_i\leq w \\ \lambda^*_i\neq p_{i+1}}} \binom{w}{k_1,\ldots,k_i} \prod_{t=1}^{i}(p_t n_t)^{k_t} = Q^*_i \ ,$$

Equation (5.13) becomes

$$\sum_{\substack{k_1+\ldots+k_{i+1}=w \\ 0\leq k_1,\ldots,k_{i+1}\leq w}} \binom{w}{k_1,\ldots,k_{i+1}} \prod_{t=1}^{i+1} p_t^{k_t} n_t^{k_t} = q_\xi o_\xi + Q^*_i \ . \tag{5.14}$$

Indeed, by the assumption that $\mathrm{lcm}(p_1, \ldots, p_{i+1}) = p_{i+1}$, there can be no tuple $(k_1, \ldots, k_{i+1})$ from the sum of Equation (5.14) such that both the conditions $k_{i+1} \neq 0$ and $\lambda^*_{i+1} \neq p_{i+1}$ hold. Now, Equation (5.14) can be rewritten as

$$(p_1 n_1 + \ldots + p_i n_i + p_{i+1} n_{i+1})^w = q_\xi o_\xi + Q^*_i \ .$$

Since Q^*_i does not depend on n_{i+1}, and, in particular, Q^*_i can be computed on the basis of $n_1, \ldots, n_i$, we get

$$n_{i+1} = \frac{\sqrt[w]{q_\xi o_\xi + Q^*_i} - \sum_{j=1}^{i} p_j n_j}{q_\xi}$$

Case 2.2): $\mathrm{lcm}(p_1, \ldots, p_{i+1}) > p_{i+1}$. When computing the w-th power of $\mathring{X}$ the set $C_{q_\xi}^{o_\xi} = C_{p_{i+1}}^{o_\xi}$ can be formed only by the contribution of those sets $C_{p_{i_1}}^{n_{i_1}}, \ldots, C_{p_{i_j}}^{n_{i_j}}$ (including $C_{p_{i+1}}^{n_{i+1}}$) such that $i_1 < \ldots < i_j$, $i_j = i+1$, and p_{i_e} divides p_{i+1} for each $1 \leq e \leq j$. Since $\mathrm{lcm}(p_{i_1}, \ldots, p_{i_j}) = p_{i+1}$, we can proceeding in the same way as the case *2.1)* but with the indexes $i_1, \ldots, i_j$ instead of $1, \ldots, i+1$, respectively. Therefore, it holds that

$$\sum_{\substack{k_{i_1}+\ldots+k_{i_j}=w \\ 0\leq k_{i_1},\ldots,k_{i_j}\leq w}} \binom{w}{k_{i_1},\ldots,k_{i_j}} \prod_{t=1}^{j}(p_{i_t})^{k_{i_t}}(n_{i_t})^{k_{i_t}} = q_\xi o_\xi + Q^{**}_i \ , \tag{5.15}$$

where

$$Q_i^{**} = \sum_{\substack{k_{i_1}+\ldots+k_{i_{j-1}}=w \\ 0\leq k_{i_1},\ldots,k_{i_{j-1}}\leq w \\ \lambda^{**}_{i_{j-1}}\neq p_{i+1}}} \binom{w}{k_{i_1},\ldots,k_{i_{j-1}}} \prod_{t=1}^{j-1}(p_{i_t}n_{i_t})^{k_{i_t}} \ ,$$

and, hence, Equation (5.15) can be rewritten as

$$(p_{i_1}n_{i_1} + \ldots + p_{i_{j-1}}n_{i_{j-1}} + p_{i_j}n_{i_j})^w = q_\xi o_\xi + Q_i^{**} \ .$$

Since Q_i^{**} does not depend on $n_{i+1} = n_{i_j}$, and, in particular, Q_i^{**} can be computed on the basis of $n_{i_1}, \ldots, n_{i_{j-1}}$, we get

$$n_{i+1} = \frac{\sqrt[w]{q_\xi o_\xi + Q_i^{**}} - \sum_{e=1}^{j-1} p_{i_e}n_{i_e}}{q_\xi}.$$

□

The proposition and the theorem introduced in this section allow to compute the w-root of a sub-dynamical system induced by its periodic points.

At this point it is clear that the w-th root of a DDS is always unique, if it exists (*i.e.*, if all n_i's turn out to be natural numbers).

Example 5.4.1 – Consider the following equation

$$\mathring{X}^2 = C_2^8 \oplus C_3^{48} \oplus C_6^{424} \oplus C_7^{63} \oplus C_{14}^{12} \oplus C_{21}^{24} \oplus C_{42}^{36}.$$

According to Proposition 5.4.1, $p_1 = q_1 = 2$ and $p_2 = q_2 = 3$.

Then, $n_1 = \sqrt[2]{\frac{8}{2}} = 2$ and $n_2 = \sqrt[2]{\frac{48}{3}} = 4$.

At this point $\mathring{X}$ contains $C_2^2 \oplus C_3^4$. Since $(C_2^2 \oplus C_3^4)^2 = C_2^8 \oplus C_3^{48} \oplus C_6^{16}$, $p_{i+1} = 6$. According to Theorem 5.4.2,

$$n_{i+1} = \frac{\sqrt[w]{6 \cdot 424 + (2^2 \cdot 2^2 + 4^2 \cdot 3^2)} - (2 \cdot 2 + 4 \cdot 3)}{6} = 6.$$

Hence, $\mathring{X}$ contains also C_6^6. Given the fact that

$$(C_2^2 \oplus C_3^4 \oplus C_6^6)^2 = C_2^8 \oplus C_3^{48} \oplus C_6^{424},$$

$p_{i+1} = 7$. However, since $\mathrm{lcm}(p_1, \ldots, p_{i+1}) \neq p_{i+1}$,

$$p_{i+1} = \frac{\sqrt[w]{63 \cdot 7}}{7} = 3.$$

Now, $\mathring{X} = C_2^2 \oplus C_3^4 \oplus C_6^6 \oplus C_7^3$ and, since

$$(C_2^2 \oplus C_3^4 \oplus C_6^6 \oplus C_7^3)^2 = C_2^8 \oplus C_3^{48} \oplus C_6^{424} \oplus C_7^{63} \oplus C_{14}^{12} \oplus C_{21}^{24} \oplus C_{42}^{36},$$

we have the value of $\mathring{X}$.

CHAPTER 6

Transient behaviour of DDS

This chapter introduces the last abstraction, the t-abstraction, which aims at enumerating all transient behaviours of the variables of multi-variate polynomial equations with constant right-hand term on $\mathcal{D}$. As for the other abstractions, this is done by introducing a specific notation, studying how the transient parts are involved in sum and product operations, but also by introducing a basic t-abstraction equation. In this chapter, a first upper bound to the number of solutions of a basic equation over t-abstractions is investigated by exploring the connection with the cancellation problem on graphs. Since the goal is the enumeration of the solutions, a first polynomial algorithmic technique for finding all t-abstraction solutions of the basic equations is presented. An exponential approach to enumerate the DDS solutions (up to isomorphism) of any basic equation concludes the chapter.

This chapter presents the t-abstractions of DDS and the two algorithmic approaches proposed in [Doré et al. (2022)].

6.1 The t-abstraction

In this chapter, we introduce the third and last abstraction, the *t-abstraction*. Its goal is to identify the transient behaviour of the variables in an Equation (4.1). As in the previous chapters, we begin by introducing the t-abstraction of a generic DDS S. Recall that, as stated in Chapter 1, we refer to each j-th component of S by its set $\mathcal{X}_j$.

First of all, let us fix an arbitrary indexing function for cycles. Given a cycle $\mathcal{C}_j$ (for $j \in \{1,\dots,L\}$), fix a node $v \in \mathcal{C}_j$ and define the *index function* $g : \{0,\dots,|\mathcal{C}_j|-1\} \longrightarrow \mathcal{C}_j$ such that $g(r) = f^r(v)$. Denote $\mathcal{T}^j_{r,h}$ the set of transient nodes of $\mathcal{X}_j$ for which there exists a path of length h ending in $g(r)$ and made only of transients nodes except for the last one. From now on, we will refer to this set as *the nodes at the layer* (or level) h for the periodic point $g(r)$. Remark that j, r and h respect the specific ranges, namely $j \in \{1,\dots,L\}$, $r \in \{0,\dots,|\mathcal{C}_j|-1\}$, and $h \in \left\{1,\dots,h_j^{max}\right\}$ (where h_j^{max} is the maximum length of a transient in the j-component). These sets can be conveniently represented by a matrix $\mathcal{T}^j$ of sets in which an element at position (r,h) is given by

$$\mathcal{T}^j_{r,h} = \begin{cases} f^{-1}(g(r)) \setminus \mathcal{C}_j & \text{if } h = 1, \\ \bigcup_{v \in \mathcal{T}^j_{r,h-1}} f^{-1}(v) & \text{if } h > 1. \end{cases}$$

Remark that $\mathcal{T}_j$ refers to the set of transient points of a component j while $\mathcal{T}^j$ refers to the same information but in matrix form (*i.e.* with additional information about their height and their first periodic state in the orbit).

Definition 6.1.1 (t-abstraction)**.** The *t-abstraction* of a DDS S, denoted by $\check{S}$, is the multiset $[T^1, T^2, \dots, T^L]$ where T^j (with $j \in \{1,\dots,L\}$) is the matrix, corresponding to the j-th component $\left(\mathcal{X}_j, f|_{\mathcal{X}_j}\right)$, in which the element at the position (r,h) is the multiset

$$T^j_{r,h} = \begin{cases} [|f^{-1}(g(r))|] & \text{if } h = 1, \\ [|f^{-1}(v)| : v \in \mathcal{T}^j_{r,h-1}] & \text{if } h > 1. \end{cases}$$

According to this definition, for each component j, we represent some information about the transient part of the system by a matrix T^j with $|\mathcal{C}_j|$ lines and $h_j^{max}+1$ columns, where each element of is a multiset $T^j_{r,h}$ (with $r \in \{0,\dots,|\mathcal{C}_j|-1\}$ and $h \in \left\{1,\dots,h_j^{max}+1\right\}$) containing the number of predecessors of each node in $\mathcal{T}^j_{r,h-1}$. Remark that multisets, here, are denoted with square brackets, for example, $[2,2,3]$ denotes the multiset containing the symbol 2 with multiplicity 2 and 3 with multiplicity 1. For $h = 1$, the number of predecessors of the periodic node $g(r)$ includes its predecessor in the cycle $\mathcal{C}_j$.

Example 6.1.1 – Consider the connected DDS S in Figure 6.1 with a cycle of length 4 and h_1^{max} equals to 3. Let us consider the index function $g(r) = v_{r+1}$ to introduce the t-abstraction of the DDS.

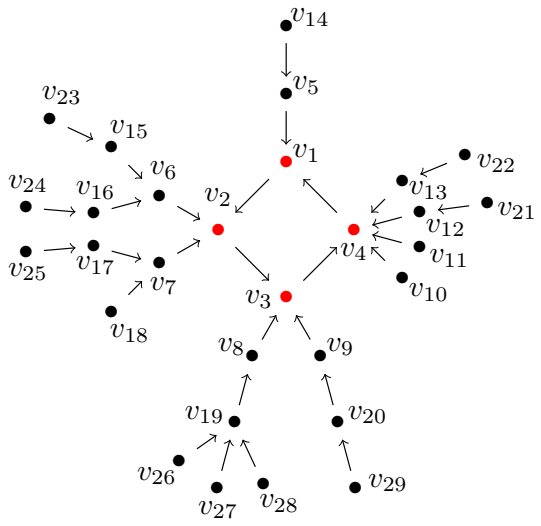


Figure 6.1 – A connected DDS with a cycle of length 4.

The transient points of the system can be represented in the following matrix.

$$\mathcal{T}^1 = \begin{array}{c} \\ 0 \\ 1 \\ 2 \\ 3 \end{array} \begin{array}{c} \begin{array}{ccc} 1 & 2 & 3 \end{array} \\ \begin{pmatrix} \{v_5\} & \{v_{14}\} & \emptyset \\ \{v_6, v_7\} & \{v_{15}, v_{16}, v_{17}, v_{18}\} & \{v_{23}, v_{24}, v_{25}\} \\ \{v_8, v_9\} & \{v_{19}, v_{20}\} & \{v_{26}, v_{27}, v_{28}, v_{29}\} \\ \{v_{10}, v_{11}, v_{12}, v_{13}\} & \{v_{21}, v_{22}\} & \emptyset \end{pmatrix} \end{array}$$

where r is represented on the rows and h on the columns.

For example, the states v_{10}, v_{11}, v_{12}, and v_{13} are the transient predecessors of the node with index 3 (*i.e.* $g(3) = v_4$). Indeed, they are contained in the $\mathcal{T}^1_{3,1}$ set. Considering the upper layers, $\mathcal{T}^1_{3,2}$ contains only v_{21}, v_{22}, while $\mathcal{T}^1_{3,3} = \emptyset$ since v_{21} and v_{22} are both leaves.

The t-abstraction of the system T^1 corresponds to the matrix

$$T^1 = \begin{array}{c} \\ 0 \\ 1 \\ 2 \\ 3 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{pmatrix} [2] & [1] & [0] & \emptyset \\ [3] & [2,2] & [1,1,1,0] & [0,0,0] \\ [3] & [1,1] & [3,1] & [0,0,0,0] \\ [5] & [0,0,1,1] & [0,0] & \emptyset \end{pmatrix} \end{array}$$

where r is represented on the rows and h on the columns.

Remark that the elements in the first column represent the number of incoming edges of the cyclic nodes including the edge coming from the previous periodic point. For example, $T^1_{3,1}$ is equal to 5 since v_4 has 4 incoming arcs from v_{10}, v_{11}, v_{12}, and v_{13}, and one from v_3.

Moreover, $T^1_{3,2}$ is computed considering the incoming edges of the nodes in $\mathcal{T}^1_{3,1}$, and then it is equal to $[0, 0, 1, 1]$. However, let us point out the difference between a 0 in a multiset and an empty one. The first case occurs when leaves are present in the layer, while the second case models an empty layer.

Note that $|T^1_{r,h}|$ is always equal to $|\mathcal{T}^1_{r,h-1}|$ since each element of $T^1_{r,h}$ represents the number of predecessors of a node in $\mathcal{T}^1_{r,h-1}$.

6.1.1 Sum and Product of t-abstractions

As with the other abstractions, let us see how the operations of the semiring $\mathcal{D}$ involve the transient part of the dynamics. In particular, let us show how the sum of t-abstractions is the disjoint union of the multisets of matrices and how the product between t-abstractions can be computed. From now on, let us denote $T^{z,j}$ the t-abstraction matrix of the j-th component of a system z.

Proposition 6.1.1. *Given $S_1 = (\mathcal{X}_1, f_1)$ (resp., $S_2 = (\mathcal{X}_2, f_2)$), let $\check{S}_1 = \sum_{j=1}^{L_1} T^{1,j}$ (resp., $\check{S}_2 = \sum_{i=1}^{L_2} T^{2,i}$). Then,*

$$\check{S}_1 + \check{S}_2 = \sum_{j=1}^{L_1} T^{1,j} + \sum_{i=1}^{L_2} T^{2,i} = T^{1,1} + \ldots + T^{1,L_1} + T^{2,1} + \ldots + T^{2,L_2}.$$

According to this last proposition, the result of a sum operation is a multiset containing the t-abstractions of the components (of both systems) with multiplicities given by the sum of the multiplicities in the two original t-abstractions. Hence, the sum of the t-abstractions is by definition the t-abstraction of the sum.

Let us now consider the product operation, for example $S_3 = S_1 \cdot S_2$. Recall that (v_1, v_2) is a periodic point of S_3 if and only if v_1 and v_2 are periodic points of S_1 and S_2, respectively. Then, we can also conclude that the transient nodes of S_3 involve the Cartesian states (v_1, v_2) where both v_1, v_2 are transient nodes of S_1 and S_2, respectively, or just one of them is a periodic point (see Example 6.1.2).

Given the fact that the product operation between DDS is distributive over the sum, let us introduce how the product of the t-abstractions of two connected DDS can be computed (see Figure 6.1.3). Note that, in a connected DDS, there is just one cycle $\mathcal{C}$ and $\mathcal{P} = \mathcal{C}$. From now on, we will then denote $\mathcal{C}_z$ (or $\mathcal{P}_z$) the cyclic states of a connected DDS S_z, $\mathcal{T}_z$ the sets of its transients nodes, and T^z, $\mathcal{T}^z$ the matrices of the same system.

For two multiset of natural numbers $\mathcal{M} = [d_1, \ldots, d_n]$ and $\mathcal{M}' = [d'_1, \ldots, d'_m]$, we denote their disjoint union as $\mathcal{M} + \mathcal{M}'$ and we define the product $\mathcal{M} \otimes \mathcal{M}' = [d \cdot d' \mid d \in \mathcal{M}, d' \in \mathcal{M}']$.

Proposition 6.1.2. *Assume $\check{S}_1 = [T^1]$ (with $|\mathcal{C}_1|$ rows and h_1^{max} columns) and $\check{S}_2 = [T^2]$ (with $|\mathcal{C}_2|$ rows and h_2^{max} columns). Then, the t-abstraction $\check{S}_3$ of the product $S_3 = S_1 \cdot S_2$ is a multiset of matrices $[T^{3,1}, \ldots, T^{3,\gcd(|\mathcal{C}_1|,|\mathcal{C}_2|)}]$. Each matrix $T^{3,i}$ has $\operatorname{lcm}(|\mathcal{C}_1|, |\mathcal{C}_2|)$ rows and $\max\{h_1^{max}, h_2^{max}\}$ columns, and each element is computed from $\check{S}_1$ and $\check{S}_2$ as*

$$T^{3,i}_{r,h} = T^1_{r,h} \otimes \left(\sum_{j=0}^{h-1} T^2_{r-j+(i-1),h-j} \right) + T^2_{r+(i-1),h} \otimes \left(\sum_{j=1}^{h-1} T^1_{r-j,h-j} \right) \tag{6.1}$$

where the lines indices r and $r-j$ (respectively $r-j+(i-1)$ and $r+(i-1)$) are interpreted modulo $|\mathcal{C}_1|$ (respectively $|\mathcal{C}_2|$).

Proof. It has been proved that the product operation between two connected components gives $\gcd(|\mathcal{C}_1|, |\mathcal{C}_2|)$ components with cycle length $\mathrm{lcm}(|\mathcal{C}_1|, |\mathcal{C}_2|)$. At this point, we need to prove that the maximum transient length of the resulting components is $\max\{h_1^{max}, h_2^{max}\}$.

Let us consider two connected DDS $S_1 = (\mathcal{X}_1, f_1)$ and $S_2 = (\mathcal{X}_2, f_2)$. According to definitions over DDS, we know that $f_1^{h_1^{max}}(v_1) \in \mathcal{P}_1$ for all $v_1 \in \mathcal{X}_1$, otherwise, with a smaller number of application of f_1, the resulting state can be a transient state. If we suppose $h_1^{max} \geq h_2^{max}$, for all $h \geq h_2^{max}$, $f_2^h(v_2) \in \mathcal{P}_2$ for all $v_2 \in \mathcal{X}_2$. Then, a state (v_1, v_2) of the resulting system (with $v_1 \in \mathcal{T}^1_{r_1,h_1^{max}}$ for some $r_1 \in \{0, \ldots, |\mathcal{C}_1| - 1\}$) is a transient node that requires h_1^{max} applications of $f_3 = f_1 \times f_2$ to obtain a periodic node. Then, for all $v_2 \in \mathcal{X}_2$, $(f_1 \times f_2)^{h_1^{max}}(v_1, v_2) \in \mathcal{P}_3$ and $(f_1 \times f_2)^h(v_1, v_2) \notin \mathcal{P}_3$ with $h < h_1^{max}$. For this reason, $h_3^{max} = h_1^{max}$. Remark that, since the statement is true for all v_2, all the components $\mathcal{T}^{3,i}$ will have maximal transient length h_1^{max}. Symmetrically, if we consider $h_1^{max} \leq h_2^{max}$, one obtains all $\mathcal{T}^{3,i}$ having h_2^{max}, for all $1 \leq i \leq \gcd(|\mathcal{C}_1|, |\mathcal{C}_2|)$.

Let g_1, g_2, and g_3 be the index functions of the three systems. By considering a state $(v_1, v_2) \in \mathcal{X}_1 \times \mathcal{X}_2$, if $(v_1, v_2) \in \mathcal{T}^{3,i}_{r,h}$ and $(f_1 \times f_2)^h(v_1, v_2) = (p_1, p_2)$, we know that $g_3(r) = (p_1, p_2)$ and at least one of the following conditions holds:

- $v_1 \in \mathcal{T}^1_{r_1,h}$ with $g_1(r_1) = p_1$;
- $v_2 \in \mathcal{T}^2_{r_2,h}$ with $g_2(r_2) = p_2$.

Suppose $v_1 \in \mathcal{T}^1_{r_1,h}$ with $g_1(r_1) = p_1$. Then, if we are interested in the elements of $T^{3,i}_{r,h}$ including v_1, we need to consider all the feasible values of v_2. In order for (v_1, v_2) to belong to $\mathcal{T}^{3,i}_{r,h}$, the state v_2 must satisfy $f_2^h(v_2) = p_2$. Then v_2 must necessarily belong to one of $\mathcal{T}^2_{r_2,h}$, $\mathcal{T}^2_{r_2-1,h-1}$, $\mathcal{T}^2_{r_2-2,h-2}$, …, $\mathcal{T}^2_{r_2-(h-1),1}$ or be the periodic point $g_2(r_2 - h)$.

The reasoning is similar for the case $v_2 \in \mathcal{T}^2_{r_2,h}$. Since either $v_1 \in \mathcal{T}^1_{r_1,h}$, or $v_2 \in \mathcal{T}^2_{r_2,h}$, or both, the predecessors (in terms of multiplicity) can be computed, for the component i containing (p_1, p_2), by

$$T^{3,i}_{r,h} = T^1_{r,h} \otimes \left(\sum_{j=1}^{h-1} T^2_{r-j,h-j} \right) + T^2_{r,h} \otimes \left(\sum_{j=1}^{h-1} T^1_{r-j,h-j} \right) + T^1_{r,h} \otimes T^2_{r,h}.$$

Suppose, once again, that $v_1 \in \mathcal{T}^1_{r_1,h}$. In order to generate all components, we must consider the fact that there might exist a state v_2' such that $(f_1 \times f_2)^h(v_1, v_2') = (p_1, f_2(p_2))$ with $(f_1 \times f_2)^{h'}(p_1, p_2) \neq (p_1, f_2(p_2))$ for all $h' \in \mathbb{N}$; this means that (v_1, v_2) and (v_1, v_2') belong to different components of S_3. In order for (v_1, v_2') to belong to $\mathcal{T}^{3,i}_{r,h}$, state v_2' must satisfy $f_2^h(v_2') = f_2(p_2)$ with $g_2(r_2) = p_2$. Then v_2' must necessarily belong to one of $\mathcal{T}^2_{(r_2+1),h}$, $\mathcal{T}^2_{(r_2+1)-1,h-1}$, $\mathcal{T}^2_{(r_2+1)-2,h-2}$, …, $\mathcal{T}^2_{(r_2+1)-(h-1),1}$ or be the periodic point $g_2((r_2 + 1) - h)$.

In general, we can consider a state u such that $(f_1 \times f_2)^h(v_1, u) = (p_1, f_2^i(p_2))$ with $1 \leq i \leq \gcd(|\mathcal{C}_1|, |\mathcal{C}_2|)$ and $(f_1 \times f_2)^{h'}(p_1, p_2) \neq (p_1, f_2^i(p_2))$ for all $h' \in \mathbb{N}$. If $(v_1, u) \in T^{3,i}_{r,h}$, u must be a state such that $f_2^h(u) = f_2^i(p_2)$ with $g_2(r_2) = p_2$. Then u must necessarily belong to one of $\mathcal{T}^2_{r_2+(i-1),h}$, $\mathcal{T}^2_{r_2+(i-1)-1,h-1}$, $\mathcal{T}^2_{r_2+(i-1)-2,h-2}$, …, $\mathcal{T}^2_{r_2+(i-1)-(h-1),1}$ or be the periodic point $g_2(r_2 + (i-1) - h)$. As a consequence, we can compute each element of the t-abstraction of an arbitrary component i, with $1 \leq i \leq \gcd(|\mathcal{C}_1|, |\mathcal{C}_2|)$, as

$$T^{3,i}_{r,h} = T^1_{r,h} \otimes \left(\sum_{j=1}^{h-1} T^2_{r-j+(i-1),h-j} \right) + T^2_{r+(i-1),h} \otimes \left(\sum_{j=1}^{h-1} T^1_{r-j,h-j} \right) + T^1_{r,h} \otimes T^2_{r+(i-1),h}.$$

Since $g_2(r-j+(i-1))$ and $g_2(r+(i-1))$ represent periodic points of S_2, and thus exhibit a cyclic behaviour, they must be interpreted modulo $|\mathcal{C}_2|$, and similarly $g_1(r)$ and $g_1(r-j)$ modulo $|\mathcal{C}_1|$. □

Example 6.1.2 – An example of product operation between connected DDS S_1 and S_2. For the sake of visual clarity, only the names of some nodes are shown.

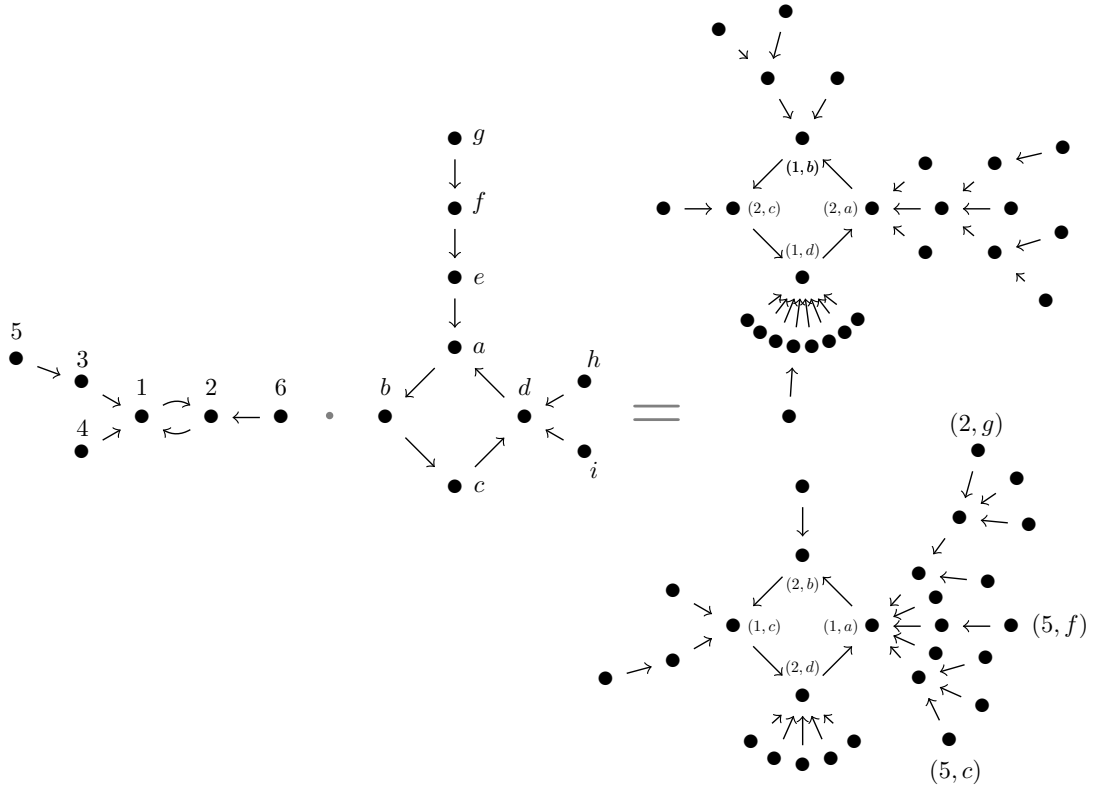


Figure 6.2 – Product operation between connected DDS

According to the cycles lengths of the two systems, the result of the product operation $S_1 \cdot S_2$ consists of $\gcd(2,4)$ components with cycles of length $\mathrm{lcm}(2,4)$. Intuitively, we can see the two resulting cycles as the two different "parallel" executions of the cyclic dynamics ($\{(1,a),(2,b),(1,c),(2,d)\}$ or $\{(2,a),(1,b),(2,c),(1,d)\}$). The set of vertices of the resulting graph is the Cartesian product $\mathcal{X}_1 \times \mathcal{X}_2$. A generic node (v_1, v_2) is cyclic in the result iff v_1 is a cyclic node of S_1 and v_2 is a cyclic node of S_2, otherwise (v_1, v_2) is transient. One way to understand the transients connected to a cyclic node (v_1, v_2) is to retrace the edges of the original graphs backwards (i.e. combinatorially consider all the predecessors). For example, the set of predecessors of the node $(1, a)$ is the Cartesian product of predecessors 1 and of a ($\{2, 3, 4\}$ and $\{d, e\}$). In fact, we see that $(1, a)$ has 6 incoming edges. Note that backtracking

the edges in the transient nodes only will produce a transient with a depth equal to the minimum of the original transients depth ($\{(5, f), (3, e)\}$ is an example). However, by backtracking only in cyclic nodes of either one graph and only in transient nodes of the other graph, we obtain a copy of the transient ($\{(2, g), (1, f), (2, e)\}$ and $\{(5, c), (3, d)\}$ are examples).

Example 6.1.3 – An example of product operation between t-abstractions T^1 and T^2. Above in Figure 6.3 there are the two t-abstractions and the corresponding connected DDS. Below there is the resulting t-abstraction T^3 computed according to Proposition 6.1.2.

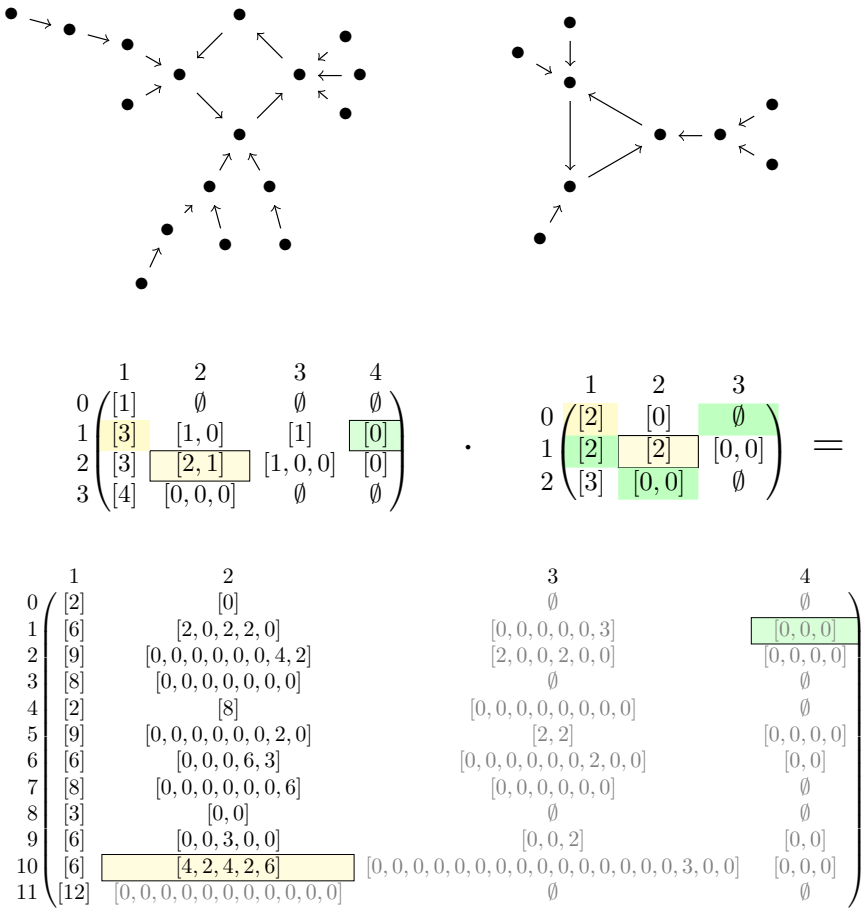


Figure 6.3 – Product of t-abstractions T^1 and T^2.

Let us suppose that we have already computed all $T^3_{r,h}$ with $r \in \{0, \ldots, 11\}$ and $h = 1$, and also all $T^3_{r,h}$ with $r \in \{0, \ldots, 9\}$ and $h = 2$. Then, we need to compute $T^3_{10,2}$ (in yellow). According to the proposition, $T^3_{10,2} = T^1_{2,2} \otimes (T^2_{1,2} + T^2_{0,1}) + T^2_{1,2} \otimes T^1_{1,1} = [2, 1] \otimes ([2, 2]) + [2] \otimes [3] = [4, 2, 4, 2, 6]$. Intuitively, we should follow the diagonal of elements in the two original matrices until we reach the first column (as shown in yellow). Note that if we need to calculate an element $T^3_{r,h}$ such that one of the two matrices does not have a column h, we still consider the elements on the diagonal (with $h' < h$). An example is $T^3_{2,4} = T^1_{1,4} \otimes (T^2_{0,3} + T^2_{2,2} + T^2_{1,1})$ (in green in the matrices).

Considering now general DDS, the t-abstraction of a product operation can be computed according to the distributivity of the operation and the previous proposition.

Proposition 6.1.3. *Given* $S_1 = (\mathcal{X}_1, f_1)$ *(resp.,* $S_2 = (\mathcal{X}_2, f_2)$*), let* $\check{S}_1 = \sum_{j=1}^{L_1} T^{1,j}$ *(resp.,* $\check{S}_2 = \sum_{i=1}^{L_2} T^{2,i}$*). Then,* $\check{S}_1 \cdot \check{S}_2$ *is*

$$\sum_{j=1}^{L_1} T^{1,j} \cdot \sum_{i=1}^{L_2} T^{2,i} = \sum_{j=1}^{L_1} \sum_{i=1}^{L_2} T^{1,j} \cdot T^{2,i}.$$

Now that we know how the operations of sum and product act on the transient parts of systems, let us go back to Equation (4.1) which is the problem that we want to solve. According to the t-abstraction definition, it can be rewritten as follows to obtain the *t-abstraction equation*

$$\sum_{i=1}^{L_1} T^{1,i} \cdot \check{X}_1^{w_1} + \sum_{i=1}^{L_2} T^{2,i} \cdot \check{X}_2^{w_2} + \ldots + \sum_{i=1}^{L_m} T^{m,i} \cdot \check{X}_m^{w_m} = \sum_{j=1}^{L_B} T^{B,j} \tag{6.2}$$

Chapter 5 shows that to solve a generic equation over the asymptotic behaviour of DDS one needs to solve a finite number of basic equations. Formally, this is possible thanks to algebraic transformations, called contractions steps. Let us show how a similar idea can also be applied to solve equations over the t-abstractions.

6.1.2 Contraction Steps over t-abstractions

Since each component of the right-hand side is the result of a product operation inside a monomial, then solving an Equation (6.2) is equivalent to solving a finite number of systems of the following form, one for each possible partition $\check{B}^{1,1}, \check{B}^{1,2}, \ldots, \check{B}^{m,L_m}$ of the multiset $\check{B} = [T^{B,1}, \ldots, T^{B,L_B}]$. This transformation from an Equation (6.2) to a finite number of systems of the form (6.3) correspond to the *contraction steps*.

In a system, a multiset $\check{B}^{z,i}$ represent the subset of components of the right term generated by the z, i monomial (with $z \in \{1, \ldots, m\}$ and $i \in \{1, \ldots, L_z\}$).

$$\begin{cases} T^{1,1} \cdot \check{X}_1^{w_1} & = \check{B}^{1,1} \\ T^{1,2} \cdot \check{X}_1^{w_1} & = \check{B}^{1,2} \\ & \vdots \\ T^{1,L_1} \cdot \check{X}_1^{w_1} & = \check{B}^{1,L_1} \\ T^{2,1} \cdot \check{X}_2^{w_2} & = \check{B}^{2,1} \\ & \vdots \\ T^{m,L_m} \cdot \check{X}_m^{w_m} & = \check{B}^{m,L_m} \end{cases} \tag{6.3}$$

Indeed, the problem boils down to be able to solve equations of the form

$$T^{z,i} \cdot \check{X}_z^{w_z} = \check{B}^{z,i}$$

(with $z \in \{1, \ldots, m\}$ and $i \in \{1, \ldots, L_z\}$). This last equation is based on a coefficient that is a connected dynamics graph and a right-hand side with multiple components. In order to solve it, we can start from the fact that when two components are multiplied they generate one or more

components but all with the same period. Then, we can further simplify the equation as for the cyclic case. Let $p_1, p_2, \ldots, p_l$ be the lengths of periods involved in a $\check{B}^{z,i}$. At this point, finding $\check{X}_z^{w_z}$ corresponds to solving l simpler equations.

$$
\begin{aligned}
T^{z,i} \cdot \check{X}_{p_1} &= \sum_{j=1,\ |\mathcal{C}_{(z,i),j}|=p_1}^{|\check{B}^{z,i}|} T^{(z,i),j} \\
T^{z,i} \cdot \check{X}_{p_2} &= \sum_{j=1,\ |\mathcal{C}_{(z,i),j}|=p_2}^{|\check{B}^{z,i}|} T^{(z,i),j} \\
&\vdots \\
T^{z,i} \cdot \check{X}_{p_l} &= \sum_{j=1,\ |\mathcal{C}_{(z,i),j}|=p_l}^{|\check{B}^{z,i}|} T^{(z,i),j}
\end{aligned}
$$

Then, $\check{X}_z^{w_z}$ will be the Cartesian product of the $\check{X}_p$ computed, with $p \in \{p_1, p_2, \ldots, p_l\}$. Up to this point, the reasoning is similar to that for a-abstractions. However, the last steps change.

Each one of these equations

$$
T^{z,i} \cdot \check{X}_p = \sum_{j=1,\ |\mathcal{C}_{(z,i),j}|=p}^{|\check{B}^{z,i}|} T^{(z,i),j} \tag{6.4}
$$

is characterised by components with the same cycle length p in the right-hand side. To solve them, we return again to the fact that the product distributes over the sum. Let us therefore consider the variable to be a connected DDS X' too. By choosing a generic component of the right-hand side $T^{B'}$, we obtain an inequality of the following form

$$
T^{z,i} \cdot T^{X'} \supseteq T^{B'} \tag{6.5}
$$

In fact, according to Proposition 6.1.2, we know that $T^{z,i} \cdot T^{X'}$ can lead to one or more components. For this reason, we denote $T^{z,i} \cdot T^{X'} \supseteq T^{B'}$, since our goal is then to find $T^{X'}$ that, once multiplied with $T^{z,i}$, generates a DDS having at least a component isomorphic to $T^{B'}$.

If $T^{z,i} \cdot T^{X'}$ generates all the components of period p of the Equation (6.4), the simplification process stops since we found the solution. If some components are not involved in the result of the product operation, the process can be iterated on the remaining components solving

$$
T^{z,i} \cdot T^{X''} \supseteq T^{B''}, \tag{6.6}
$$

with $T^{B''} \in \sum_{j=1,\ |\mathcal{C}_{(z,i),j}|=p}^{|\check{B}^{z,i}|} T^{(z,i),j} \setminus (T^{z,i} \cdot T^{X'})$, to find the missing components of $\check{X}_p$. In fact, this means that X_p contains more than one component, in particular $\check{X}_p$ contains $T^{X'}$ and $T^{X''}$ at least. This process is iterated until all the components in the right-hand side of an Equation (6.4) are generated.

In the end, solving a system of the form (6.3) requires to be able to solve a finite number of inequalities of the from (6.5). Remark that, if the goal is the enumeration of the solution set of an

Equation (6.2) (or of an Equation (4.1)), it is necessary to enumerate the solutions of Equations (6.5) too. For the t-abstraction, we will then consider the following basic equation

$$T^A \cdot T^X \supseteq T^B \tag{6.7}$$

where T^A, T^X, and T^B are the t-abstractions of three connected DDS A, X and B such that $A \cdot X \supseteq B$.

Here, as in the case of equations over a-abstractions, it would be better to consider a simplified equation to limit the computations that have to be done. However, remember that it could be even better to apply the t-abstraction only after studying the solutions of the corresponding a-abstraction equation since it limits the solution space giving the feasible cycle length of the solutions (as explained in Chapter 4). For this reason, the number of cyclic nodes in X of a basic t-abstraction equation will be a given parameter.

6.2 Basic t-abstraction equations

This section aims at studying the problem of solving an Equation (6.7). In the beginning, the goal is to identify the number of solutions of a basic t-abstraction equation and then introduce different algorithmic approaches. First, a polynomial method to find all feasible t-abstractions of the solutions and another exponential to identify the real solutions up to isomorphism.

Before going into details, let us recall that, as anticipated, in this thesis work, DDS are always considered up to isomorphism. However, in the following section, we will follow the ideas of graph theory presented in Chapter 2 and hence we will express the isomorphism by means of the symbol $\cong$.

6.2.1 The Cancellation Problem over Transients

In this section, we aim to introduce a first upper bound of the number of solutions of a basic equation $A \cdot X \supseteq B$ over connected DDS. We will represent such a dynamics with an infinite anti-arborescence (or in-tree). An in-tree is a directed rooted tree where the edges are directed towards the root. We call this in-tree the *unroll* of a DDS (see Figure 6.4 for an example).

Definition 6.2.1 (Unroll of a DDS)**.** Given a connected DDS $S = (\mathcal{X}, f)$ and a periodic node v of S, the *unroll* of S from v is an infinite tree $U_v(S) = (\mathcal{V}, \mathcal{E})$ with vertices $\mathcal{V} = \bigcup_{i\in\mathbb{N}} \mathcal{V}_i$, where $\mathcal{V}_0 = \{(v, 0)\}$ and $\mathcal{V}_{i+1} = \{(u, i + 1) \mid u \in f^{-1}(v) \text{ for some } (u, i) \in \mathcal{V}_i\}$, and edges between vertices $(u, i + 1)$ and $(f(u), i)$ on two consecutive levels $i + 1$ and i (with $i \in \mathbb{N}$).

Remark that, starting from a generic connected DDS (with just one cycle of length p), we can introduce p different unrolls, one for each node $v \in \mathcal{P}$. Let us introduce a product of (finite or infinite) in-trees to be applied over unrolls to obtain the (finite or infinite) in-tree modelling the dynamics of the result of a direct product over DDS. Intuitively, this product is the direct product applied layer by layer (see Figure 6.5).

Definition 6.2.2 (Product of in-trees)**.** Consider two infinite or finite in-trees $I_1 = (\mathcal{V}_1, \mathcal{E}_1)$ and $I_2 = (\mathcal{V}_2, \mathcal{E}_2)$ with roots r_1 and r_2, respectively. The *product of in-trees* $I_1 \star I_2$ is the in-tree $(\mathcal{V}, \mathcal{E})$ such that $(r_1, r_2) \in \mathcal{V}$ and, for all $(v, u) \in \mathcal{V}$, if there exist $v' \in \mathcal{V}_1$ and $u' \in \mathcal{V}_2$ such that $(v', v) \in \mathcal{E}_1$ and $(u', u) \in \mathcal{E}_2$, then $(v', u') \in \mathcal{V}$. The set of edges is defined as $\mathcal{E} = \{((v', u'), (v, u)) \mid (v', v) \in \mathcal{E}_1 \text{ and } (u', u) \in \mathcal{E}_2\}$. Notice that $I_1 \star I_2$ is an infinite in-tree iff I_1 and I_2 are infinite in-trees.

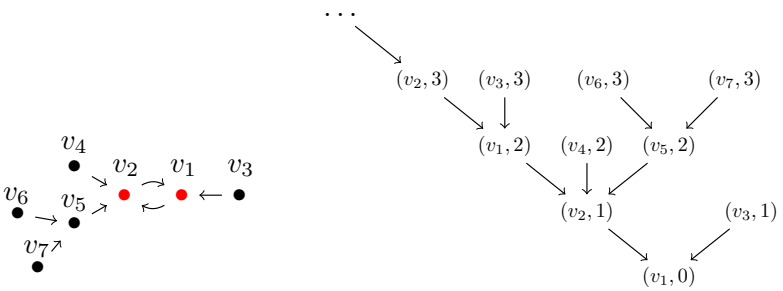


Figure 6.4 – The unroll $U_{v_1}(S)$ (right) of the connected DDS S (left). The root $(v_1, 0)$ of the infinite in-tree is in the bottom layer of the structure. Intuitively, the unroll shows all the trajectories which reach at some point the root state.

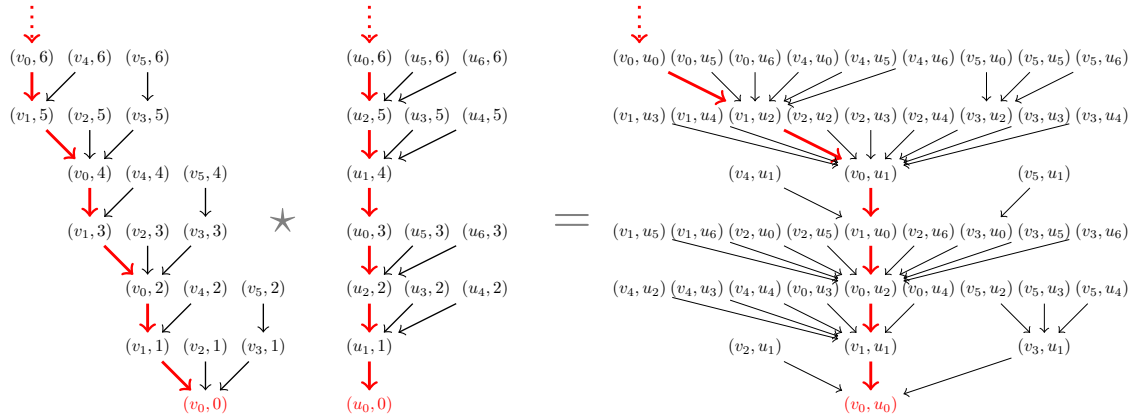


Figure 6.5 – Example of product $\star$ (Definition 6.2.2) over infinite in-trees (in this case two unrolls $U_{v_0}(G_1)$ and $U_{u_0}(G_2)$). The product is intuitively equivalent to the direct product applied level by level. The roots of the unrolls and their infinite paths are highlighted in red.

Considering an instance of Equation (6.7), we can now introduce a corresponding equation over unrolls, and study whether there can be more than one solution. For the moment, let us consider a generic cyclic node of A, X and B (let a, x and b be respectively these three states). Suppose that there exists X such that $U_a(A) \star U_x(X) \cong U_b(B)$, and Y (not isomorphic to X) such that $U_a(A) \star U_y(Y) \cong U_b(B)$ (with y also a generic cyclic node of Y). If $U_x(X) \ncong U_y(Y)$, it means that there is a difference at some minimal level between the two infinite in-trees. Given a generic infinite in-tree I (such as an unroll), we define the function *cut* which gives the finite subtree up to the layer t. Then, the result of $\mathrm{cut}(I, t)$ (with $t \in \mathbb{N}$) is a finite sub-in-tree induced by the set of vertices having at most distance t from the root. We will denote the cut of an unroll $U_a(A)$, at a level t, by $C_{a,t}(A)$. Remark that the cut distributes over the $\star$ product. In our problem, if $U_x(X) \ncong U_y(Y)$, then there exists a minimum $t \in \mathbb{N}\backslash\{0\}$ such that $C_{x,t}(X) \ncong C_{y,t}(Y)$ and $C_{x,t-1}(X) \cong C_{y,t-1}(Y)$.

As presented in Chapter 2, an important result of graph theory is that two graphs G_1 and G_2 are isomorphic iff, for all graph G, $\hom(G, G_1) = \hom(G, G_2)$, where $\hom(G, G_1)$ is the number of homomorphisms from G to G_1 [Lovász (1971)]. Then, in our case, if $C_{a,t}(A) \star C_{x,t}(X) \cong C_{b,t}(B)$ and $C_{a,t}(A) \star C_{y,t}(Y) \cong C_{b,t}(B)$, we must have $\hom(G, C_{(a,x),t}(A \star X)) = \hom(G, C_{(a,y),t}(A \star$

$Y)) = \hom(G, C_{b,t}(B))$ for all G. Then, it is necessary to understand how $\hom(G, C_{a,t}(A) \star C_{x,t}(X))$ and $\hom(G, C_{a,t}(A) \star C_{y,t}(Y))$ can be computed. As a convention, we will use F to denote finite in-trees (such as infinite in-trees after a cut operation).

Theorem 6.2.1. *For any graph G,* $\hom(G, F_1 \star F_2) = \hom(G, F_1) \cdot \hom(G, F_2)$.

To prove this theorem, we need the following lemma. In the following, $\mathrm{Hom}(G_1, G_2)$ will denote the set of homomorphisms from G_1 to G_2.

Lemma 6.2.2. *If $\tau \in \mathrm{Hom}(G, F_1 \star F_2)$ with $\tau(u) = (u_1, u_2)$, then $\pi_1 \circ \tau \in \mathrm{Hom}(G, F_1)$ and $\pi_2 \circ \tau \in \mathrm{Hom}(G, F_2)$, where $\pi_1(u_1, u_2) = u_1$ and $\pi_2(u_1, u_2) = u_2$.*

Proof. If (u, v) is an arc of G, $(\tau(u), \tau(v))$ must be an arc of $F_1 \star F_2$. Let us suppose that $\tau(u) = (u_1, u_2)$ and $\tau(v) = (v_1, v_2)$. Then, from the product definition, the fact that $((u_1, u_2), (v_1, v_2))$ is an arc of $F_1 \star F_2$ implies that (u_1, v_1) is an arc of F_1 and (u_2, v_2) is an arc of F_2. Then, by applying the projection π_1, we obtain $(\pi_1 \circ \tau)(u) = u_1$ and $(\pi_1 \circ \tau)(v) = v_1$. Then, $\pi_1 \circ \tau \in \mathrm{Hom}(G, F_1)$. By same reasoning we obtain $\pi_2 \circ \tau \in \mathrm{Hom}(G, F_2)$. □

Proof of Theorem 6.2.1. Let $\varphi : \mathrm{Hom}(G, F_1 \star F_2) \to \mathrm{Hom}(G, F_1) \times \mathrm{Hom}(G, F_2)$ be the function defined by $\varphi(\tau) = (\pi_1 \circ \tau, \pi_2 \circ \tau)$; by Lemma 6.2.2 this is indeed a well-defined function. We prove that φ is a bijection. We first show that φ is surjective, let $(\tau_1, \tau_2) \in \mathrm{Hom}(G, F_1) \times \mathrm{Hom}(G, F_2)$; we need to find $\tau \in \mathrm{Hom}(G, F_1 \star F_2)$ such that $\varphi(\tau) = (\tau_1, \tau_2)$. Let $\tau(v) = (\tau_1(v), \tau_2(v))$. Then, $\varphi(\tau)(v) = (\pi_1(\tau(v)), \pi_2(\tau(v))) = (\pi_1(\tau_1(v), \tau_2(v)), \pi_2(\tau_1(v), \tau_2(v)))$ $= (\tau_1(v), \tau_2(v))$ implies $\varphi(\tau) = (\tau_1, \tau_2)$. Let us now prove that φ is also injective, *i.e.* $\varphi(\tau) = \varphi(\tau')$ implies $\tau = \tau'$. We have $\varphi(\tau)(v) = (\pi_1(\tau(v)), \pi_2(\tau(v)))$ and $\varphi(\tau')(v) = (\pi_1(\tau'(v)), \pi_2(\tau'(v)))$. If $\varphi(\tau) = \varphi(\tau')$ then $\pi_1(\tau(v)) = \pi_1(\tau'(v))$ and $\pi_2(\tau(v)) = \pi_2(\tau'(v))$, which implies $\tau = \tau'$. □

This allows us to write $\hom(G, C_{(a,x),t}(A \star X)) = \hom(G, C_{(a,y),t}(A \star Y)) = \hom(G, C_{b,t}(B))$ as $\hom(G, C_{a,t}(A)) \cdot \hom(G, C_{x,t}(X)) = \hom(G, C_{a,t}(A)) \cdot \hom(G, C_{y,t}(Y)) = \hom(G, C_{b,t}(B))$. Now, we need an additional lemma to continue our reasoning.

Lemma 6.2.3. *Consider any graph G and any finite in-tree F, then* $\hom(G, F) \neq 0$ *iff* $\hom(G, P_s) \neq 0$ *where P_s is just a directed path with s edges and s is the height of F.*

Proof. An homomorphism from an in-tree F to P_s always exists with s being height of F: one can map all the nodes of the i-th level of the tree to the i-th node of the path. Conversely, a homomorphism from P_s to F exists by choosing any path of length s in F. Therefore, by composition of homomorphisms, $\hom(G, F) \neq 0$ iff $\hom(G, P_s) \neq 0$. □

We are studying when $\hom(G, C_{a,t}(A)) \cdot \hom(G, C_{x,t}(X)) = \hom(G, C_{a,t}(A)) \cdot \hom(G, C_{y,t}(Y)) = \hom(G, C_{b,t}(B))$ implies $C_{x,t}(X) \cong C_{y,t}(Y)$. This is a special case of the *cancellation problem* presented in Chapter 2. Here, we aim at studying the cancellation over DDS through the $\star$ product over unrolls. At this point of the reasoning, we can reduce our problem to a corresponding one over finite trees.

Theorem 6.2.4. *For any finite in-trees F, X, and Y of the same height, $F \star X \cong F \star Y$ implies $X \cong Y$.*

Proof. According to Theorem 6.2.1, since $F \star X \cong F \star Y$, we have $\hom(G,F) \cdot \hom(G,X) = \hom(G,F) \cdot \hom(G,Y)$ for all graph G. If $\hom(G,F) \neq 0$, we can divide by it to obtain $\hom(G,X) = \hom(G,Y)$. If $\hom(G,F) = 0$, according to Lemma 6.2.3, we have $\hom(G,P_s) = 0$ where s is the height of F. Since F, X, and Y have the same height, also $\hom(G,X) = 0$ and $\hom(G,Y) = 0$. □

Theorem 6.2.4 over finite trees can then be generalised to infinite trees (including unrolls).

Theorem 6.2.5. *$I \star X \cong I \star Y$ implies $X \cong Y$, for all infinite in-trees X, Y, and I.*

Proof. If $X \ncong Y$, there exists a minimum $t \in \mathbb{N} \setminus \{0\}$ such that $C_t(X) \ncong C_t(Y)$ and $C_{t-1}(X) \cong C_{t-1}(Y)$. Then, since $I \star X \cong I \star Y$, we have $C_t(I \star X) \cong C_t(I \star Y)$. Since the cut operation is distributive over product, we have $C_t(I) \star C_t(X) \cong C_t(I) \star C_t(Y)$. According to the previous theorem, this means that $C_t(X) \cong C_t(Y)$ which is a contradiction. □

Up to now, we have proved that considering two equations over unrolls $U_a(A) \star U_x(X) \cong U_a(A) \star U_y(Y) \cong U_b(B)$ implies $U_x(X) \cong U_y(Y)$. This means that given a basic equation of unrolls and given a and b, the solution, if any, is unique. Now, we want to show how many equations over unrolls we need to study to enumerate the solutions of $A \cdot X \supseteq B$. Note that, in the case of $A \cdot X$ with $\gcd(|\mathcal{P}_A|, |\mathcal{P}_X|) \neq 1$, the result of the product of the unrolls of A and X will be the unroll of only one component of $A \cdot X$, which is in accordance with Equation (6.7).

Let us fix b and study, for every $a \in \mathcal{P}_A$, the corresponding equation over unrolls. This gives us $|\mathcal{P}_A|$ equations to study. If we fix a, we have $|\mathcal{P}_B|$ equations to consider. Hence, fixing b is more efficient, since $|\mathcal{P}_B| \geq |\mathcal{P}_A|$. The question is now to determine if it is necessary to try another b'. To answer this, we will introduce the notion of roll, which is intuitively the opposite of an unroll (see Figure 6.6 for an example).

Definition 6.2.3 (Roll of an infinite in-tree)**.** Let $I = (\mathcal{V}, \mathcal{E})$ be an infinite in-tree with root r and only one infinite path $(\ldots, v_n, v_{n-1}, \ldots, v_2, v_1 = r)$, and let $\ell \geq 1$ be an integer. Let $J = (\mathcal{V}, \mathcal{E}')$ with $\mathcal{E}' = (\mathcal{E} \setminus \{(v_{\ell+1}, v_\ell)\}) \cup \{(r, v_\ell)\}$. Then $J = I' + J'$ where I' is an infinite in-tree and J' is a finite, connected dynamics graph with a cycle of length ℓ. We call J' the *roll* (of length ℓ) of I, in symbols $R_\ell(I)$.

Notice that, for all $a \in \mathcal{P}(A)$, we have $R_{|\mathcal{P}_A|}(U_a(A)) \cong A$, that is, by rolling up at length $|\mathcal{P}_A|$ the unroll of A in any of its periodic points we obtain the initial dynamics A.

Lemma 6.2.6. *Let $A \cdot X \supseteq B$ with $\operatorname{lcm}(|\mathcal{P}_A|, |\mathcal{P}_X|) = |\mathcal{P}_B|$. Suppose that $U_a(A) \star U_x(X) \cong U_b(B)$ for some $a \in \mathcal{P}_A$, $x \in \mathcal{P}_X$, and $b \in \mathcal{P}_B$. Also suppose that $U_{a'}(A) \star U_y(Y) \cong U_{b'}(B)$ for some connected DDS Y with $y \in \mathcal{P}_Y$, $|\mathcal{P}_Y| = |\mathcal{P}_X|$ and $a' = f_A^k(a)$ and $b' = f_B^k(b)$ for some $k \in \mathbb{N}$. Then $X \cong Y$.*

Proof. Let k' such that $f_B^{k'}(b') = f_B^{k'}(f_B^k(b)) = b$. Since $|\mathcal{P}_B|$ is a multiple of $|\mathcal{P}_A|$, this also implies $f_A^{k'}(a') = f_A^{k'}(f_A^k(a)) = a$. This means that $U_b(B)$ is the unroll of the same DDS as in $U_{b'}(B)$ but taking the k'-th successor of b as the root, and similarly for $U_a(A)$ and $U_{a'}(A)$. According to the same reasoning, $U_{y'}(Y)$ with $y' = f_Y^{k'}(y)$ satisfies $U_a(A) \star U_{y'}(Y) \cong U_b(B)$. Then, by Theorem 6.2.5, we have $U_{y'}(Y) \cong U_x(X)$. Then, $R_{|\mathcal{P}_Y|}(U_{y'}(Y)) \cong R_{|\mathcal{P}_X|}(U_x(X))$ implies $Y \cong X$. □

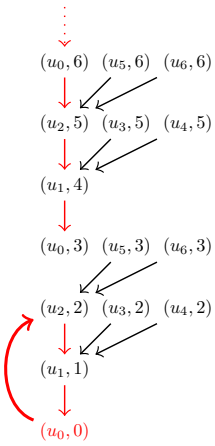


Figure 6.6 – According to Definition 6.2.3, given an infinite in-tree (or an unroll as in this case) and an integer ℓ (here equal to 2), we can define the roll as the connected component with a cycle of length $\ell + 1$ obtained by deleting the edge from $(u_0, 3)$ to $(u_2, 2)$ and adding another one from $(u_0, 0)$ to $(u_2, 2)$, in this case.

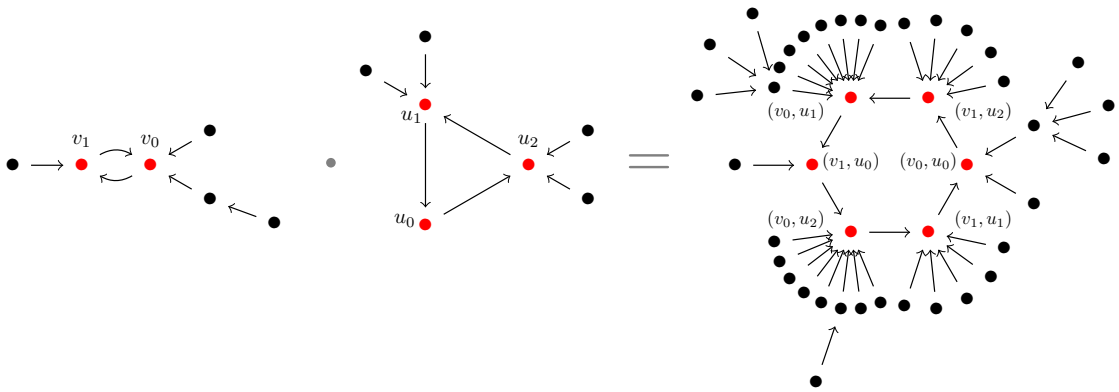


Figure 6.7 – The result of the roll operation on the three unrolls in Figure 6.5 is shown here (with ℓ equal to 2, 3, and 6 respectively). It can thus be seen that the product $\star$ of the unrolls is equivalent to the product of the DDS.

Now if we try to fix $b' = f_B^k(b)$, and if we consider again a k' such as $f_B^{k'}(b') = f_B^{k'}(f_B^k(b)) = b$, every pairing of b' with an $a \in \mathcal{P}_A$ will lead, by Lemma 6.2.6, to the same solution as fixing $f_B^{k'}(b') = b$ and $f_A^{k'}(a)$, which has already been done since we already considered every $a \in \mathcal{P}_A$ with b. Thus, fixing one b is sufficient to check all solutions.

Theorem 6.2.7. *Given an inequality $A \cdot X \supseteq B$ with A, B connected DDS and an integer $p_X \geq 1$, the inequality admits at most $|\mathcal{P}_A|$ connected solutions X having $|\mathcal{P}_X| = p_X$.*

Corollary 6.2.8. *An inequality $A \cdot X \supseteq B$ with A, B connected DDS admits at most $|\mathcal{P}_A|$ connected solutions X with $|\mathcal{P}_X| = p_X$ for each $p_X \geq 1$ such that $\mathrm{lcm}(|\mathcal{P}_A|, p_X) = |\mathcal{P}_B|$.*

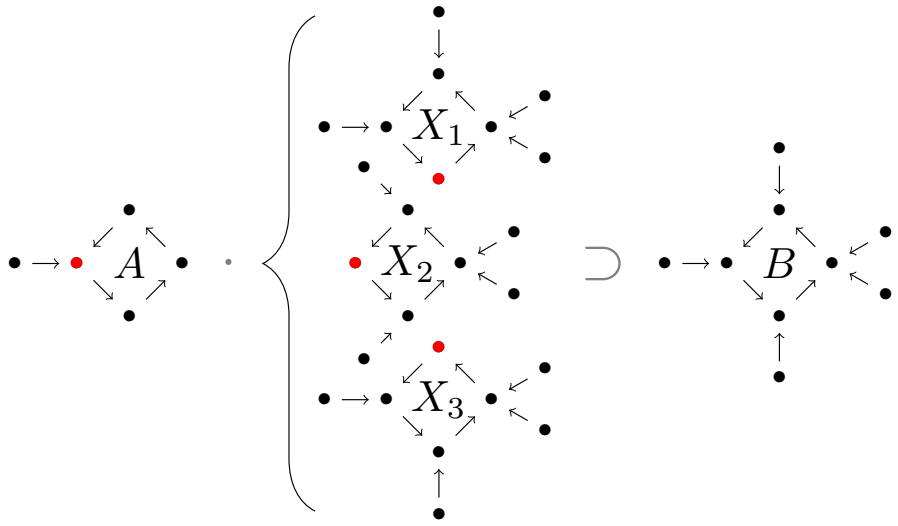


Figure 6.8 – An example of $A \cdot X \supseteq B$ with multiple solutions. The DDS A, multiplied with one of the X_i, generates 4 components, one of which will be isomorphic to B. For one X_i, the cyclic alignment resulting in B will be the one overlaying the periodic point with a transient in A with the periodic point without ones in X_i (*i.e.* the red ones). Note that this example can be generalised to create instances of $A \cdot X \supseteq B$ with an arbitrary large number of solutions for X.

The Figure 6.8 shows an example in the case of $\gcd(|\mathcal{P}_A|, |\mathcal{P}_X|) > 1$ where the number of solutions approaches the upper bound of Theorem 6.2.7, but we actually conjecture that a stronger statement holds when $\gcd(|\mathcal{P}_A|, |\mathcal{P}_X|) = 1$.

Conjecture 1. *An equation $A \cdot X = B$ with A, B connected DDS admits at most one solution.*

Having introduced Theorem 6.2.7, the goal then becomes to provide an algorithmic approach for finding solutions to an inequality $A \cdot X \supseteq B$ over connected DDS. We begin by considering it in its t-abstraction version (*i.e.* $T^A \cdot T^X \supseteq T^B$).

6.2.2 A polynomial algorithm for basic t-abstraction equations

In this section, we introduce a polynomial time algorithm to find all T^X (if they exists) such that $T^A \cdot T^X \supseteq T^B$. Recall that according to Proposition 6.1.2, the product of T^A and T^X generates a multiset, in which we want T^B to be one of its elements. The algorithm takes the t-abstraction

of two connected dynamics graphs A and B and a value p_X (an admissible length of the cycle of X) to reconstruct inductively the feasible t-abstractions of X. Remark that p_X can be any positive natural number such that $\mathrm{lcm}(|\mathcal{P}_A|, p_X) = |\mathcal{P}_B|$.

As explained in the previous section, we have at most $|\mathcal{P}_A|$ solutions since the upper bound to the number of solutions for DDS also applies for t-abstractions (*i.e.* each solution can potentially have a different t-abstraction). Then, we can take the matrix T^B as it is, and try to reconstruct T^X for each cyclic permutations of the lines of T^A. This corresponds to what was explained in the previous section, *i.e.* fixing a $b \in \mathcal{P}_B$ and checking whether a solution exists for $a \in \mathcal{P}_A$.

The algorithm goes through every element $T^B_{r,h}$, column by column, to compute T^X column by column. We know that:

$$T^B_{r,h} = T^X_{r,h} \otimes \mathcal{M}_1 + \mathcal{M}_2, \text{ where } \mathcal{M}_1 = \sum_{j=0}^{h-1} T^A_{r-j+i-1,h-j} \text{ and } \mathcal{M}_2 = T^A_{r+i-1,h} \otimes (\sum_{j=1}^{h-1} T^X_{r-j,h-j}).$$

As the algorithm proceeds level by level increasing the value of h, for each $T^B_{r,h}$, it computes the corresponding $T^X_{r,h}$ where the $\mathcal{M}_2$ is known, since all the $T^X_{r',h'}$ (for all $h' \in \{1, \ldots, h-1\}$ and $r' \in \{0, \ldots, p_X - 1\}$) have already been computed in a previous step of the algorithm. Note that, due to how the matrix is defined, it is always possible to know the expected cardinality of a $T^X_{r,h}$. It is the sum of the elements in $T^X_{r,h-1}$ due to the fact that, for each node, we have to calculate the number of predecessors.

This is the general idea. However, it may happen, when we go through the $T^B_{r,h}$ with $r > p_X - 1$, that the corresponding $T^X_{r,h}$ has already been computed, since its row index is considered modulo p_X. In this case, we only check if the product holds (an example of this mechanism is shown in Example 6.2.1).

At this point, the goal is to compute $T^X_{r,h}$ such that $T^B_{r,h} - \mathcal{M}_2 = T^X_{r,h} \otimes \mathcal{M}_1$[1]. Find such $T^X_{r,h}$ can be done as follows. Let $\mathcal{M}_3$ be the multiset $T^B_{r,h} - \mathcal{M}_2$, and d, d' be the maximum elements of respectively $\mathcal{M}_1$ and $\mathcal{M}_3$. Then, if T^B does indeed satisfies the product $T^A \cdot T^X$, there must exists a $y \in T^X_{r,h}$ which satisfies $d \cdot y = d'$, so we know that $\frac{d}{d'}$ is an element of $T^X_{r,h}$. We can save it and update $\mathcal{M}_3$ to be $\mathcal{M}_3 - [\frac{d}{d'}] \otimes \mathcal{M}_1$. We continue until either $\mathcal{M}_3 = \emptyset$ or $\mathcal{M}_3$ contains only zeros. In this last case, all the missing y must be zeros too. Indeed, $\frac{|\mathcal{M}_3|}{|\mathcal{M}_1|}$ zeros must be added to $T^X_{r,h}$. This procedure computes non ambiguously all the elements of $T^X_{r,h}$.

As for integer division, we have to consider the case where the denominator is zero (*i.e.* $\mathcal{M}_1$ filled with only zeros). For such case, it is impossible to define with certainty our $T^X_{r,h}$, since every multiset with the suitable cardinality would satisfy the equation. Fortunately, this particular scenario cannot happen in our case because $\mathcal{M}_1$, as defined above, contains the multiset $T^A_{r-h+1,1}$ which necessarily contains a positive integer. We will call *msDivision* the process of finding $T^X_{r,h}$ such that $\mathcal{M}_3 = T^X_{r,h} \otimes \mathcal{M}_1$.

At any point of the algorithm, if a contradiction is detected, it stops and tries another i (*i.e.* another cyclic permutations of the lines of T^A). Such contradictions occur, for instance, when we compute a difference between two multisets and the second one is not included in the first one,

[1] For two multisets $\mathcal{M}$ and $\mathcal{M}'$, we denote their subtraction as $\mathcal{M} - \mathcal{M}'$. It contains all those elements present in both sets such that the multiplicity in the former is greater than the one in the latter (with a multiplicity that is the difference of the original ones).

when the computed value $\frac{d}{d'}$ is not an integer, or when the cardinality of the multiset returned by $msDivision$ does not match the expected one.

An implementation of the approach just outlined is presented in Algorithm 17.

Algorithm 17: Polynomial t-abstraction

Input : T^A, T^B matrices of integer multisets, and p_X positive natural number
Output: A list of solutions for T^X

```
1  solutions ← ∅;
2  forall i ∈ {1, ..., | C_A |} do
3      h ← 1;
4      complete ← false;
5      error ← false;
6      expectedCardinalities ← [1] * p_X;
7      T^X ← emptyMatrix(p_X, h_B^max);
8      while complete==false and error==false do
9          complete ← true;
10         forall r ∈ {0, ..., | C_B | −1} do
11             if r < p_X then
12                 if expectedCardinalities[r]=0 then
13                     continue;
14                 M_2 ← T^A_{r+i−1,h} ⊗ (Σ_{j=1}^{h−1} T^X_{r−j,h−j});
15                 error, T^X_{r,h} ←msDivision(T^B_{r,h} − M_2, Σ_{j=0}^{h−1} T^A_{r−j+i−1,h−j});
16                 nextCardinality ←sum(T^X_{r,h});
17                 error ← error ∨ (| T^X_{r,h} |≠ expectedCardinalities[r]);
18                 if h == 1 then
19                     expectedCardinalities[r] ← nextCardinality − 1;
20                 else
21                     expectedCardinalities[r] ← nextCardinality;
22                 if nextCardinality > 0 then
23                     complete ← false;
24             else
25                 R ← T^A_{r,h} ⊗ (Σ_{j=0}^{h−1} T^X_{r−j+(i−1),h−j}) + T^X_{r+(i−1),h} ⊗ (Σ_{j=1}^{h−1} T^A_{r−j,h−j});
26                 error ← (T^B_{r,h} ≠ R);
27             if error then
28                 break;
29         h ← h + 1;
30     if error == false then
31         add T^X to solutions;
32 return solutions;
```

Example 6.2.1 – Let us explain how to reconstruct the T^X (with $p_X = 3$) considering the following T^A and T^B.

$$T^A = \begin{array}{c} \\ 0 \\ 1 \end{array} \begin{array}{c} \begin{array}{ccc} 1 & 2 & 3 \end{array} \\ \begin{pmatrix} [2] & [0] & \emptyset \\ [5] & [0,0,0,1] & [0] \end{pmatrix} \end{array}$$

$$T^B = \begin{array}{c} \\ 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{cccccc} 1 & 2 & 3 & 4 & 5 & 6 \end{array} \\ \begin{pmatrix} [4] & [0,0,10] & [2] & [1] & [0,0,0,0,0,0,0] & [0,0] \\ [20] & [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,2] & [0,0] & \emptyset & \emptyset & \emptyset \\ [6] & [0,0,0,0,5] & [0,0,0,0,0] & \emptyset & \emptyset & \emptyset \\ [10] & [0,0,0,0,0,0,2,3,4] & [0,0,0,0,0,0,0,0,0,5] & [0,0,0,2,4] & [0,0,0,0,0,5] & [0,0,0,0,0] \\ [8] & [0,0,0,0,0,0,0] & \emptyset & \emptyset & \emptyset & \emptyset \\ [15] & [0,0,0,0,0,0,0,0,0,0,0,0,1,2,4] & [0,0,0,0,0,0,0] & \emptyset & \emptyset & \emptyset \end{pmatrix} \end{array}$$

The algorithm starts by creating an empty matrix T^X, with p_X rows and h_B^{max} columns, and an *expectedCardinalities* vector. Then, at the beginning of the algorithm the only information known about T^X is that p_X equals 3. We thus start by initialising a matrix with 3 lines and 6 columns (the number of columns of T^B) filled with empty multisets.

The *expectedCardinalities* array, which stores at any time the mandatory cardinalities of the multisets at the next layer, is filled with p_X ones. We start by considering $i = 1$.

$$T^X = \begin{array}{c} \\ 0 \\ 1 \\ 2 \end{array} \begin{array}{c} \begin{array}{cccccc} 1 & 2 & 3 & 4 & 5 & 6 \end{array} \\ \begin{pmatrix} \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset \\ \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset \\ \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset \end{pmatrix}, \end{array}$$

$$expectedCardinalities = \begin{array}{ccc} 0 & 1 & 2 \\ [1, & 1, & 1] \end{array}.$$

For the first layer, *i.e.* $h = 1$, let us consider the computation made by the algorithm for r up to 2:

$$\begin{aligned} r = 0 \rightarrow & \mathcal{M}_2 = T^A_{0,1} \otimes \emptyset = [2] \otimes \emptyset = \emptyset, \\ & T^X_{0,1} = msDivision(T^B_{0,1} - \emptyset, T^A_{0,1}) = msDivision([4],[2]) = [2]; \\ r = 1 \rightarrow & \mathcal{M}_2 = T^A_{1,1} \otimes \emptyset = [5] \otimes \emptyset = \emptyset, \\ & T^X_{1,1} = msDivision(T^B_{1,1} - \emptyset, T^A_{1,1}) = msDivision([20],[5]) = [4]; \\ r = 2 \rightarrow & \mathcal{M}_2 = T^A_{0,1} \otimes \emptyset = [2] \otimes \emptyset = \emptyset, \\ & T^X_{2,1} = msDivision(T^B_{2,1} - \emptyset, T^A_{0,1}) = msDivision([6],[2]) = [3]. \end{aligned}$$

Recall that the lines indices of T^A are interpreted modulo $|\mathcal{C}_A|$.

Since we are on the first layer, the *expectedCardinalities* is updated to have the sum of the computed multisets minus 1 (since the cyclic preimage will be not considered). We will thus have

$$expectedCardinalities = \begin{array}{ccc} 0 & 1 & 2 \\ [1, & 3, & 2] \end{array}.$$

From $r = 3$, the algorithm will now enter in a verification phase, and we will have:

$$r = 3 \rightarrow T^A_{1,1} \otimes T^X_{0,1} = T^B_{3,1}, [5] \otimes [2] = [10];$$
$$r = 4 \rightarrow T^A_{0,1} \otimes T^X_{1,1} = T^B_{4,1}, [2] \otimes [4] = [8];$$
$$r = 5 \rightarrow T^A_{1,1} \otimes T^X_{2,1} = T^B_{5,1}, [5] \otimes [3] = [5].$$

Recall that also the lines indices of T^X are interpreted modulo p_X.

At this point, we have

$$T^X = \begin{array}{c} \\ 0 \\ 1 \\ 2 \end{array} \begin{array}{c} \begin{array}{cccccc} 1 & 2 & 3 & 4 & 5 & 6 \end{array} \\ \begin{pmatrix} [2] & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset \\ [4] & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset \\ [3] & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset \end{pmatrix} \end{array}.$$

For the second layer, *i.e.* $h = 2$, the algorithm proceeds in the same way:

$$\begin{aligned}
r = 0 \rightarrow & \mathcal{M}_2 = T^A_{0,2} \otimes T^X_{2,1} = [0] \otimes [3] = [0], \\
& T^X_{0,2} = msDivision(T^B_{0,2} - [0], T^A_{1,1} + T^A_{0,2}) = msDivision([0, 10], [0, 5]) = [2]; \\
r = 1 \rightarrow & \mathcal{M}_2 = T^A_{1,2} \otimes T^X_{0,1} = [0, 0, 0, 1] \otimes [2] = [0, 0, 0, 2], \\
& T^X_{1,2} = msDivision(T^B_{1,2} - [0, 0, 0, 2], T^A_{0,1} + T^A_{1,2}) = \\
& = msDivision([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 1, 2]) = [0, 0, 0]; \\
r = 2 \rightarrow & \mathcal{M}_2 = T^A_{0,2} \otimes T^X_{1,1} = [0] \otimes [4] = [0], \\
& T^X_{2,2} = msDivision(T^B_{2,2} - [0], T^A_{1,1} + T^A_{0,2}) = \\
& = msDivision([0, 0, 0, 5], [0, 5]) = [0, 1].
\end{aligned}$$

No error is triggered since the cardinalities of the computed multisets match the expected ones. The *excpectedCardinalites* array is then updated to be

$$expectedCardinalities = \begin{array}{c} \begin{array}{ccc} 0 & 1 & 2 \end{array} \\ [2, \quad 0, \quad 1] \end{array}.$$

We can again check if these multisets work for the remaining lines.

$$\begin{aligned}
r = 3 \rightarrow & T^A_{1,2} \otimes T^X_{0,2} + T^A_{0,1} \otimes T^X_{0,2} + T^A_{1,2} \otimes T^X_{2,1} = T^B_{3,2} \\
& [0, 0, 0, 1] \otimes [2] + [2] \otimes [2] + [0, 0, 0, 1] \otimes [3] = [0, 0, 0, 0, 0, 0, 2, 3, 4] \\
r = 4 \rightarrow & T^A_{0,2} \otimes T^X_{1,2} + T^A_{1,1} \otimes T^X_{1,2} + T^A_{0,2} \otimes T^X_{0,1} = T^B_{4,2} \\
& [0] \otimes [0, 0, 0] + [5] \otimes [0, 0, 0] + [0] \otimes [2] = [0, 0, 0, 0, 0, 0, 0] \\
r = 5 \rightarrow & T^A_{1,2} \otimes T^X_{2,2} + T^A_{0,1} \otimes T^X_{2,2} + T^A_{1,2} \otimes T^X_{1,1} = T^B_{5,2} \\
& [0, 0, 0, 1] \otimes [0, 1] + [2] \otimes [0, 1] + [0, 0, 0, 1] \otimes [4] = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 2, 4]
\end{aligned}$$

No contradiction is found, the algorithm can go on.

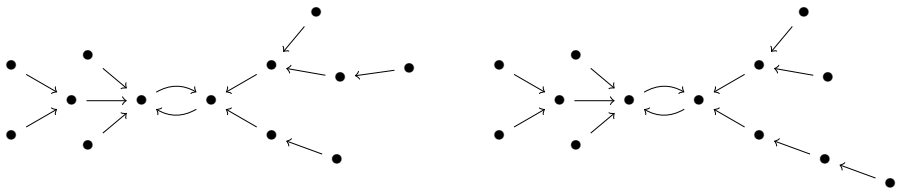
$$T^X = \begin{array}{c} \\ 0 \\ 1 \\ 2 \end{array} \begin{array}{c} \begin{array}{cccccc} 1 & 2 & 3 & 4 & 5 & 6 \end{array} \\ \begin{pmatrix} [2] & [2] & [0,1] & [2] & [0,1] & [0] \\ [4] & [0,0,0] & \emptyset & \emptyset & \emptyset & \emptyset \\ [3] & [0,1] & [0] & \emptyset & \emptyset & \emptyset \end{pmatrix} \end{array}$$

In this example, we have shown how the first two columns of the matrix can be computed. If one continues, the values shown in grey in the matrix are obtained.

Let us now analyse the complexity of the algorithm. Each update of $\mathcal{M}_3$ in $msDivision$ requires linear time *w.r.t.* the size of $\mathcal{M}_3$. We repeat it for each element of $T^X_{r,h}$, namely, $\frac{|\mathcal{M}_3|}{|\mathcal{M}_1|}$ times, with $|\mathcal{M}_1|$ possibly equal to 1. Therefore, *msDivision* complexity is in $O(|\mathcal{M}_3|^2)$. Due to the definition of t-abstractions, we know that T^B contains $|\mathcal{X}_B|$ integer values partitioned into the different $T^B_{r,h}$. Since the $msDivision$ is applied to all $\mathcal{M}_3$ (or $T^B_{r,h}$ - $\mathcal{M}_2$ with $\mathcal{M}_2$ potentially empty), the worst case is when $msDivision$ is applied to one $\mathcal{M}_3$ containing $|\mathcal{X}_B|$ elements. If we consider also that we go through the matrix, for a certain i, the complexity is in $O(|\mathcal{P}_B| \cdot h_B^{max} + |\mathcal{X}_B|^2)$. Since we evaluate i from 0 to $|\mathcal{P}_A| - 1$, the total complexity of the algorithm is $O(|\mathcal{P}_A| \cdot (|\mathcal{P}_B| \cdot h_B^{max} + |\mathcal{X}_B|^2))$. Note that this algorithm does not ensure that the corresponding equation over DDS has a solution. However, it ensures that if the latter has solutions, they satisfy one of the t-abstractions found by the algorithm.

6.2.3 An exponential algorithm for basic equations over DDS

This section introduces an exponential time algorithm which takes in input the dynamics graphs of A and B, and an integer p_X and outputs all X which satisfy $A \cdot X \supseteq B$. The polynomial algorithm of the previous section is able to find, all T^X such that $T^A \cdot T^X \supseteq T^B$, if they exist. Unfortunately, these solutions lack information to reconstruct instantly the dynamics of X. Since the multisets are not ordered, we cannot know which indegree of a $T^X_{r,h}$ is related to which node in $\mathcal{T}^X_{r,h-1}$. The Figure 6.9 shows this potential ambiguity.



$$\begin{array}{c} \\ 0 \\ 1 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{pmatrix} [3] & [2,1] & [0,0,1] & [0] \\ [4] & [0,0,2] & [0,0] & \emptyset \end{pmatrix} \end{array}$$

Figure 6.9 – Two non-isomorphic graphs (top) having the same t-abstraction (bottom).

6.2.3.1 Naive reconstruction

The most intuitive way to reconstruct the graph X, starting from a T^X (computed with the polynomial algorithm), is to test all the connected DDS satisfying the t-abstraction. This corresponds to test all the ways to connect the elements of a $T^X_{r,h}$ to the elements of $\mathcal{T}^X_{r,h-1}$ (level by level). Since, $|T^X_{r,h}| = |\mathcal{T}^X_{r,h-1}|$, for a level h and an index r of a node in the cycle, we have $|T^X_{r,h}|!$ possibilities. Then, there are

$$\prod_{(r,h)\in\{0,\dots,p_X-1\}\times\{1,\dots,h_x^{max}\}} (|T^X_{r,h}|!)$$

possible systems (not up to isomorphism). However, we know that the polynomial algorithm returns at most one t-abstraction solution for each cyclic permutation of T^A (*i.e.* one for each cyclic alignment of A on B). Then, according to Theorem 6.2.7, for each T^X returned by the polynomial method, there is just one feasible graph that is solution of $A \cdot X \supseteq B$. To verify if a system is a solution, one can verify if, in the graph obtained from the product of itself with A, the component corresponding to the alignment considered before is isomorphic to B.

6.2.3.2 An improved approach

Knowing the full dynamics of A and B allows us to associate a certain degree in a multiset of T^A or T^B with the corresponding node in the graph. In practice, this can be done by providing the multisets with an order such that the k-th node in a $\mathcal{T}^A_{r,h}$ have the corresponding k-th indegree in $T^A_{r,h+1}$ (the same applies for B). In the same way as the polynomial algorithm, this new algorithm (Algorithms 18 and 19) will go through every layer h of the transients, and for each one of them, reconstructs the layer of X.

At the beginning, X contains just a cycle of length p_X. At level $h = 1$, the multisets computed by the polynomial algorithm are sufficient to reconstruct the dynamics of the first layer of X. Indeed, all the $T^X_{r,1}$ have only one element d. Then, we just attach, to the r-th cyclic node, d new nodes. At $h = 2$, the multisets computed by the polynomial algorithm are still sufficient to reconstruct the dynamics because all nodes in $\mathcal{T}^X_{r,1}$ are equivalent from a dynamics point of view. Then, for each $d \in T^X_{r,2}$, we assign d predecessors to an arbitrary node in $\mathcal{T}^X_{r,1}$ (which must still be a leaf). Now, the nodes in $\mathcal{T}^X_{r,2}$ are potentially no longer equivalent. For this reason, starting from $h = 3$, the algorithm will make assumptions on which node of X at the layer h has generated which node of B at the same layer. These assumptions allows to reconstruct the next layer of X without ambiguity. We will call these assumptions the *origins* of the nodes of B. From a graph point of view, the origins will be only labels on nodes of X and B such that all the nodes in B with origin o are supposed to be generated by the node with label o in X. Let $\mathcal{O}_{r,h}$ be the set of origins of the nodes in a certain $\mathcal{T}^B_{r,h}$.

We now consider the computation of a generic layer $h \geq 3$. Let suppose that we know the origins of the nodes in $\mathcal{T}^B_{r,h-2}$ (we will explain later how origins can be initialised). With this information, we can partition the multiset $T^B_{r,h}$ into submultisets depending on the origin of the nodes in $\mathcal{T}^B_{r,h-2}$. Let $\mathcal{T}^B_{r,h|o}$ be the set of all $v \in \mathcal{T}^B_{r,h}$ such that $f_B(v)$ has origin o. Then, let $T^B_{r,h|o}$ be the multiset of indegrees of the nodes $v \in \mathcal{T}^B_{r,h-1|o}$. According to these definitions, $T^B_{r,h} = \sum_{o\in\mathcal{O}_{r,h-2}} T^B_{r,h|o}$. We can also partition the multiset of unknowns $T^X_{r,h}$, as $T^X_{r,h} = \sum_{o\in\mathcal{O}_{r,h-2}} T^X_{r,h|o}$.

Hence, we can rewrite the equivalence of Proposition 6.1.2 as follows:

$$\sum_{o\in\mathcal{O}_{r,h-2}} T^B_{r,h|o} = \sum_{o\in\mathcal{O}_{r,h-2}} \left(T^X_{r,h|o} \otimes \sum_{j=0}^{h-1} T^A_{r-j,h-j} \right) + T^A_{r,h} \otimes \sum_{j=1}^{h-1} T^X_{r-j,h-j}. \tag{6.8}$$

Note that there are some nodes in $\mathcal{T}^B_{r,h-1}$ which are not generated from a node in $\mathcal{T}^X_{r,h-1}$. These nodes are marked with a special origin (noted -1). The indegrees of these nodes are the ones found in the second term, *i.e.* $T^A_{r,h} \otimes \sum_{j=1}^{h-1} T^X_{r-j,h-j}$, which we denote $\mathcal{M}_2$ as in the previous section. The peculiarity of -1 origin is that all the nodes attached to a -1 origin will also have -1 as origin. In fact, these are the nodes (belonging to a level of B) that come from the multiplication of lower levels of X with the last level of A. Then, we should always have $T^B_{r,h|-1} = \mathcal{M}_2$ (line 20 Algorithm 18).

By considering the same definition of $\mathcal{M}_1$ of Section 6.2.2, we can now decompose the product formula into smaller ones, by grouping together the multisets which have the same origin. We thus have $T^B_{r,h|o} = T^X_{r,h|o} \otimes \mathcal{M}_1$ for all $o \in \mathcal{O}_{r,h-2} \setminus \{-1\}$, meaning that we can compute each $T^X_{r,h|o}$ with the $msDivision$. Then, we have to assign to each node of $\mathcal{T}^X_{r,h-1|o}$ one degree from $T^X_{r,h|o}$. However, we can notice that we brought the problem back to the special case of $h = 2$. Indeed, each node of $\mathcal{T}^X_{r,h-1|o}$ have the exact same successor, the only one with the label o. This means that we can again, for each $d \in T^X_{r,h|o}$, attach d new nodes to an arbitrary node of $\mathcal{T}^X_{r,h-1|o}$ (line 25 Algorithm 18). This mechanism is the key to reconstruct X layer by layer without ambiguity. Incidentally, each time we add a k-th new node v in X, we set its label equal to k. This is handled by the $attachNewNodes$ method which, in addition to create the nodes, assigns to them the corresponding origin and returns $true$ if at least a node is created (Note that the $cyclicGraph$ function also automatically assign origins while creating the nodes). Hence, this ensures that all nodes in X have a unique label. The returned value, for its part, allow us to know if X is completely constructed.

It remains now an issue: how to assign origins to the nodes in the layer of B. Up to now, we assigned to each node in $\mathcal{T}^X_{r,h-1}$ a number of predecessors. This means that we also know for each node $v \in \mathcal{T}^X_{r,h-1|o}$, all the degrees induced in $T^B_{r,h|o}$. Indeed, if v has an incoming degree d, there is a submultiset $\mathcal{M}_v = [d] \otimes \mathcal{M}_1$ included in $T^B_{r,h|o}$. We must then apply the label of v as the origin of all the corresponding nodes in $\mathcal{T}^B_{r,h-1|o}$. However, several choices may be possible. Intuitively, for each element d' in $\mathcal{M}_1$, we assign the label of v as origin to any node w in $\mathcal{T}^B_{r,h-1|o}$ that has $d \cdot d'$ predecessors but two nodes in $\mathcal{T}^B_{r,h-1|o}$ can have the same amount of predecessor, and both can be valid choices at this point[2]. An approach is to choose one, and try to reconstruct the layers above (lines 31 and 33 Algorithm 19). If at a further level it is not possible to reconstruct the layer of X, the algorithm backtracks to the last choice made, and tries another origin assignation (lines 35 and 36). The enumeration of all the valid choices for each layer is the cause of the exponential time complexity.

Note that the -1 origins have to be initialised at some point. The first layer where $\mathcal{M}_2$ term appears is at height $h = 2$. The origins are not needed here to construct X but must be affected

[2]The $enumerateAssignemt$ function takes as parameters a list of constraints over origins. The constraints are triplets made of 1. the name of the origin, 2. the multiset M_v of indegrees on which the origins must be assigned, and 3. a set of nodes among which we can assign it. It outputs then all the valid assignments (as hashmaps linking nodes to origins).

Algorithm 18: FindSolutionAlignment

Input : $i \in \{1, \ldots, |\mathcal{C}_A|\}$, A, B, X connected DDS and h positive natural number
Output: $true$ if a solution has been reconstructed, $false$ otherwise
(X is modified to be the solution, if it exists, and is not returned)

$originsData \leftarrow \emptyset$;
$nodeAttached \leftarrow false$;
forall $r \in \{1, \ldots, |\mathcal{C}_X|\}$ **do**
  if $h == 1$ **then**
    $error, T^X_{r,1} \leftarrow$ msDivision$(T^B_{r,1}, T^A_{r+i-1,1})$;
    $nodeAttached \leftarrow$ attachNewNodes$(g_X(r), T^X_{r,1}[0] - 1)$;
    $originsData$.append((getOrigin$(g_X(r)), T^B_{r,1}, [g_B(r)]))$;
  if $h == 2$ **then**
    $\mathcal{M}_1 \leftarrow T^A_{r+i-2,1} + T^A_{r+i-1,2}$;
    $error, T^X_{r,2} \leftarrow$ msDivision$(T^B_{r,2} - (T^A_{r+i-1,2} \otimes T^X_{r-1,1}), \mathcal{M}_1)$;
    forall $k \in \left\{0, \ldots, |T^X_{r,2}| - 1\right\}$ **do**
      $nodeAttached \leftarrow$ attachNewNodes$(\mathcal{T}^X_{r,1}[k], T^X_{r,2}[k])$;
      $originsData$.append((getOrigin$(\mathcal{T}^X_{r,1}[k]), T^X_{r,2}[k] \otimes \mathcal{M}_1, \mathcal{T}^B_{r,1}))$;
    $originsData$.append$((-1, T^A_{r+i-1,2} \otimes T^X_{r-1,1}, \mathcal{T}^B_{r,1}))$;
  if $h > 2$ **then**
    $\mathcal{M}_1 \leftarrow \sum_{j=0}^{h-1} T^A_{r-j+i-1,h-j}$;
    $\mathcal{M}_2 \leftarrow T^A_{r+i-1,h} \otimes (\sum_{j=1}^{h-1} T^X_{r-j,h-j})$;
    forall $o \in \mathcal{O}_{r,h-2}$ **do**
      if $o == -1$ **then**
        $error \leftarrow (\mathcal{M}_2 \neq T^B_{r,h|-1})$;
        $originsData$.append$((-1, \mathcal{M}_2, \mathcal{T}^B_{r,h-1|-1}))$;
      else
        $error, T^X_{r,h|o} \leftarrow$msDivision$(T^B_{r,h|o}, \mathcal{M}_1)$;
        forall $k \in \left\{0, \ldots, |T^X_{r,h|o}| - 1\right\}$ **do**
          $nodeAttached \leftarrow$ attachNewNodes$(\mathcal{T}^X_{r,h-1|o}[k], T^X_{r,h|o}[k])$;
          $originsData$.append((getOrigin$(\mathcal{T}^X_{r,h-1|o}[k]), T^X_{r,h|o}[k] \otimes \mathcal{M}_1, \mathcal{T}^B_{r,h-1|o}))$;
if $error$ *or* $\neg(truncate(A, h) \cdot X \supseteq truncate(B, h))$ **then**
  return *false*;
if $\neg nodeAttached$ *and* $(A \cdot X \supseteq B)$ **then**
  return *true*;
forall $origins \in$ *enumerateAssignements*$(originsData)$ **do**
  applyOrigins$(B, origins)$;
  if *FindSolutionAlignment*$(i, A, B, X, h + 1)$ **then**
    return *true*;
  detachOrigins$(B, origins)$;
$X \leftarrow$ truncate$(X, h - 1)$;
return *false*;

for the layer $h = 3$. We proceed as explained for $h = 2$ and then we only affect a -1 origin to all the nodes in $\mathcal{T}^B_{r,1}$ remaining without origins.

For the verification part, at each height h after having reconstructed the layer, we check that X, multiplied by A considered only up to layer h, contains B (again only up to height h). The truncation of the graphs, consisting in returning subgraphs by removing transients in higher levels, and (in Algorithm 19) it is done by the function $truncate$.

As for the polynomial algorithm, we actually repeat this whole mechanism for all cyclic alignments (Algorithm 19). According to Section 6.2.1, considering each value of $a \in \mathcal{C}_A$ as the first attractor, we can enumerate the solutions of the equation $A \cdot X \supseteq B$. For this reason, even if we have not tested all the feasible origin assignments for a certain a, after finding a solution the algorithm starts calculating a new solution for another a.

Algorithm 19: ExponentialSolver

Input : A, B connected functional graphs, and p_X positive natural number
Output: A list of solutions for X

```
1 solutions ← ∅;
2 forall i ∈ {1, ..., | C_A |} do
3     X ← cyclicGraph(p_X) ;        ▷ cyclicGraph also assign new origins on X
4     if FindSolutionAlignment(i, A, B, X, 1) then
5         solutions.append(clone(X)) ; ▷ A copy of X is saved as a solution
6 return solutions;
```

Before introducing an optimisation, let us use an example to clarify what has been explained so far.

Example 6.2.2 – Let us show an example for the reconstruction of one layer of X (the red one in Figure 6.10) and the feasible origin assignments on B. To construct this third layer of the dynamics of X, $T^B_{0,3}$ is divided by the origins of the nodes of $\mathcal{T}^B_{0,1}$: 1, 2 and -1. Firstly, $T^B_{0,3|1} = [4, 2, 2, 1, 0, 0]$ (the indegrees of v_0 to v_3 plus the ones of the two leaves v_9 and v_{10}), and also $T^B_{0,3|2} = [2, 1, 0]$ and $T^B_{0,3|-1} = [0, 0, 0]$. We then compute $\mathcal{M}_1 = T^A_{0,3} + T^A_{0,2} + T^A_{0,1} = [0] + [1] + [2]$ and $\mathcal{M}_2 = T^A_{0,3} \otimes (T^X_{0,2} + T^X_{0,1}) = [0] \otimes ([2, 1] + [3]) = [0, 0, 0]$. After checking that $\mathcal{M}_2$ is equal to $T^B_{0,3|-1}$, we can now compute, with $msDivision$, $T^X_{0,3|1}$ such that $\mathcal{M}_1 \otimes T^X_{0,3|1} = T^B_{0,3|1}$ or $[2, 1, 0] \otimes T^X_{0,3|1} = [4, 2, 2, 1, 0, 0]$. We found that $T^X_{0,3|1} = [1, 2]$, which we can directly apply to the nodes in X with label 3 and 4. We compute in the same way $T^X_{0,3|2} = [1]$. Finally, we assign to the nodes in $\mathcal{T}^B_{0,2}$ the origins 3, 4 and 5. v_4,v_5 and v_{11} will receive the origin 5 and v_6,v_7 and v_8 will receive the origin -1 since there is no choice. For the origins 3 and 4, we can fix one of each for the two nodes v_9 and v_{10} (the choices for the leaves do not matter). For the nodes (v_0, v_1, v_2, v_3), since both nodes in X can produce a degree 2 in B, we will have to test the two assignments $(3, 3, 4, 4)$ and $(3, 4, 3, 4)$.

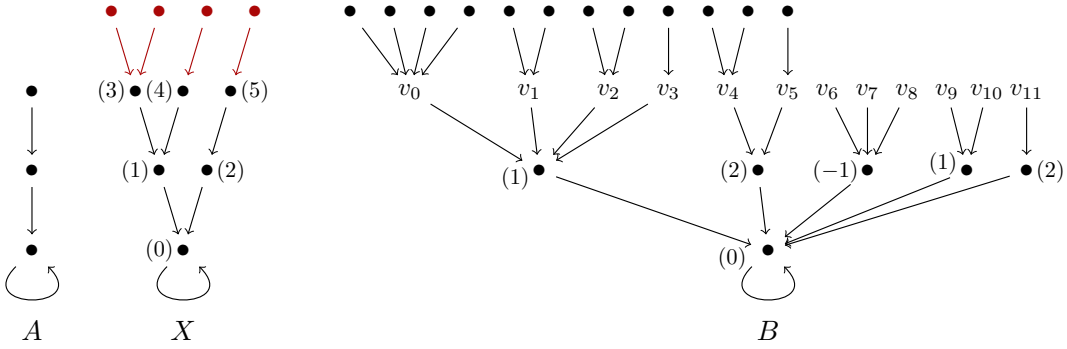


Figure 6.10 – Three DDS A, X and B (origins are shown in parentheses).

6.2.3.3 Optimised origins affectation

To reduce the number of possibilities and to avoid impossible affectations, we consider the height of the transients in A and B. For a generic node v of a DDS, let $\mathscr{H}(v)$ be the height of the subtree (which is an in-tree) rooted on v (with just the root that can be a cyclic node, in the case of $v \in \mathcal{C}$). For each node $v \in \mathcal{T}^X_{r,h-1|o}$ with d predecessors, we know that it generates the subset $\mathcal{M}_v = [d] \otimes \mathcal{M}_1$. Let us consider an element d' of $\mathcal{M}_1$ and the corresponding node u in A. According to our reasoning, one can give the label of v to any node in $\mathcal{T}^B_{r,h-1|o}$ with $d \cdot d'$ predecessors. However, according to the product definition, a node w of B, generated by u of A and v of X, have $\mathscr{H}(w) = \min\{\mathscr{H}(u), \mathscr{H}(v)\}$. Then, we can only choose nodes w in $\mathcal{T}^B_{r,h-1|o}$ with $d \cdot d'$ predecessors and such that $\mathscr{H}(w) \leq \mathscr{H}(u)$. In line with the direct product definition, we can use this optimisation technique just if u and v are transients nodes.

Example 6.2.3 – After having computed the second layer of X, one needs to assign one origin among three possible ones (1, 2 and -1) for each of the eight nodes w_i in B.

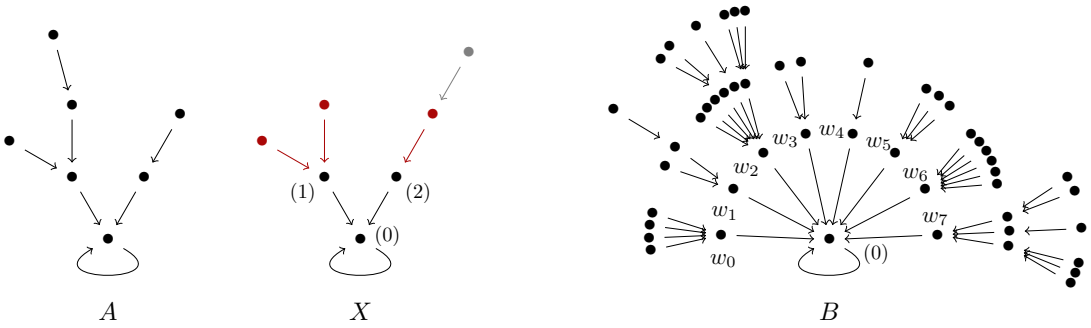


Figure 6.11 – Three DDS A, X and B (origins are shown in parentheses).

Let us compute, for each origin, the multiset of degrees $\mathcal{M}_v$.

$$o = -1 \rightarrow T^A_{0,2} \otimes T^X_{0,1} = [1,2] \otimes [3] = [3,6];$$
$$o = 1 \rightarrow [2] \otimes (T^A_{0,2} + T^A_{0,1}) = [2] \otimes [1,2,3] = [2,4,6];$$
$$o = 2 \rightarrow [1] \otimes (T^A_{0,2} + T^A_{0,1}) = [1] \otimes [1,2,3] = [1,2,3].$$

Following only the rules on the indegrees, some nodes can be given origins without ambiguity. The node w_0, being the only one having indegree 4 must have origin 1 and w_4, with indegree 1, must have origin 2. However, several possibilities exist for the other ones:

- w_1 and w_3 (with both indegree 2) can either have the origin 1 or 2,
- w_5 and w_7 (with both indegree 3) can either have the origin -1 or 2,
- w_2 and w_6 (with both indegree 6) can either have the origin -1 or 1.

This gives us 8 possible affectations. However, by looking at the heights, we can reduce the number of feasible affectations to only one.

First, the nodes w_i in B originating from a node in $u \in \mathcal{T}^A_{0,1}$ and the cyclic point of X (and hence having a -1 origin), must have $\mathscr{H}(w_i) = \mathscr{H}(u)$. As previously said, the height can not be higher. Moreover, there must also be, in the subtree rooted in w_i, a subtree which is the exact copy of the corresponding one rooted in u. This is true since that we are considering the Cartesian states with the cyclic point of X and the nodes in the subtree of A. As a consequence, between w_5 and w_7, just w_5 can take origin -1. Then, w_2 takes origin -1 too since we need to find the copy of the subtree containing the highest transient of A. In conclusion, w_6 and w_7 can not have the origin -1.

In the second hand, the node having indegree 2 and origin 1 in B, can not have a rooted subtree of height exceeding 1 since it is supposed to originate from the node u' (in $\mathcal{T}^A_{0,1}$) with just one predecessor and $\mathscr{H}(u') = 1$. Therefore, w_1 can be excluded. Indeed, the only feasible affectation of origins, for the nodes from w_0 to w_7, is 1, 2, -1, 1, 2, -1, 1 and 2.

To compute the complexity, we must study the total number of assignation for a certain layer. To do so, one have to know the number of distinct $T^B_{r,h|o}$ at this layer and their lengths. The first corresponds to the number of possible origins (*i.e.* the number of node in X in a layer), which is $\frac{|\mathcal{X}_X|}{h_B^{max}}$ (h_X^{max} will be at most h_B^{max}). For their sizes, the number of nodes in a layer of B can be bounded as $\frac{|\mathcal{X}_B|}{h_B^{max}}$. However, the algorithm consider only the first p_X values of r, then we have $\frac{|\mathcal{X}_B| \cdot p_X}{h_B^{max} \cdot |\mathcal{C}_B|}$. From the explanation of the algorithm, the combinatorial affectation of origins is computed on the equal indegrees in a $T^B_{r,h|o}$. We assume that the number of different incoming degrees is the maximum one, which we will denote as d_{max}. Therefore, the total number N of feasible assignments of origins for a layer is $\left(\frac{|\mathcal{X}_B| \cdot p_X}{|\mathcal{X}_X| \cdot |\mathcal{C}_B| \cdot d_{max}}!\right)^{\frac{|\mathcal{X}_X| \cdot d_{max}}{h_B^{max}}}$. Since the origin are computed in every layer and this whole process is repeated for each alignment of A and B, the total complexity is in $O(|\mathcal{C}_A| \cdot (N^{h_B^{max}}))$.

6.2.3.4 An experimental evaluation

Let us evaluate the exponential version of the algorithm to investigate the performances over different instances[3]. The results are shown in Figure 6.12. For this experimental evaluation, random instances of equations over connected DDS based on different parameters have been generated. For the three plots above in Figure 6.12, cases with $|\mathcal{C}_A|$ and p_X prime numbers (this ensures $\gcd(|\mathcal{C}_A|, p_X) = 1$) have been tested, since they represent the worst-case scenario in which all the nodes of B are in one connected component. As the total number of nodes is always the same for A and X, one can remark that the axes also give the information on the number of transient points, (inversely proportional to the cycle lengths). The naive reconstruction algorithm of X (left column of Figure 6.12) has been compared to the presented approach, with (right) and without (middle) height optimisation. As one might have guessed, the naive algorithm strongly depends on the number of transient nodes of X. Indeed, the naive one does not take into account the information on the dynamics of B and A, but just their t-abstractions to compute T^X. The approach explained above, on the other hand, applies the combinatorial reasoning on elements of B instead of X. It thus depends on the number of transient of A and X, as B does. As explained in the section above, the $T^B_{r,h|o}$ comes from a multiplication of $\mathcal{M}_1$, which comes from A, and $T^X_{r,h|o}$ which comes from X. If one of the two is smaller, the resulting $T^B_{r,h|o}$ will be smaller as well. This will

[3]The experiments have been done on an AMD EPYC 7301 processor running at 2.2GHz, with 128GiB of RAM and 16 cores. The algorithm has been implemented in Python 3.9.12 and runs with version 7.3.9 of PyPy.

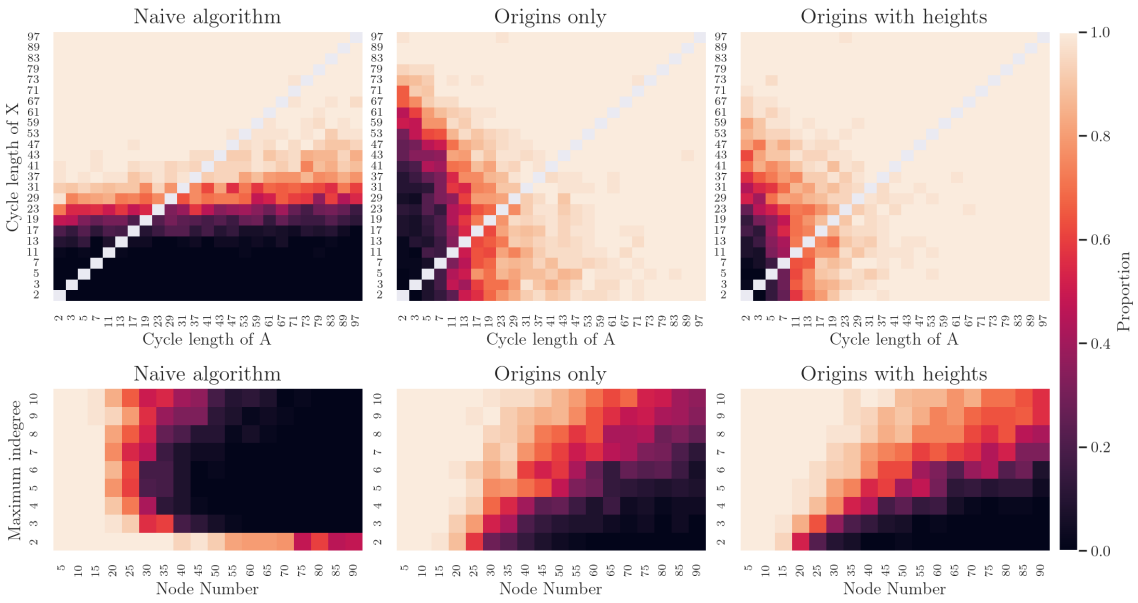


Figure 6.12 – Plots showing the proportion of random instances solved in less than 1 second over the 50 cases executed for each box. The three plots above are based on 30000 instances with A and X of 100 nodes each, and B of 10000 nodes, with increasing cycle lengths for A (x-axis) or X (y-axis). The three plots below are on 9500 instances with A, X and B having only one cyclic node and an increasing number of nodes (x-axis) and maximum indegree (y-axis), for both A and X.

decrease the number of feasible affectations. The drawback is when X has much fewer transient points than A. In this case, the combinatorics on X would be more efficient, which explains why the naive algorithm has better results in the far left region of the plots. The optimisation over the origins affectation turns out to be relevant to delay the exponential growth of the algorithm. Note that, in emphasis on the conjecture, the time is considered up to the moment that the first solution has been found. If the goal is the enumeration of all the solutions, the time would be multiplied by a linear factor.

As smaller cycle lengths tends to increase the execution time, the extreme case of $|\mathcal{C}_A|$, $|\mathcal{C}_X|$ and $|\mathcal{C}_B|$ equal to 1 is studied too (*i.e.* the plots below). As in the previous study, the naive algorithm depends almost exclusively on the number of transient points of X. It appears to be efficient only when the incoming degree equals 2. It is a special case where a lot of symmetries are involved and where a great number of distinct permutations lead to isomorphic dynamics. In the two other plots, the incoming degree plays a greater role. One may think that greater indegrees imply wider layers and more combinatorics. Surprisingly, the maximum incoming degree is associated with a shorter execution time. Actually, the combinatorial aspect is correlated to the number of nodes with the same indegree. This allows us to have much better results than the naive approach as the maximum indegree grows. One can notice that also in these cases the combinatorial optimisation improves the performance of the algorithm. [4]

In conclusion, we experimentally observed that this algorithm has good results in most cases and can compute solutions of equations with large number of nodes in the known term, despite its exponential runtime. We also introduced some further optimisations by considering interesting properties of the direct product computed over functional graphs, and succeeded to push back a little more the exponential cost. However, also the polynomial approach is interesting since it finds all solutions of basic equations over t-abstractions and identifies some impossible equations over DDS in polynomial time. In fact, if there are no possible solutions according to the abstraction, one can be sure that the equation over dynamics graphs is impossible too.

[4] The performance loss for low degrees comes from the graph generation routine which tends to create balanced in-trees. Thus, the height upgrade cannot overcome the extra time of the optimisation itself.

Conclusion et Perspectives

In this thesis, an abstractions-based approach to solving multivariate polynomial equations (with a constant right-hand term and one variable per monomial) is proposed. These equations are chosen as they prove to be a good starting point for investigating this type of approach and allow hypotheses on DDS to be modelled. In this manuscript, it is also intended to enumerate the solutions of a DDS equation to be able to obtain the maximum amount of information from it (different factorisations/decompositions or values for the unknowns supporting the hypothesis modelled by the polynomial).

Three abstractions are introduced.

The c-abstraction is concerned with the number of states of the unknowns of the polynomial. It transforms the original equation in a new one in which each DDS (*i.e.* coefficients, unknowns, and right-hand side) is replaced by the cardinality of the set of states of the corresponding system. In Chapter 4, we saw that sums and products of DDS correspond to sums and products of the c-abstractions (*i.e.* natural numbers), and that the enumeration of the solutions of the abstraction can be done with a Multi-valued Decision Diagram.

After finding the potential number of states for each variable, the goal becomes to reconstruct the feasible dynamic behaviour between them.

The a-abstraction focus on the dynamics between the periodic points. Indeed, it allows to identify the number of periodic points and the cyclic behaviour of the variables. Concerning the operations of the semiring, the sum of DDS turns out to be just the disjoint union of the cycles of the addends, while, the result of a product operation depends on the lengths and the number of cycles involved. An important part of Chapter 5 shows that enumerating the solutions of an a-abstraction equation corresponds to enumerating the solutions of a finite number of systems involving just equations of the form $C_p^1 \odot \mathring{\mathbb{X}} = C_q^n$ (*i.e.* basic equations). This is possible thanks to some algebraic transformations called contraction steps. Indeed, a fundamental step becomes to be able to enumerate the solutions of a basic equation. Then, Section 5.2.1 explores the complexity of deciding if a basic equation admits solutions and the complexity to list all of them, while Sections 5.2.2 and 5.2.3 point out the connection with the Change-Making problem and provide two algorithmic solutions. The solution based on MDD proves to be interesting from a memory and time point of view. Then, we conceived a pipeline to enumerate the solutions of general a-abstraction equations. It is based on the idea to avoid unfeasible systems (*i.e.* which contain impossible basic equations) and on the combination of the solutions (of the basic equations) performing Cartesian products and intersections directly on MDD. Hence, this allows us to control the number of operations to retrieve the set of values for each $\mathring{X}^w$ in the a-abstraction equation. The last but non-trivial step is the introduction of a technique to compute roots over the cyclic behaviours of DDS. Let us recall that, since we are in a commutative semiring, inverse operations do not exist. For this reason, being able to compute the roots over cyclic parts of DDS is a key point in the suitability of the pipeline to be able to solve polynomial equations with a degree greater than one.

Finally, the t-abstraction is presented. This abstraction is founded on the idea of representing the transient behaviours of a DDS with a multiset of matrices modelling the transient part of each component of a system. Transient dynamics is more precisely represented through multisets containing the number of predecessors of transient nodes with a certain distance from a certain periodic node. This last abstraction is certainly the one that proves most insidious when trying to understand how transients are involved in the product operation. However, by exploring the connection with the cancellation problem on graphs, we introduced a linear upper bound to the number of solutions of basic equations over t-abstractions, as well as over connected DDS. This result is particularly important because it allows us to limit the combinatorics when trying to find solutions to basic equations. Indeed, in Section 6.2.2 we provided a polynomial algorithm to list all solutions of a basic t-abstraction equation, and in Section 6.2.3 an exponential one to find all solutions (up to isomorphism) of basic equations over connected DDS.

Several questions arise at this point, as well as, prospects for future works.

Concerning the t-abstraction, it would be important to be able to prove Conjecture 1. From the experiments carried out, no counterexamples have been found unless there are symmetries between the solutions. Proving it would allow the algorithms presented in Chapter 6 to stop at the first solution found, hence limiting the calculations. Therefore, it would become interesting and useful to find conditions to establish which alignment is most likely to yield a solution.

Another important step is certainly to introduce a pipeline to combine the solutions of the basic equations to reconstruct the solutions of a t-abstraction equation. The general idea of the pipeline would be similar to the one showed for the a-abstraction, but with two extra features. It would have to take into account the solutions calculated for the basic equations over connected DDS instead of the t-abstraction ones, and provide an additional step for calculating the solutions of equations with a right-hand term containing several components with cycles of the same length (Equation (6.4)). This would allow us to solve more complex equations than simple basic ones. However, to solve equations of a degree greater than one, it is necessary to study the root computation on the transient part in accordance with what has been studied on the product. In other words, one would have to be able to solve

$$T^X + \ldots + T^X \supseteq T^B$$

and also, in this case, one would have to be able to solve it on t-abstractions, as well as, reconstruct the corresponding DDS. It is indeed important to emphasise that it has already been shown that, if the solution T^X exists, it is unique [Imrich et al. (2007)].

Another target, concerning the algorithms presented in Chapter 6, could be to introduce a parallel version. It is in fact possible to parallelise the computation between different alignments (for both the polynomial and the exponential algorithm). For the exponential version, also the computations with different affectations of origins can be parallelised. However, the trade-off between the "cost" of the treads and the computations they perform would have to be explored. In any case, parallelisation would allow the exponential to be pushed a little further and this could be particularly significant because it allows DDS with larger sizes to be solved (and this is very important from an application point of view).

To conclude on t-abstraction, it would be interesting to study the complexity of the problems involved. Given the algorithms presented, we know that deciding whether a solution on t-abstrations exists is in P, while on connected DDS, it is in NP. Nevertheless, it would be useful to explore further into the complexity of knowing whether an equation admits solutions before

enumerating which ones. The complexity of the enumeration problems can also be better investigated. Based on what we have seen in Chapter 6, we know that they are both in $EnumP$, as verification can be done in polynomial time, but they could also be part of other classes. Enumeration problems can be classified according to: the total time (depending on the size of the input and/or output) needed for the enumeration, the average time (total time divided by the number of solutions), the delay (time between two solutions), or even according to the time to find the first k solutions. For each of these criteria, there are different classes that have been already defined [Strozecki (2021)]. This analysis would also be interesting for the enumeration problems presented in Chapters 4 and 5.

Other future developments, now considering the a-abstraction, would be to study alternatives for the computation of the intersection of SB-Cartesian MDD and the introduction of parallelism in the pipeline. With regard to the intersection, one could imagine testing, for example, an approach based on an initial non-deterministic MDD that would allow to use the classical intersection, or to try to "reorganise" the result of a Cartesian product in such a way that it is an SB-MDD. On the other hand, as far as the pipeline is concerned, one could parallelise the computation of the solutions of the different basic equations and of the solutions of the different systems (*i.e. root-tt* paths in CS), and also the roots for the different values found for each $\mathring{X}^w$. Both of these improvements would be just implementative, but significant for performance.

Finally coming to consider the research directions opened up in Chapter 4, we come to the somewhat more general questions.

An important step would be to compare experimentally the approach based on the independent resolution of abstractions (followed by an intersection) and the guided search based on the resolution of abstractions in the order presented to better understand the gain.

Furthermore, one could consider multivariate polynomial equations with constant right term and several variables per monomial. One approach would be to consider each monomial as the result of multiplying the coefficient and the unknown part Y, and then solving $Y = X_1^{w_1} \dots X_k^{w_k}$ afterwards. This is possible on the basis of what has been learned about the product operation in the three abstractions, but needs to be formalised.

Another interesting direction aim is to investigate the possibility of introducing "equivalence classes" based on assumptions modelled in a polynomial equation. In other words, identify dynamical systems that respect a certain property represented in the polynomial side and can hence be defined as belonging to the same "class". Obviously, an important step would be to write the polynomial according to what one wants to study.

Evaluating the provided methodology in an application context would also be an important step to better understand the modelling phase of a hypothesis and the significance of the solutions found to the equation.

Finally, one could try to extend the abstractions-based solution to other types of dynamics (*i.e.* other than functional graphs). This would require reverifying the existence of an algebraic structure to be able to formulate polynomial equations and also review all abstractions from both a theoretical and an implementation point of view. This is certainly an ambitious goal, but it would allow us to apply the idea presented in this thesis to a much larger class of DDS, namely the non-deterministic finite DDS.

Bibliography

[Adamatzky et al. (2020)] Adamatzky, A., Goles, E., Martinez, G. J., Tsompanas, M.-A., Tegelaar, M., and Wosten, H. A. B. (2020). Fungal Automata.

[Ahmadi and Khadir (2020)] Ahmadi, A. A. and Khadir, B. E. (2020). Learning Dynamical Systems with Side Information.

[Akers (1978)] Akers, S. B. (1978). Binary decision diagrams. *IEEE Transactions on computers*, pages 509–516.

[Alonso-Sanz (2012)] Alonso-Sanz, R. (2012). Cellular automata and other discrete dynamical systems with memory. In Smari, W. W. and Zeljkovic, V., editors, *Proceedings of HPCS*, page 215. IEEE.

[Amilhastre et al. (2014)] Amilhastre, J., Fargier, H., Niveau, A., and Pralet, C. (2014). Compiling CSPs: A complexity map of (non-deterministic) multivalued decision diagrams. *International Journal on Artificial Intelligence Tools*, 23(04):1460015.

[Andersen (1997)] Andersen, H. R. (1997). An introduction to binary decision diagrams. *Lecture notes, available online, IT University of Copenhagen*, page 5.

[Antoulas (2004)] Antoulas, A. (2004). Approximation of large-scale dynamical systems: An overview. *IFAC Proceedings Volumes*, 37(11):19–28.

[Aracena et al. (2011)] Aracena, J., Fanchon, E., Montalva, M., and Noual, M. (2011). Combinatorics on update digraphs in Boolean networks. *Discrete Applied Mathematics*, 159(6):401–409.

[Aracena et al. (2009)] Aracena, J., Goles, E., Moreira, A., and Salinas, L. (2009). On the robustness of update schedules in Boolean networks. *Biosystems*, 97:1–8.

[Bergman et al. (2014)] Bergman, D., Cire, A. A., and van Hoeve, W. (2014). MDD propagation for sequence constraints. *Journal of Artificial Intelligence Research*, 50:697–722.

[Bergman et al. (2016)] Bergman, D., Cire, A. A., Van Hoeve, W.-J., and Hooker, J. (2016). *Decision diagrams for optimization*, volume 1. Springer.

[Berndt et al. (2012)] Berndt, R., Bazan, P., Hielscher, K.-S., German, R., and Lukasiewycz, M. (2012). Multi-valued decision diagrams for the verification of consistency in automotive product data. In *2012 12th International Conference on Quality Software*, pages 189–192. IEEE.

[Bournez et al. (2022)] Bournez, O., Chetrite, R., Fijalkow, Nathanaël and. Formenti, E., Kaufmann, E., Mattei, P.-A., Ollinger, N., Patras, F., Rifford, L., Romashchenko, A., and Villata, S. (2022). *Informatique Mathématique Une photographie en 2020*.

[Bower and Bolouri (2004)] Bower, J. M. and Bolouri, H. (2004). *Computational modeling of genetic and biochemical networks*. MIT press.

[Bryant (1986)] Bryant, R. E. (1986). Graph-Based Algorithms for Boolean Function Manipulation. *IEEE Trans. Computers*, 35(8):677–691.

[Calderoni et al. (2021)] Calderoni, L., Margara, L., and Marzolla, M. (2021). A Heuristic for Direct Product Graph Decomposition. *arXiv preprint arXiv:2107.03133*.

[Cattaneo et al. (1997)] Cattaneo, G., Formenti, E., Margara, L., and Mauri, G. (1997). Transformations of the One-Dimensional Cellular Automata Rule Space. *Parallel Comput.*, 23(11):1593–1611.

[Chaki and Gurfinkel (2018)] Chaki, S. and Gurfinkel, A. (2018). BDD-based symbolic model checking. In *Handbook of Model Checking*, pages 219–245. Springer.

[Cheng and Yap (2010)] Cheng, K. C. and Yap, R. H. (2010). An MDD-based generalized arc consistency algorithm for positive and negative table constraints and some global constraints. *Constraints*, 15(2):265–304.

[Clarke et al. (1994)] Clarke, E. M., Grumberg, O., and Long, D. E. (1994). Model checking and abstraction. *ACM transactions on Programming Languages and Systems (TOPLAS)*, 16(5):1512–1542.

[Cook (2004)] Cook, M. (2004). Universality in Elementary Cellular Automata. *Complex Systems*, 15(1):1–40.

[Coste and Henon (1986)] Coste, J. and Henon, M. (1986). Invariant Cycles in the Random Mapping of N Integers Onto Themselves. Comparison with Kauffman Binary Network. In Bienenstock, E., Soulié, F. F., and Weisbuch, G., editors, *Disordered Systems and Biological Organization*, pages 361–365, Berlin, Heidelberg. Springer Berlin Heidelberg.

[Cucker et al. (1999)] Cucker, F., Koiran, P., and Smale, S. (1999). A Polynomial Time Algorithm for Diophantine Equations in One Variable. *J. Symb. Comput.*, 27(1):21–29.

[Danca and Fečkan (2019)] Danca, M.-F. and Fečkan, M. (2019). Hidden chaotic attractors and chaos suppression in an impulsive discrete economical supply and demand dynamical system. *Communications in Nonlinear Science and Numerical Simulation*, 74:1–13.

[Darwiche and Marquis (2002)] Darwiche, A. and Marquis, P. (2002). A knowledge compilation map. *Journal of Artificial Intelligence Research*, 17:229–264.

[Datta et al. (2009)] Datta, S., Limaye, N., Nimbhorkar, P., Thierauf, T., and Wagner, F. (2009). Planar graph isomorphism is in log-space. In *2009 24th Annual IEEE Conference on Computational Complexity*, pages 203–214. IEEE.

[Demongeot and Sené (2020)] Demongeot, J. and Sené, S. (2020). About block-parallel Boolean networks: a position paper. *Nat. Comput.*, 19(1):5–13.

[Dennunzio et al. (2018)] Dennunzio, A., Dorigatti, V., Formenti, E., Manzoni, L., and Porreca, A. E. (2018). Polynomial Equations over Finite, Discrete-Time Dynamical Systems. In *Proc. of ACRI'18*, pages 298–306.

[Dennunzio et al. (2012)] Dennunzio, A., Formenti, E., Manzoni, L., and Mauri, G. (2012). m-Asynchronous cellular automata. In *International Conference on Cellular Automata*, pages 653–662. Springer.

[Dennunzio et al. (2013)] Dennunzio, A., Formenti, E., Manzoni, L., and Mauri, G. (2013). m-Asynchronous cellular automata: from fairness to quasi-fairness. *Natural Computing*, 12(4):561–572.

[Derrida and Flyvbjerg (1987)] Derrida, B. and Flyvbjerg, H. (1987). The random map model: a disordered model with deterministic dynamics. *Journal de Physique*, 48(6):971–978.

[Devaney (2018)] Devaney, R. (2018). *An introduction to chaotic dynamical systems*. CRC Press.

[Dorigatti (2018)] Dorigatti, V. (2018). Algorithms and Complexity of the Algebraic Analysis of Finite Discrete Dynamical Systems. Master's thesis, Università degli Studi di Milano-Bicocca.

[Drechsler et al. (2004)] Drechsler, R. et al. (2004). *Advanced formal verification*, volume 122. Springer.

[Dubrova and Teslenko (2011)] Dubrova, E. and Teslenko, M. (2011). A SAT-Based Algorithm for Finding Attractors in Synchronous Boolean Networks. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 8(5):1393–1399.

[Fatès (2014)] Fatès, N. (2014). A guided tour of asynchronous cellular automata. *Journal of Cellular Automata*, 9(5-6):387–416.

[Finkenstädt and Grenfell (2000)] Finkenstädt, B. F. and Grenfell, B. T. (2000). Time series modelling of childhood diseases: a dynamical systems approach. *Journal of the Royal Statistical Society: Series C (Applied Statistics)*, 49(2):187–205.

[Gardner (1970)] Gardner, M. (1970). Mathematical games: The fantastic combinations of John Conway's new solitaire game "Life". *Scientific American*, 223(4):120–123.

[Garg et al. (2008)] Garg, A., Di Cara, A., Xenarios, I., Mendoza, L., and De Micheli, G. (2008). Synchronous versus asynchronous modeling of gene regulatory networks. *Bioinformatics*, 24(17):1917–1925.

[Garg et al. (2007)] Garg, A., Xenarios, I., Mendoza, L., and DeMicheli, G. (2007). An Efficient Method for Dynamic Analysis of Gene Regulatory Networks and in silico Gene Perturbation Experiments. In Speed, T. and Huang, H., editors, *Research in Computational Molecular Biology*, pages 62–76, Berlin, Heidelberg. Springer Berlin Heidelberg.

[Gillard et al. (2021)] Gillard, X., Coppé, V., Schaus, P., and Cire, A. A. (2021). Improving the filtering of branch-and-bound MDD solver. In *International Conference on Integration of Constraint Programming, Artificial Intelligence, and Operations Research*, pages 231–247. Springer.

[Ginocchio (1998)] Ginocchio, M. (1998). On the Hopf algebra of functional graphs and differential algebras. *Discrete mathematics*, 183(1-3):119–140.

[Golan (2013)] Golan, J. S. (2013). *Semirings and affine equations over them: theory and applications*, volume 556. Springer Science & Business Media.

[Goles et al. (2018)] Goles, E., Montalva-Medel, M., Maclean, S., and Mortveit, H. S. (2018). Block Invariance in a Family of Elementary Cellular Automata. *J. Cellular Automata*, 13(1-2):15–32.

[Goles et al. (2015)] Goles, E., Montalva-Medel, M., Mortveit, H. S., and Ramírez-Flandes, S. (2015). Block Invariance in Elementary Cellular Automata. *J. Cellular Automata*, 10(1-2):119–135.

[Hammack and Toman (2010)] Hammack, R. and Toman, K. (2010). Cancellation of direct products of digraphs. *Discussiones Mathematicae Graph Theory*, 30(4):575–590.

[Hammack (2009)] Hammack, R. H. (2009). On direct product cancellation of graphs. *Discrete Mathematics*, 309(8):2538–2543.

[Hammack et al. (2011)] Hammack, R. H., Imrich, W., Klavžar, S., Imrich, W., and Klavžar, S. (2011). *Handbook of product graphs*, volume 2. CRC press Boca Raton.

[Harary and Trauth (1966)] Harary, F. and Trauth, Jr, C. A. (1966). Connectedness of products of two directed graphs. *SIAM Journal on Applied Mathematics*, 14(2):250–254.

[Hebisch and Weinert (1998)] Hebisch, U. and Weinert, H. J. (1998). *Semirings: algebraic theory and applications in computer science*, volume 5. World Scientific.

[Hong et al. (2013)] Hong, S., Rodia, N. C., and Olukotun, K. (2013). On Fast Parallel Detection of Strongly Connected Components (SCC) in Small-World Graphs. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, SC '13, New York, NY, USA. Association for Computing Machinery.

[Hopcroft and Wong (1974)] Hopcroft, J. E. and Wong, J.-K. (1974). Linear time algorithm for isomorphism of planar graphs (preliminary report). In *Proceedings of the sixth annual ACM symposium on Theory of computing*, pages 172–184.

[Imrich et al. (2007)] Imrich, W., Klavžar, S., and Rall, D. F. (2007). Cancellation properties of products of graphs. *Discrete applied mathematics*, 155(17):2362–2364.

[Jakubczyk and Sontag (1990)] Jakubczyk, B. and Sontag, E. D. (1990). Controllability of Nonlinear Discrete-Time Systems: A Lie-Algebraic Approach. *SIAM Journal on Control and Optimization*, 28(1):1–33.

[Jongsma (2019)] Jongsma, C. (2019). Basic Set Theory and Combinatorics. In *Introduction to Discrete Mathematics via Logic and Proof*, pages 205–253. Springer.

[Jung and Régin (2022)] Jung, V. and Régin, J.-C. (2022). Efficient operations between mdds and constraints. In *International Conference on Integration of Constraint Programming, Artificial Intelligence, and Operations Research*, pages 173–189. Springer.

[Kari (2005)] Kari, J. (2005). Theory of cellular automata: A survey. *Theor. Comput. Sci.*, 334(1-3):3–33.

[Kartal et al. (2016)] Kartal, S., Kar, M., Kartal, N., and Gurcan, F. (2016). Modelling and analysis of a phytoplankton–zooplankton system with continuous and discrete time. *Mathematical and Computer Modelling of Dynamical Systems*, 22(6):539–554.

[Katz (1955)] Katz, L. (1955). Probability of indecomposability of a random mapping function. *The Annals of Mathematical Statistics*, pages 512–517.

[Kauffman (1969)] Kauffman, S. A. (1969). Metabolic stability and epigenesis in randomly constructed genetic nets. *Journal of theoretical biology*, 22(3):437–467.

[Knoblock (1990)] Knoblock, C. A. (1990). Learning Abstraction Hierarchies for Problem Solving. In *AAAI*, pages 923–928.

[Knuth (2011)] Knuth, D. E. (2011). *The art of computer programming, volume 4A: combinatorial algorithms, part 1*. Pearson Education India.

[Kruskal (1954)] Kruskal, M. D. (1954). The expected number of components under a random mapping function. *The American Mathematical Monthly*, 61(6):392–397.

[Lamprey and Barnes (1974)] Lamprey, R. and Barnes, B. (1974). Product graphs and their applications. *Modelling and Simulation*, 5:1119–1123.

[Laskar and Gastineau (2009)] Laskar, J. and Gastineau, M. (2009). Existence of collisional trajectories of Mercury, Mars and Venus with the Earth. *Nature*, 459(7248):817–819.

[Lawvere and Schanuel (1997)] Lawvere, F. and Schanuel, S. (1997). First introduction to categories.

[Lee (1959)] Lee, C.-Y. (1959). Representation of switching circuits by binary-decision programs. *The Bell System Technical Journal*, 38(4):985–999.

[Lovász (1971)] Lovász, L. (1971). On the cancellation law among finite relational structures. *Periodica Mathematica Hungarica*, 1(2):145–156.

[Ma (2009)] Ma, Z. (2009). *Dynamical modeling and analysis of epidemics*. World Scientific.

[Macauley et al. (2007)] Macauley, M., McCammond, J., and Mortveit, H. S. (2007). Order independence in asynchronous cellular automata. *arXiv preprint arXiv:0707.2360*.

[Macauley et al. (2011)] Macauley, M., McCammond, J., and Mortveit, H. S. (2011). Dynamics groups of asynchronous cellular automata. *Journal of Algebraic Combinatorics*, 33(1):11–35.

[Martello and Toth (1990)] Martello, S. and Toth, P. (1990). *Knapsack Problems: Algorithms and Computer Implementations*. John Wiley & Sons, Inc., New York, NY, USA.

[Mary and Strozecki (2019)] Mary, A. and Strozecki, Y. (2019). Efficient enumeration of solutions produced by closure operations. *Discrete Mathematics & Theoretical Computer Science*, 21(3).

[Matijaszevic (1970)] Matijaszevic, J. V. (1970). Solution of the tenth problem of Hilbert. *Mat. Lapok*, 21:83–87.

[Matiyasevich (1996)] Matiyasevich, Y. (1996). Hilbert's tenth problem: What can we do with diophantine equations? In *Proceedings of a seminar of Institut de Recherche sur l'Enseignement des Mathematique*.

[Matiyasevich (1993)] Matiyasevich, Y. V. (1993). Hilbert's tenth problem. Foundations of Computing Series.

[McAndrew (1963)] McAndrew, M. (1963). On the product of directed graphs. *Proceedings of the American Mathematical Society*, 14(4):600–606.

[McKenzie (1971)] McKenzie, R. (1971). Cardinal multiplication of structures with a reflexive relation. *Fundamenta Mathematicae*, 70(1):59–101.

[Metropolis and Ulam (1952)] Metropolis, N. and Ulam, S. (1952). A property of randomness of an arithmetical function. Technical report, Los Alamos Scientific Lab.

[Minato (2007)] Minato, S.-i. (2007). A Theoretical Study on Variable Ordering of ZBDDs for Representing Frequent Itemsets.

[Nakahara et al. (2017)] Nakahara, H., Jinguji, A., Sato, S., and Sasao, T. (2017). A random forest using a multi-valued decision diagram on an FPGA. In *2017 IEEE 47th International Symposium on Multiple-Valued Logic (ISMVL)*, pages 266–271. IEEE.

[Naldi et al. (2007)] Naldi, A., Thieffry, D., and Chaouiya, C. (2007). Decision diagrams for the representation and analysis of logical models of genetic networks. In *International Conference on Computational Methods in Systems Biology*, pages 233–247. Springer.

[Okninski and Radziszewski (2010)] Okninski, A. and Radziszewski, B. (2010). Simple models of bouncing ball dynamics and their comparison. *arXiv preprint arXiv:1006.1236*.

[Paulevé (2020)] Paulevé, L. (2020). *Réseaux booléens : méthodes formelles et outils pour la modélisation en biologie*. Habilitation à diriger des recherches, Université Paris-Saclay.

[Pemmaraju et al. (2003)] Pemmaraju, S., Skiena, S., et al. (2003). *Computational discrete mathematics: Combinatorics and graph theory with mathematica®*. Cambridge university press.

[Perez (2017)] Perez, G. (2017). *Diagrammes de décision: contraintes et algorithmes*. PhD thesis, Université Côte d'Azur (ComUE).

[Perez and Régin (2015)] Perez, G. and Régin, J.-C. (2015). Efficient operations on MDDs for building constraint programming models. In *IJCAI 2015*.

[Perrot et al. (2019)] Perrot, K., Montalva-Medel, M., de Oliveira, P. P. B., and Ruivo, E. L. P. (2019). Maximum sensitivity to update schedule of elementary cellular automata over periodic configurations. *Natural Computing*.

[Poincaré (1892)] Poincaré, H. (1892). Les méthodes nouvelles de la méchanique céleste vol. 1 (paris).

[Poincaré (1893)] Poincaré, H. (1893). *Les méthodes nouvelles de la mécanique céleste: Méthodes de MM. Newcomb, Gyldén, Linstadt et Bohlin*, volume 2. Gauthier-Villars et fils, imprimeurs-libraires.

[Poincaré (1899)] Poincaré, H. (1899). *Les méthodes nouvelles de la mécanique céleste*, volume 3. Gauthier-Villars et fils.

[Romero and Zertuche (2003)] Romero, D. and Zertuche, F. (2003). The asymptotic number of attractors in the random map model. *Journal of Physics A: Mathematical and General*, 36(13):3691.

[Romero and Zertuche (2005)] Romero, D. and Zertuche, F. (2005). Grasping the connectivity of random functional graphs. *Studia Scientiarum Mathematicarum Hungarica*, 42(1):1–19.

[Ruivo et al. (2020)] Ruivo, E. L. P., de Oliveira, P. P. B., Montalva-Medel, M., and Perrot, K. (2020). Maximum sensitivity to update schedules of elementary cellular automata over infinite configurations. *Information and Computation*, 274:104538.

[Ruivo et al. (2018)] Ruivo, E. L. P., Montalva-Medel, M., de Oliveira, P. P. B., and Perrot, K. (2018). Characterisation of the elementary cellular automata in terms of their maximum sensitivity to all possible asynchronous updates. *Chaos, Solitons & Fractals*, 113:209–220.

[Sandefur (1993)] Sandefur, J. T. (1993). *Discrete dynamical modeling*. Oxford University Press on Demand.

[Schwab et al. (2020)a] Schwab, J. D., Kühlwein, S. D., Ikonomi, N., Kühl, M., and Kestler, H. A. (2020a). Concepts in Boolean network modeling: What do they all mean? *Computational and structural biotechnology journal*, 18:571–582.

[Schwab et al. (2020)b] Schwab, J. D., Kühlwein, S. D., Ikonomi, N., Kühl, M., and Kestler, H. A. (2020b). Concepts in Boolean network modeling: What do they all mean? *Computational and Structural Biotechnology Journal*, 18:571–582.

[Sené (2012)] Sené, S. (2012). *On the bioinformatics of automata networks*. HDR, University of Évry Val d'Essonne, France.

[Sené (2012)] Sené, S. (2012). *Sur la bio-informatique des réseaux d'automates*. Habilitation à diriger des recherches, Université d'Evry-Val d'Essonne.

[Strozecki (2021)] Strozecki, Y. (2021). *Enumeration Complexity: Incremental Time, Delay and Space*. PhD thesis, Université de Versailles Saint-Quentin-en-Yvelines.

[The Online Encyclopedia of Integer Sequences (1996)a] The Online Encyclopedia of Integer Sequences (1996a). OEIS A001372. Technical report. https://oeis.org/A001372.

[The Online Encyclopedia of Integer Sequences (1996)b] The Online Encyclopedia of Integer Sequences (1996b). OEIS A005248. Technical report. https://oeis.org/A005248.

[Thomas (1973)] Thomas, R. (1973). Boolean formalization of genetic control circuits. *Journal of theoretical biology*, 42(3):563–585.

[Wang et al. (2021)] Wang, R., Maddix, D., Faloutsos, C., Wang, Y., and Yu, R. (2021). Bridging physics-based and data-driven modeling for learning dynamical systems. In *Learning for Dynamics and Control*, pages 385–398. PMLR.

[Weichsel (1962)] Weichsel, P. M. (1962). The Kronecker product of graphs. *Proceedings of the American mathematical society*, 13(1):47–52.

[Wille and Drechsler (2009)] Wille, R. and Drechsler, R. (2009). BDD-based synthesis of reversible logic for large functions. In *Proceedings of the 46th Annual Design Automation Conference*, pages 270–275.

[Wright (1975)] Wright, J. (1975). The change-making problem. *Journal of the ACM (JACM)*, 22(1):125–128.

[Zaitseva et al. (2013)] Zaitseva, E., Levashenko, V., Kostolny, J., and Kvassay, M. (2013). A multi-valued decision diagram for estimation of multi-state system. In *Eurocon 2013*, pages 645–650. IEEE.

[Zhang et al. (2019)] Zhang, L., Xing, L., Liu, A., and Mao, K. (2019). Multivalued Decision Diagrams-Based Trust Level Analysis for Social Networks. *IEEE Access*, 7:180620–180629.

[Zheng et al. (2013)] Zheng, D., Yang, G., Li, X., Wang, Z., Liu, F., and He, L. (2013). An efficient algorithm for computing attractors of synchronous and asynchronous Boolean networks. *PloS one*, 8(4):e60593.

Web Sources

[1] Dynamical systems and their algebra. https://aeporreca.org/talks/dynamical-systems-and-their-algebra.pdf.

[2] Séminaire cana : Antonio e. porreca, the semiring of dynamical systems. https://amupod.univ-amu.fr/video/7228-seminaire-cana-antonio-e-porreca-the-semiring-of-dynamical-systems/.

Publications

[Balbi et al. (2022)] Balbi, P. P., Formenti, E., Perrot, K., Riva, S., and Ruivo, E. L. (2022). Non-maximal sensitivity to synchronism in elementary cellular automata: Exact asymptotic measures. *Theoretical Computer Science*, 926:21–50.

[Balbi et al. (2020)] Balbi, P. P., Formenti, E., Perrot, K., Riva, S., and Ruivo, E. L. P. (2020). Non-maximal Sensitivity to Synchronism in Periodic Elementary Cellular Automata: Exact Asymptotic Measures. In Zenil, H., editor, *Cellular Automata and Discrete Complex Systems*, pages 14–28, Cham. Springer International Publishing.

[Dennunzio et al. (2020)] Dennunzio, A., Formenti, E., Margara, L., Montmirail, V., and Riva, S. (2020). Solving Equations on Discrete Dynamical Systems. In Cazzaniga, P., Besozzi, D., Merelli, I., and Manzoni, L., editors, *Computational Intelligence Methods for Bioinformatics and Biostatistics*, pages 119–132, Cham. Springer International Publishing.

[Dennunzio et al. (2022)] Dennunzio, A., Formenti, E., Margara, L., and Riva, S. (2022). An Algorithmic Pipeline for Solving Equations over Discrete Dynamical Systems Modelling Hypothesis on Real Phenomena. *arXiv preprint arXiv:2211.05038*. Accepted for publication by Journal of Computational Science.

[Doré et al. (2022)] Doré, F., Formenti, E., Porreca, A. E., and Riva, S. (2022). Algorithmic reconstruction of discrete dynamics. *arXiv preprint arXiv:2208.08310*. submitted to Theoretical Computer Science.

[Formenti et al. (2021)] Formenti, E., Régin, J.-C., and Riva, S. (2021). MDDs Boost Equation Solving on Discrete Dynamical Systems. In *International Conference on Integration of Constraint Programming, Artificial Intelligence, and Operations Research*, pages 196–213. Springer.

List of Figures

List of Examples

List of Algorithms

Appendix

A Non-maximal sensitivity to synchronism in ECA: exact asymptotic measures

This appendix presents the results of research activities carried out during the thesis period concerning Elementary Cellular Automata and their sensitivity to synchronism. The results presented here are the outcome of a collaboration with Kévin Perrot, Dr. HDR (Université Aix-Marseille, France), Professor Pedro Paulo Balbi de Oliveira, Dr. (Universidade Presbiteriana Mackenzie, São Paulo, Brazil), Eurico Ruivo, Dr. (Universidade Presbiteriana Mackenzie, São Paulo, Brazil), and Professor Enrico Formenti, Dr. HDR (Université Côte d'Azur, France) and they have been published in [Balbi et al. (2020), Balbi et al. (2022)].

As presented in Chapter 1, Cellular automata (CA) are dynamical systems and interesting mathematical and computational objects suitable for modelling real-world complex systems. The dynamics of a CA is locally-defined: every cell computes its next state based upon its own state and neighbours states (according to a specific definition of neighbourhood). Here, we deal with deterministic *block-sequential* updates [Aracena et al. (2009), Aracena et al. (2011)], as presented in Section 1.3.2. Recall that, under these updating schemes, the cells of a CA are partitioned into disjoint blocks with different priorities of being updated. The initial updating order (priority scheme) is unchanged throughout the temporal evolution. A possible cells partition is called a block-sequential update schedule. For a complete presentation of asynchronous CA, the reader can refer to Nazim Fatès analysis [Fatès (2014)].

In the context of asynchronous CA, a natural question concerns the number of different possible dynamics.

In the literature, several analyses are proposed to answer this question. McAuley et al. study asynchronous CA focusing on periodic points of the dynamics [Macauley et al. (2011), Macauley et al. (2007)]. According to them, a rule is π-independent if the set of periodic points is independent of the asynchronous update procedure (permutations of cells). They characterised 104 elementary cellular automata rules (ECA) as being π-independent.

Remark that one can extend this analysis also to non-periodic behaviour. Indeed, we fill this gap by considering two dynamics as non-equivalent if there is at least a configuration with a cell that can be updated differently (according to different update procedures).

Considering this definition, a possible way to study the number of different dynamics is through the notion of *sensitivity to synchronism* [Perrot et al. (2019), Ruivo et al. (2018)]. In this case, the idea is to investigate the number of dynamics over the number of valid block-sequential update schedules. A CA rule has been defined as *max-sensitive to synchronism* when each different block-sequential update schedule corresponds to a different dynamics.

Concerning the 256 ECA, 200 present maximum sensitivity [Ruivo et al. (2020)]. Therefore, it is natural to investigate the degree of sensitivity for the remaining 56 rules ($0, 3, 8, 12, 15, 28, 32, 34, 44, 51, 60, 128, 136, 140, 160, 162, 170, 200$ and 204).

We prove that these rules form four classes (insensitive, low-sensitive, medium-sensitive, and almost max-sensitive) according to their sensitivity to synchronism. We also characterise their sensitivity function *w.r.t.* periodic and bi-infinite configurations.

The results exhibit an interesting range of sensitivity functions (see Table 2). We show that some rules can have a number of dynamics that can be constant, exponential with respect to the number of cells (with base 2), equal to the number of valid update schedules minus a function of the number of cells, or equal to an exponential function with the golden ratio as a base.

This appendix is organised as follows. Section A.1 presents fundamental definitions and results on Boolean networks, update digraphs, and elementary cellular automata. Considering configurations of arbitrary size, Section A.2 provides formal expressions to the sensitivity to synchronism of the four classes. Concluding remarks are drawn in Section A.3.

A.1 Definitions

Finite elementary cellular automata will be presented in the more general framework of Boolean automata networks, for which the variation of update schedule benefits from useful considerations already studied in the literature. Figure A.13 illustrates the definitions.

A.1.1 Boolean networks

A Boolean Network (BN) of size n is an arrangement of n finite Boolean automata (or components) interacting with each other according to a *global rule* $F\colon \{0,1\}^n \to \{0,1\}^n$, which describes how the global state changes after one time step. Let $[\![n]\!] = \{0,\dots,n-1\}$. Each automaton is identified with a unique integer $i \in [\![n]\!]$ and x_i denotes the current state of the automaton i. A *configuration* $x \in \{0,1\}^n$ is a snapshot of the current state of all automata and represents the global state of the BN.

For convenience, we identify configurations with words on $\{0,1\}^n$. Hence, for example, 01111 or 01^4 both denote the configuration $(0,1,1,1,1)$. Remark that the global function $F : \{0,1\}^n \to \{0,1\}^n$ of a BN of size n induces a set of n *local functions* $f_i : \{0,1\}^n \to \{0,1\}$, one per each component, such that $F(x) = (f_0(x), f_1(x), \dots, f_{n-1}(x))$ for all $x \in \{0,1\}^n$. This gives a static description of a discrete dynamical system, and it remains to set the order in which components are updated in order to get a dynamics. Before going to update schedules, let us first introduce interaction digraphs.

The component i *influences* the component j if $\exists x \in \{0,1\}^n : f_j(x) \neq f_j(\overline{x}^i)$, where $\overline{x}^i$ is the configuration obtained from x by flipping the state of component i. The component i *influences* the component j if $\exists x \in \{0,1\}^n$ such that the result of the update procedure of j is different according to whether the state of component i is 0 or 1. Note that in the literature one may also consider *positive* and *negative* influences, but they will not be useful for the present study. The *interaction digraph* $G_F = (V, A)$ of a BN F represents the effective dependencies among its set of components

$$V = [\![n]\!] \quad \text{and} \quad A = \{(i,j) \mid i \text{ influences } j\}\,.$$

It will turn out to be pertinent to consider $\hat{G_F} = (V, A)$, obtained from G_F by removing the loops (arcs of the form (i,i)).

For $n \in \mathbb{N}$, one can denote by P_n the set of ordered partitions of $[\![n]\!]$ and by F a BN of size n. A *block-sequential update schedule* $\Delta = (\Delta_1, \dots, \Delta_k)$ is an element of P_n. It defines the

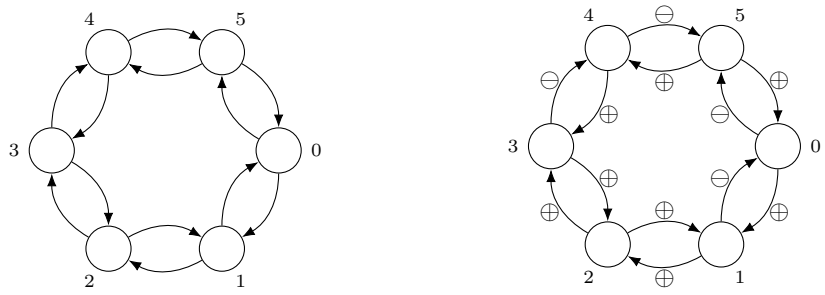


Figure A.13 – Left: interaction digraph G_6^{ECA} of the ECA rule 128 for $n = 6$, with local functions $f_i(x) = x_{i-1} \wedge x_i \wedge x_{i+1}$ for all $i \in \{0, \dots, 5\}$. Right: update digraph corresponding to the update schedules $\Delta = (\{1, 2, 3\}, \{0, 4\}, \{5\})$ and $\Delta' = (\{1, 2, 3\}, \{0\}, \{4\}, \{5\})$, which are therefore equivalent ($\Delta \equiv \Delta'$). For example, $f^{(\Delta)}(\mathtt{111011}) = \mathtt{110000}$ whereas for the synchronous update schedule we have $f^{(\Delta^{\mathrm{sync}})}(\mathtt{111011}) = \mathtt{110001}$.

following dynamics $F^{(\Delta)} : \{0,1\}^n \to \{0,1\}^n$,

$$F^{(\Delta)} = F^{(\Delta_k)} \circ \cdots \circ F^{(\Delta_2)} \circ F^{(\Delta_1)} \quad \text{with} \quad f^{(\Delta_j)}(x)_i = \begin{cases} f_i(x) & \text{if } i \in \Delta_j, \\ x_i & \text{if } i \notin \Delta_j. \end{cases}$$

In words, the components are updated in the order given by Δ: sequentially block by block, and in parallel within each block. The *parallel* or *synchronous* update schedule is $\Delta^{\mathrm{sync}} = ([\![n]\!])$. In this article, since only block-sequential update schedules are considered, they are simply called *update schedule* for short. They are

- "*fair*" in the sense that all components are updated exactly the same number of times,
- "*periodic*" in the sense that the same ordered partition is repeated.

Given a BN F of size n and an update schedule Δ, the *transition digraph* $D_{F^{(\Delta)}} = (V, A)$ is such that

$$V = \{0,1\}^n \quad \text{and} \quad A = \{(x, F^{(\Delta)}(x)) \mid x \in \{0,1\}^n\}.$$

It describes the *dynamics* of F under the update schedule Δ. The set of all possible dynamics of the BN F, at the basis of the measure of sensitivity to synchronism, is then defined as

$$\mathcal{D}(F) = \{D_{F^{(\Delta)}} \mid \Delta \in P_n\}.$$

A.1.2 Update digraphs and equivalent update schedules

For a given BN, some update schedules always give the same dynamics. Indeed, if, for example, two components do not influence each other, their order of updating has no effect on the dynamics (see Example A.13 for a detailed example). The notion of update digraph has been introduced in a previous work in order to study update schedules [Aracena et al. (2009)].

Given a BN F with loopless interaction digraph $\hat{G_F} = (V, A)$ and an update schedule $\Delta \in P_n$, define $lab_\Delta : A \to \{\oplus, \ominus\}$ as

$$\forall (i,j) \in A, \quad lab_\Delta((i,j)) = \begin{cases} \oplus & \text{if } i \in \Delta_a, j \in \Delta_b \text{ with } 1 \le b \le a \le n, \\ \ominus & \text{if } i \in \Delta_a, j \in \Delta_b \text{ with } 1 \le a < b \le n. \end{cases}$$

In other words, a $\ominus$-label on the arc (i, j) means that the cell i will be updated before the cell j. Similarly, a $\oplus$-label means that the cell i will be updated at the same time or after the cell j.

The *update digraph* $U_{F^{(\Delta)}}$ of the BN F for the update schedule $\Delta \in P_n$ is the loopless interaction digraph decorated with lab_Δ, *i.e.* $U_{F^{(\Delta)}} = (V, A, lab_\Delta)$. Note that loops are removed because they bring no meaningful information: indeed, an edge (i, i) would always be labeled $\oplus$. Now we have that if two update schedules define the same update digraph then they also define the same dynamics.

Theorem A.1 ([Aracena et al. (2009)]). *Given a BN F and two update schedules Δ, Δ', if $lab_\Delta = lab_{\Delta'}$ then $D_{F^{(\Delta)}} = D_{F^{(\Delta')}}$.*

A very important remark is that not all labelings correspond to *valid* update digraphs (*i.e.* such that there are update schedules giving these labelings). For example, if two arcs (i, j) and (j, i) belong to the interaction digraph and are both labeled $\ominus$, it would mean that i is updated prior to j and j is updated prior to i, which is contradictory. Fortunately there is a nice characterisation of *valid* update digraphs.

Theorem A.2 ([Aracena et al. (2011)]). *Given F with $\hat{G_F} = (V, A)$, the label function $lab : A \to \{\oplus, \ominus\}$ is valid if and only if there is no cycle $(i_0, i_1, \ldots, i_k)$, with $i_0 = i_k$ and $k > 0$, such that*

- $\forall 0 \leq j < k : \Big((i_j, i_{j+1}) \in A \text{ and } lab((i_j, i_{j+1})) = \oplus\Big)$ *or* $\Big((i_{j+1}, i_j) \in A \text{ and } lab((i_{j+1}, i_j)) = \ominus\Big)$,
- $\exists 0 \leq i < k : lab((i_{j+1}, i_j)) = \ominus$.

In words, Theorem A.2 states that a labeling is valid if and only if the multi-digraph where the labeling is unchanged but the orientation of $\ominus$-arcs is reversed does not contain a cycle with at least one arc labeled $\ominus$ (*forbidden cycle*).

According to Theorem A.1, update digraphs define equivalence classes of update schedules: $\Delta \equiv \Delta'$ if and only if $lab_\Delta = lab_{\Delta'}$. Given a BN F, the set of equivalence classes of update schedules is therefore defined as

$$\mathcal{U}(F) = \{U_{F^{(\Delta)}} \mid \Delta \in P_n\}.$$

A.1.3 Sensitivity to synchronism

The sensitivity to synchronism $\mu_s(F)$ of a BN F quantifies the proportion of distinct dynamics with respect to non-equivalent update schedules. The idea is that when two or more update schedules are equivalent then $\mu_s(F)$ decreases, while it increases when distinct update schedules bring to different dynamics. More formally, given a BN F we define

$$\mu_s(F) = \frac{|\mathcal{D}(F)|}{|\mathcal{U}(F)|}.$$

Obviously, it holds that $\frac{1}{|\mathcal{U}(F)|} \leq \mu_s(F) \leq 1$, and a BN F is as much sensible to synchronism as it has different dynamics when the update schedule varies. The extreme cases are a BN F with $\mu_s(F) = \frac{1}{|\mathcal{U}(F)|}$ that has always the same dynamics $D_{F^{(\Delta)}}$ for any update schedule Δ, and a BN F with $\mu_s(F) = 1$, which has a different dynamics for different update schedules (for each $\Delta \not\equiv \Delta'$

0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 18, 19, 22, 23, 24, 25, 26, 27, 28, 29, 30, 32, 33, 34, 35, 36, 37, 38, 40, 41, 42, 43, 44, 45, 46, 50, 51, 54, 56, 57, 58, 60, 62, 72, 73, 74, 76, 77, 78, 90, 94, 104, 105, 106, 108, 110, 122, 126, 128, 130, 132, 134, 136, 138, 140, 142, 146, 150, 152, 154, 156, 160, 162, 164, 168, 170, 172, 178, 184, 200, 204, 232

Table 1 – ECA local rules up to τ_ι, τ_ϱ, τ_η and $\tau_{\varrho\eta}$.

it holds that $D_{F(\Delta)} \neq D_{F(\Delta')}$). A BN F is *max-sensitive* to synchronism iff $\mu_s(F) = 1$. Note that a BN F is max-sensitive if and only if

$$\forall \Delta, \Delta' \in P_n,\ (\Delta \not\equiv \Delta') \Rightarrow \exists x \in \{0,1\}^n \text{ and } \exists i \in [\![n]\!] \text{ such that } f^{(\Delta)}(x)_i \neq f^{(\Delta')}(x)_i \ . \tag{A.9}$$

A.1.4 Elementary cellular automata

In this study we investigate the sensitivity to synchronism of *elementary cellular automata* (ECA) over periodic configurations. Indeed, they are a subclass of BN in which all components (also called *cells* in this context) have the same local rule, as follows. Given a size n, the ECA of local function $r : \{0,1\}^3 \to \{0,1\}$ is the BN F such that

$$\forall i \in [\![n]\!],\quad f_i(x) = r(x_{i-1}, x_i, x_{i+1})$$

where components are taken modulo n (this will be the case throughout all the appendix without explicit mention). We use the convention introduced by Wolfram to designate each of the 256 ECA local rule $r : \{0,1\}^3 \to \{0,1\}$ as the number

$$w(r) = \sum_{(x_1,x_2,x_3)\in\{0,1\}^3} r(x_1,x_2,x_3)2^{(2^2x_1+2^1x_2+2^0x_3)}.$$

We denote by r_α the Boolean function such that $w(r) = \alpha$ with $\alpha \in \{0,\dots,255\}$. Given a Boolean function $r : \{0,1\}^3 \to \{0,1\}$, consider the following transformations over local rules: $\tau_\iota(r)(x,y,z) = r(x,y,z)$, $\tau_\varrho(r)(x,y,z) = r(z,y,x)$, $\tau_\eta(r)(x,y,z) = 1 - r(1-z,1-y,1-x)$ and $\tau_{\varrho\eta}(r)(x,y,z) = 1 - r(1-z,1-y,1-x)$ for all $x,y,z \in \{0,1\}$. It is known that these transformations preserve the dynamics [Cattaneo et al. (1997)]. It is not difficult to see that they also preserve sensitivity to synchronism. For this reason we consider only 88 ECA rules up to equivalences with τ_ι, τ_ϱ, τ_η and $\tau_{\varrho\eta}$. Table 1 reports these equivalence classes of ECA, the smallest Wolfram number per class is indicated.

The definitions of Subsection A.1.3 are applied to ECA rules as follows. Given a size n, the *ECA interaction digraph of size* n $G_n^{\text{ECA}} = (V,A)$ is such that $V = [\![n]\!]$ and $A = \{(i+1,i),(i,i+1) \mid i \in [\![n]\!]\}$.

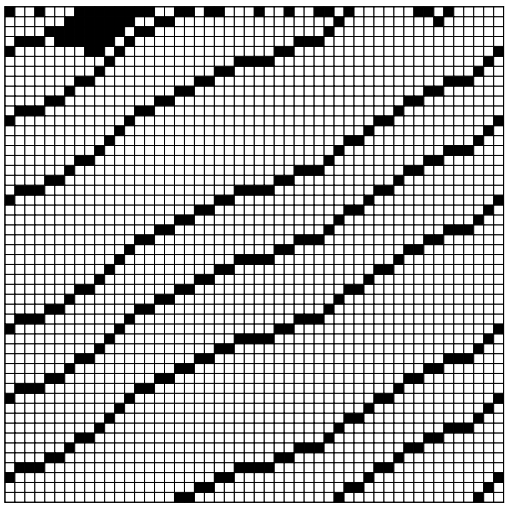
In [Perrot et al. (2019), Ruivo et al. (2018)], it is proved that

$$|\mathcal{U}^{\text{ECA}}(n)| = 3^n - 2^{n+1} + 2.$$

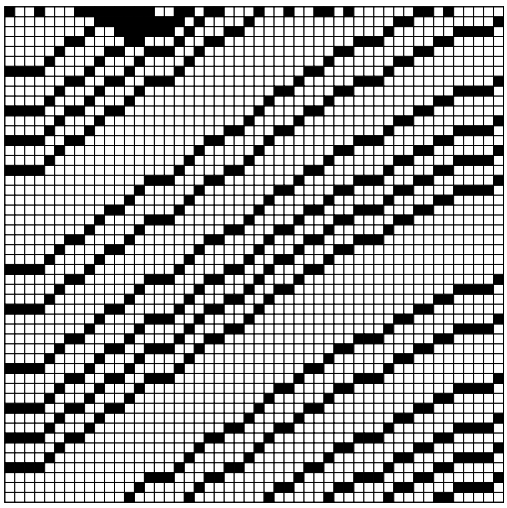
where $\mathcal{U}^{\text{ECA}}(n)$ is the set of valid labelings of G_n^{ECA}. The sensitivity to synchronism of ECA is measured relatively to the family of ECA, and therefore relatively to this count of valid labelings of G_n^{ECA}, even for rules where some arcs do not correspond to effective influences (one may think

of rule 0). Except from this subtlety, the measure is correctly defined by considering, for an ECA rule number α and a size n, that $r_\alpha \colon \{0,1\}^3 \to \{0,1\}$ is its local rule, and that $F_{\alpha,n} \colon \{0,1\}^n \to \{0,1\}^n$ is its global function on periodic configurations of size n, which is defined as follows

$$\forall x \in \{0,1\}^n, \forall i \in [\![n]\!], \quad f_\alpha(x)_i = r_\alpha(x_{i-1}, x_i, x_{i+1}).$$

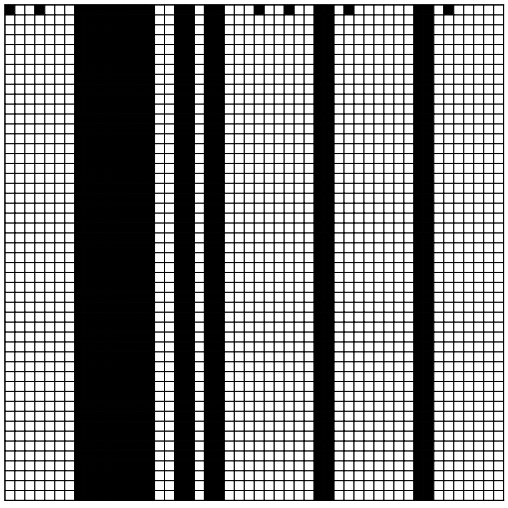


(a) $\Delta = \{(34, 30, 6, 46), (26, 5), (17, 13, 49, 48, 21, 29, 19, 44), (1, 37, 20, 23, 47, 36), (31, 28, 25), (24, 22, 42, 7, 11, 45, 14, 27, 15, 2, 40, 12), (35, 33, 43, 18, 32), (16, 39, 38, 41, 3, 10, 4, 8, 9, 0)\}$.

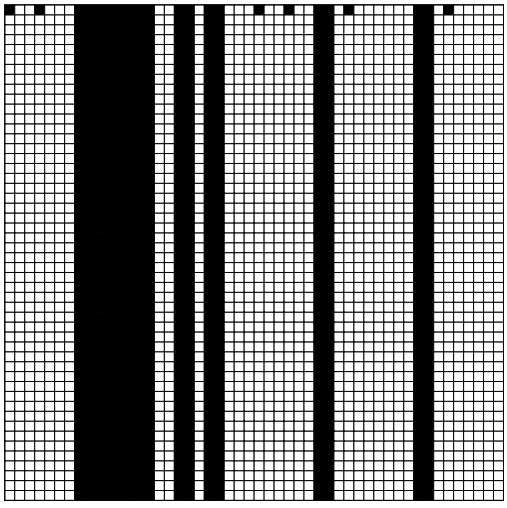


(b) $\Delta' = \{(1, 6, 25), (3, 17, 36, 31, 47, 7, 4, 40, 38), (15, 48, 23, 24, 29, 42), (19, 18, 13, 45, 49, 20, 43, 5), (12, 34, 32, 46, 2, 11, 9, 21, 8, 30, 44), (16, 37, 28), (27, 14, 26, 22, 33, 0, 10, 39, 35, 41)\}$.

Figure A.14 – Space-time diagrams for ECA rule 162 *w.r.t.* two non-equivalent update schedules Δ and Δ' (starting from the same configuration). We see that Δ and Δ' lead to different dynamics. Indeed, ECA rule 162 is sensitive (see Section A.2.4).



(a) Initial configuration x updated according to the same Δ as in Figure A.14.



(b) Initial configuration x updated according to the same Δ' as in Figure A.14.

Figure A.15 – Space-time diagrams for ECA rule 200 *w.r.t.* two non-equivalent update schedules Δ and Δ' (starting from the same configuration). We see that Δ and Δ' lead to the same dynamics. Indeed, ECA rule 200 is insensitive to synchronism (see Section A.2.1).

Then, the sensitivity to synchronism of ECA rule number α is given by

$$\mu_s(F_{\alpha,n}) = \frac{|\mathcal{D}(F_{\alpha,n})|}{3^n - 2^{n+1} + 2}.$$

Figures A.14 and A.15 show examples of a ECA rule that is sensitive to synchronism and one that is not.
An ECA rule number α is ultimately *max-sensitive to synchronism* when

$$\lim_{n\to+\infty} \mu_s(F_{\alpha,n}) = 1.$$

The following result provides a first overview of sensitivity to synchronism in ECA.

Theorem A.3 ([Perrot et al. (2019), Ruivo et al. (2018)]). *For any size $n \geq 7$, the nineteen ECA rules 0, 3, 8, 12, 15, 28, 32, 34, 44, 51, 60, 128, 136, 140, 160, 162, 170, 200 and 204 are not max-sensitive to synchronism. The remaining sixty-nine other rules are max-sensitive to synchronism.*

Theorem A.3 gives a precise measure of sensitivity for the sixty-nine maximum sensitive rules, for which $\mu_s(F_{\alpha,n}) = 1$ for all $n \geq 7$, but for the nineteen that are not maximum sensitive it only informs that $\mu_s(F_{\alpha,n}) < 1$ for all $n \geq 7$. In the rest of this appendix, we study the precise dependency on n of $\mu_s(F_{\alpha,n})$ for these rules, filling the huge gap between $\frac{1}{3^n-2^{n+1}+2}$ and 1. This will offer a finer view on the sensitivity to synchronism of ECA. The results are summarised in Table 2.

Class	Rules (α)	Sections	Sensitivity ($\mu_s(F_{\alpha,n})$)	Sensitivity ($\mu_s(F_\alpha)$)
I	$0, 51, 200, 204$	A.2.1	$\frac{1}{3^n-2^{n+1}+2}$ for any $n \geq 3$	0
II	$3, 12, 15, 34, 60, 136, 170$ $28, 32, 44, 140$	A.2.2	$\frac{2^n-1}{3^n-2^{n+1}+2}$ for any $n \geq 4$	0
III	8	A.2.3	$\frac{\phi^{2n}+\phi^{-2n}-2^n}{3^n-2^{n+1}+2}$ for any $n \geq 5$	0
IV	$128, 160, 162$	A.2.4	$\frac{3^n-2^{n+1}-cn+2}{3^n-2^{n+1}+2}$ for any $n \geq 5$	1

Table 2 – The rules are divided into four classes (ϕ is the golden ratio and c is a constant).

Finally, remark that the notion of sensitivity to synchronism $\mu_s(F_{\alpha,n})$ that we defined over configurations of size n with periodic boundary conditions can be naturally extended to bi-infinite configurations by setting

$$\mu_s(F_\alpha) = \lim_{n\to+\infty} \mu_s(F_{\alpha,n}).$$

Indeed, configurations of size n with periodic boundary conditions can be seen as bi-infinite configurations with spatial period n. These configurations are dense in the set of bi-infinite configurations whenever they are equipped with the standard Cantor topology. In other words, for any bi-infinite update schedule u there exists an infinite sequence p_n of periodic schedules such that $\lim_{n\to\infty} p_n = u$. Of course, in order to grant consistency, one should require that u is such that if we take a segment of size, say $2n + 1$, centered in 0, then we see all integers between $-n$ and n in the considered segment.

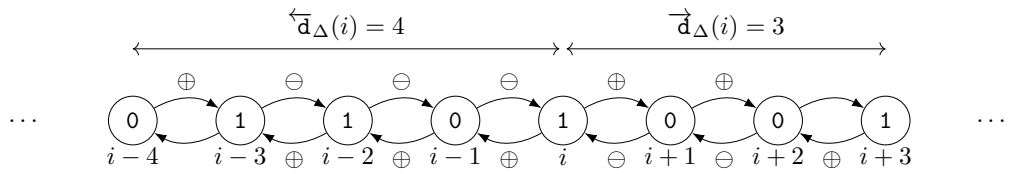


Figure A.16 – Illustration of the chain of influences for some update schedule Δ. According to Equation A.10, the result of the update of the cell i is $r_\alpha(r_\alpha(r_\alpha(r_\alpha(x_{i-4}, x_{i-3}, x_{i-2}), x_{i-2}, x_{i-1}), x_{i-1}, x_i), i, r_\alpha(x_i, x_{i+1}, r_\alpha(x_{i+1}, x_{i+2}, x_{i+3})))$.

A.2 Theoretical measures of sensitivity to synchronism

This section contains the main results of [Balbi et al. (2020), Balbi et al. (2022)], regarding the dependency on n of $\mu_s(F_{\alpha,n})$ for ECA rules that are not max-sensitive to synchronism.
As illustrated in Table 2, the ECA rules can be divided into four classes according to their sensitivity functions. Each class will require specific proof techniques but all of them have interaction digraphs as a common denominator.
As a warm up, one can consider the case of ECA rules having an interaction digraph, which is a proper subgraph of G_n^{ECA}. Indeed, when considering them as BN many distinct update schedules give the same labelings and hence, by Theorem A.1 and the definition of $\mu_s(F_{\alpha,n})$, they cannot be max-sensitive. This is the case of the following set of ECA rules $\mathcal{S} = \{0, 3, 12, 15, 34, 51, 60, 136, 170, 204\}$. Indeed, denoting by $G_{F_{\alpha,n}} = (\llbracket n \rrbracket, A_{F_{\alpha,n}})$ the interaction digraph of ECA rule α of size n, one finds that the graph $G_{F_{\alpha,n}}$ for rules 0, 51 and 204 contains no arcs as these rules do not depend on the left and right neighbour. Similarly, the other rules only depend on the left or right neighbours. This reasoning will allow us to study the sensitivity of the first two classes in Table 2.
Let us now introduce some useful results and notations that will be widely used in the following. Given an update schedule Δ, in order to study the chain of influences involved in the computation of the image at cell $i \in \llbracket n \rrbracket$, define

$$\begin{aligned}
\overleftarrow{\mathtt{d}}_\Delta(i) &= \max\{k \in \mathbb{N} \mid \forall j \in \mathbb{N},\ 0 < j < k \implies lab_\Delta((i-j, i-j+1)) = \ominus\} \\
\overrightarrow{\mathtt{d}}_\Delta(i) &= \max\{k \in \mathbb{N} \mid \forall j \in \mathbb{N},\ 0 < j < k \implies lab_\Delta((i+j, i+j-1)) = \ominus\}\,.
\end{aligned}$$

These quantities are well defined because $k = 1$ is always a possible value, and moreover, if $\overleftarrow{\mathtt{d}}_\Delta(i)$ or $\overrightarrow{\mathtt{d}}_\Delta(i)$ is greater than n, then there is a forbidden cycle in the update digraph of schedule Δ (Theorem A.2). Note that for any $\Delta \in P_n$

$$lab_\Delta((i - \overleftarrow{\mathtt{d}}_\Delta(i), i - \overleftarrow{\mathtt{d}}_\Delta(i) + 1)) = \oplus \quad \text{and} \quad lab_\Delta((i + \overrightarrow{\mathtt{d}}_\Delta(i), i + \overrightarrow{\mathtt{d}}_\Delta(i) - 1)) = \oplus.$$

See Figure A.16 for an illustration.

The purpose of these quantities is that it holds that, for any $x \in \{0,1\}^n$,

$$f_\alpha^{(\Delta)}(x)_i = r_\alpha(\underbrace{\underbrace{\underbrace{\cdots}_{}, r_\alpha(\quad, x_{i-1}, x_i)}_{r_\alpha(x_{i-\overleftarrow{\mathtt{d}}_\Delta(i)}, x_{i-\overleftarrow{\mathtt{d}}_\Delta(i)+1}, x_{i-\overleftarrow{\mathtt{d}}_\Delta(i)+2})}}_{}, x_i, \underbrace{\underbrace{r_\alpha(x_i, x_{i+1}, \quad)}_{r_\alpha(x_{i+\overrightarrow{\mathtt{d}}_\Delta(i)-2}, x_{i+\overrightarrow{\mathtt{d}}_\Delta(i)-1}, x_{i+\overrightarrow{\mathtt{d}}_\Delta(i)})}\underbrace{\cdots}_{}}_{}) \tag{A.10}$$

i.e. the quantities $\overleftarrow{\mathtt{d}}_\Delta(i)$ and $\overrightarrow{\mathtt{d}}_\Delta(i)$ are the lengths of the chain of influences at cell i for the update schedule Δ, on both sides of the interaction digraph. If the chains of influences at some cell i are identical for two update schedules, then the images at i will be identical for any configuration, as stated in the following lemma.

Lemma A.4. *For any ECA rule α, any $n \in \mathbb{N}$, any $\Delta, \Delta' \in P_n$ and any $i \in [\![n]\!]$, it holds that if $\overleftarrow{\mathtt{d}}_\Delta(i) = \overleftarrow{\mathtt{d}}_{\Delta'}(i)$ and $\overrightarrow{\mathtt{d}}_\Delta(i) = \overrightarrow{\mathtt{d}}_{\Delta'}(i)$, then $\forall x \in \{0,1\}^n,\ f_\alpha^{(\Delta)}(x)_i = f_\alpha^{(\Delta')}(x)_i$.*

Proof. This is a direct consequence of Equation A.10, because the nesting of local rules for Δ and Δ' is identical at cell i. □

For any rule α, size n, and update schedules $\Delta, \Delta' \in P_n$, it holds that

$$\forall i \in [\![n]\!],\ \overleftarrow{\mathtt{d}}_\Delta(i) = \overleftarrow{\mathtt{d}}_{\Delta'}(i) \text{ and } \overrightarrow{\mathtt{d}}_\Delta(i) = \overrightarrow{\mathtt{d}}_{\Delta'}(i) \iff \Delta \equiv \Delta' \tag{A.11}$$

and this implies $D_{F_{\alpha,n}^{(\Delta)}} = D_{F_{\alpha,n}^{(\Delta')}}$. Remark that it is possible that $\overleftarrow{\mathtt{d}}_\Delta(i) + \overrightarrow{\mathtt{d}}_\Delta(i) \geq n$ in which case the image at cell i depends on the whole configuration. Moreover, the previous inequality may be strict, meaning that the dependencies on both sides may overlap for some cell. This will be a key in computing the dependency on n of the sensitivity to synchronism for rule 128 for example. Let

$$\mathtt{d}_\Delta(i) = \{j \leq i \mid i - j \leq \overleftarrow{\mathtt{d}}_\Delta(i)\} \cup \{j \geq i \mid j - i \leq \overrightarrow{\mathtt{d}}_\Delta(i)\}$$

be the set of cells that i depends on under the update schedule $\Delta \in P_n$. When $\mathtt{d}_\Delta(i) \neq [\![n]\!]$ then cell i does not depend on the whole configuration, and $\mathtt{d}_\Delta(i)$ describes precisely Δ, as stated in the following lemma.

Lemma A.5. *For any $\Delta, \Delta' \in P_n$, it holds that if $\forall i \in [\![n]\!],\ \mathtt{d}_\Delta(i) = \mathtt{d}_{\Delta'}(i) \neq [\![n]\!]$, then $\Delta \equiv \Delta'$.*

Proof. If $\mathtt{d}_\Delta(i) \neq [\![n]\!]$ then $\overleftarrow{\mathtt{d}}_\Delta(i)$ and $\overrightarrow{\mathtt{d}}_\Delta(i)$ do not overlap. Moreover, remark that $\overleftarrow{\mathtt{d}}_\Delta(i)$ and $\overrightarrow{\mathtt{d}}_\Delta(i)$ can be deduced from $\mathtt{d}_\Delta(i)$. Indeed,

$$\begin{aligned}\overleftarrow{\mathtt{d}}_\Delta(i) &= \max\{j \mid \forall k \in [\![j]\!], i - j + k \in \mathtt{d}_\Delta(i)\} \\ \overrightarrow{\mathtt{d}}_\Delta(i) &= \max\{j \mid \forall k \in [\![j]\!], i + j - k \in \mathtt{d}_\Delta(i)\}.\end{aligned}$$

The result follows since knowing $\overrightarrow{\mathtt{d}}_\Delta(i)$ and $\overleftarrow{\mathtt{d}}_\Delta(i)$ for all $i \in [\![n]\!]$ allows one to completely reconstruct lab_Δ, which would be the same as $lab_{\Delta'}$ if $\mathtt{d}_\Delta(i) = \mathtt{d}_{\Delta'}(i)$ for all $i \in [\![n]\!]$ (Formula A.11). □

A.2.1 Class I: Insensitive rules

This class contains the simplest dynamics, with sensitivity function $\frac{1}{3^n-2^{n+1}+2}$, and it is a good starting point for our analysis.

Theorem A.6. *For any $n \geq 1$ and $\alpha \in \{0, 51, 204\}$, $\mu_s(F_{\alpha,n}) = \frac{1}{3^n-2^{n+1}+2}$.*

Proof. The result for ECA rule 0 is obvious since $\forall n \geq 1$ and $\forall x \in \{0,1\}^n$, $F_{0,n}(x) = 0^n$. The ECA Rule 51 is based on the Boolean function $r_{51}(x_{i-1}, x_i, x_{i-1}) = \neg x_i$ and ECA rule 204 is the identity. Therefore, similarly to ECA rule 0, for any n their interaction digraph has no arcs. Hence, there is only one equivalence class of update digraph, and one dynamics. □

The ECA rule 200 also belongs to Class I. It has the following local function $r_{200}(x_1, x_2, x_3) = x_2 \wedge (x_1 \vee x_3)$. Indeed, it is almost equal to the identity (ECA rule 204), except for $r_{200}(0, 1, 0) = 0$. It turns out that, even if its interaction digraph has all of the $2n$ arcs, this rule produces always the same dynamics, regardless of the update schedule.

Theorem A.7. *For any $n \geq 1$, $\mu_s(F_{200,n}) = \frac{1}{3^n-2^{n+1}+2}$.*

Proof. We prove that $F^{(\Delta)}_{200,n}(x) = F^{(\Delta^{\text{sync}})}_{200,n}(x)$ for any configuration $x \in \{0,1\}^n$ and for any update schedule $\Delta \in P_n$. For any $i \in [\![n]\!]$ such that $x_i = 0$, the ECA rule 200 is the identity, therefore it does not depend on the states of its neighbours, which may have been updated before itself, *i.e.* $f^{(\Delta)}_{200}(x)_i = 0 = f^{(\Delta^{\text{sync}})}_{200}(x)_i$. Moreover, for any $i \in [\![n]\!]$ such that $x_i = 1$, if its two neighbours x_{i-1} and x_{i+1} are both in state 0 then they will remain in state 0 and $f^{(\Delta)}_{200}(x)_i = 0 = f^{(\Delta^{\text{sync}})}_{200}(x)_i$, otherwise the ECA 200 acts as the identity rule and the two neighbours of cell i also apply the identity, thus again $f^{(\Delta)}_{200}(x)_i = 1 = f^{(\Delta^{\text{sync}})}_{200}(x)_i$. □

A.2.2 Class II: Low-sensitivity rules

This class contains rules whose sensitivity function equals $\frac{2^n-1}{3^n-2^{n+1}+2}$. This is a interesting class that demands the development of specific arguments and tools. However, the starting point is always the interaction digraph.

One-way ECA.

The following result counts the number of equivalence classes of update schedules for ECA rules α having only arcs of the form $(i, i+1)$, or only arcs of the form $(i+1, i)$ in their interaction digraph $G_{F_{\alpha,n}}$.

Lemma A.8. *For the ECA rules $\alpha \in \{3, 12, 15, 34, 60, 136, 170\}$, it holds that $|\mathcal{U}(F_{\alpha,n})| \leq 2^n-1$.*

Proof. The interaction digraph of these rules is the directed cycle on n vertices (with n arcs). There can be only a forbidden cycle of length n, in the case that all edges are labeled $\ominus$ (see Theorem A.2). Except for the $\oplus$-cycle (which is valid), any other labeling prevents the formation of an invalid cycle, since the orientation of at least one arc is unchanged (labeled $\oplus$), and the orientation of at least one arc is reversed (labeled $\ominus$). □

In the following, we are going to exploit Lemma A.8 to obtain one of the main results of this section. The idea is to show that, considering two valid and non-equivalent update schedules, it is

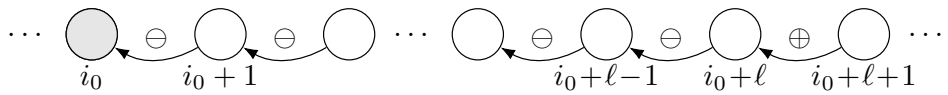
always possible to construct a configuration x such that there will be at least a cell with different values after the update procedure. The ECA rule 170 (left shift), which is based on the following Boolean function: $r_{170}(x_{i-1}, x_i, x_{i+1}) = x_{i+1}$, shows the way.

Theorem A.9. *For any* $n \geq 2$, $\mu_s(F_{170,n}) = \frac{2^n-1}{3^n-2^{n+1}+2}$.

Proof. Let $F = F_{170,n}$ and $n \geq 2$. By definition, one finds that for any two non-equivalent update schedules $\Delta \not\equiv \Delta'$ it holds that

$$\exists i_0 \in [\![n]\!]\ lab_\Delta((i_0+1, i_0)) = \oplus \text{ and } lab_{\Delta'}((i_0+1, i_0)) = \ominus.$$

Furthermore, since having $lab_{\Delta'}((i+1, i)) = \ominus$ for all $i \in [\![n]\!]$ creates an invalid cycle of length n, there exists a minimal $\ell \geq 1$ such that $lab_{\Delta'}((i_0+\ell+1, i_0+\ell)) = \oplus$ (this requires $n > 1$). A part of the update digraph corresponding to Δ' is pictured below.



By definition of the labels and the minimality of ℓ, we have that

$$\forall 0 \leq k < \ell,\ \ f^{(\Delta')}(x)_{i_0+k} = x_{i_0+\ell+1}.$$

Since, for the update schedule Δ, we have $f^{(\Delta)}(x)_{i_0} = x_{i_0+1}$, it is always possible to construct a configuration x with $x_{i_0+1} \neq x_{i_0+\ell+1}$ such that the two dynamics differ, *i.e.* $f^{(\Delta)}(x)_{i_0} \neq f^{(\Delta')}(x)_{i_0}$. The result holds by Formula A.9. □

Generalising the idea behind the construction used for ECA rule 170, one may prove that ECA rules $3, 12, 15, 34, 60, 136$ have identical sensitivity functions.

Theorem A.10. *For any* $n \geq 2$ *and for all* $\alpha \in \{3, 12, 15, 34, 60, 136\}$, $\mu_s(F_{\alpha,n}) = \frac{2^n-1}{3^n-2^{n+1}+2}$.

Proof. We present the case when the interaction digraph has only arcs of type $(i+1, i)$ (such as rule 170), the case $(i, i+1)$ is symmetric. Fix $n \geq 2$ and choose two update schedules $\Delta, \Delta' \in P_n$ such that $\Delta \not\equiv \Delta'$, then it holds that:

$$\exists i_0 \in [\![n]\!],\ lab_\Delta((i_0+1, i_0)) = \oplus \text{ and } lab_{\Delta'}((i_0+1, i_0)) = \ominus,$$
$$\exists \ell \in [\![n]\!],\ (\forall 0 \leq k < \ell,\ lab_{\Delta'}((i_0+k+1, i_0+k)) = \ominus) \text{ and } (lab_{\Delta'}((i_0+\ell+1, i_0+\ell)) = \oplus).$$

Fix $\alpha \in \{3, 12, 15, 34, 60, 136\}$ and let r be the corresponding Boolean function. Moreover let $F = F_{\alpha,n}$. We know that, for any $x \in \{0,1\}^n$, we will have $f^{(\Delta)}(x)_{i_0} = r(b, x_{i_0}, x_{i_0+1})$ for any $b \in \{0,1\}$. Our goal is to construct a configuration $x \in \{0,1\}^n$ such that $f^{(\Delta)}(x)_{i_0} \neq f^{(\Delta')}(x)_{i_0}$. In order to start, we need

$$\exists x_{i_0}, x_{i_0+1}, o_1, o_2 \in \{0,1\}, \forall b, b' \in \{0,1\},\quad r(b, x_{i_0}, x_{i_0+1}) \neq r(b', x_{i_0}, o_1) \text{ and } r(x_{i_0}, x_{i_0+1}, o_2) = o_1. \tag{A.12}$$

In other words, we can choose x_{i_0}, x_{i_0+1} so that there is a target output o_1 for $f^{(\Delta')}(x)_{i_0+1}$, such that if $f^{(\Delta')}(x)_{i_0+1} = o_1$ then $f^{(\Delta)}(x)_{i_0} \neq f^{(\Delta')}(x)_{i_0}$ (the values of b and b' do not matter). Similarly, given x_{i_0}, x_{i_0+1}, o_1, there is a target output o_2 for $f^{(\Delta')}(x)_{i_0+2}$. We can now construct

x by finite induction by expressing such target outputs. In order to continue we want (the idea is to use this by induction for all $0 < k \leq \ell$)

$$\forall x_{i_0+k-1}, o_k \in \{0,1\}, \exists x_{i_0+k}, o_{k+1} \in \{0,1\},\ r(x_{i_0+k-1}, x_{i_0+k}, o_{k+1}) = o_k. \tag{A.13}$$

If Formulas A.12 and A.13 hold then we can construct the desired configuration x. Indeed, Formula A.12 gives $x_{i_0}, o_1, x_{i_0+1}, o_2$, and by induction, knowing x_{i_0+k-1}, o_k Formula A.13 gives x_{i_0+k}, o_{k+1} for $0 < k \leq \ell$. The construction ends with $x_{i_0+\ell+1} = o_{\ell+1}$. A sequence $(x_i, o_i) \in \{0,1\}^2$ for $i \in [\![\ell]\!]$, which satisfies both Formulas A.12 and A.13, is called a *witness* sequence. Given a witness sequence $(x_i, o_i)_{i \in [\![\ell]\!]}$ it holds that $f^{(\Delta)}(x)_{i_0} \neq f^{(\Delta')}(x)_{i_0}$, and hence, by Formula A.9 we have the result. We end by providing the witness sequences for all the local rules in the hypothesis. We start by those rules which have an interaction digraph made by arcs of type $(i+1, i)$.

- Rule 34 :
  - Formula (A.12): $x_{i_0} = 0, x_{i_0-1} = 1, o_1 = 1, o_2 = 1$.
  - Formula (A.13): $(x_{i_0+k}, o_{k+1}) = (0,1)$.
- Rule 136 :
  - Formula (A.12): $x_{i_0} = 1, x_{i_0-1} = 1, o_1 = 0, o_2 = 0$.
  - Formula (A.13): $(x_{i_0+k}, o_{k+1}) = \begin{cases} (0,0), & \text{if } k \text{ is even} \\ (1,1), & \text{otherwise} \end{cases}$

We conclude with the witness sequences for the rules which have interaction digraph made by arcs of type $(i, i+1)$.

- Rule 3 :
  - Formula (A.12): $x_{i_0} = 0, x_{i_0+1} = 0, o_{-1} = 1, o_{-2} = 0$.
  - Formula (A.13): $(x_{i_0-k}, o_{-k-1}) = \begin{cases} (0,1), & \text{if } k \text{ is even} \\ (0,0), & \text{otherwise} \end{cases}$
- Rule 12 :
  - Formula (A.12): $x_{i_0} = 1, x_{i_0+1} = 1, o_{-1} = 0, o_{-2} = 1$.
  - Formula (A.13): $(x_{i_0-k}, o_{-k-1}) = \begin{cases} (0,0), & \text{if } k \text{ is even} \\ (1,0), & \text{otherwise} \end{cases}$
- Rule 15 :
  - Formula (A.12): $x_{i_0} = 0, x_{i_0+1} = 0, o_{-1} = 1, o_{-2} = 0$.
  - Formula (A.13): $(x_{i_0-k}, o_{-k-1}) = \begin{cases} (0,1), & \text{if } k \text{ is even} \\ (0,0), & \text{otherwise} \end{cases}$
- Rule 60 :
  - Formula (A.12): $x_{i_0} = 0, x_{i_0+1} = 0, o_{-1} = 1, o_{-2} = 1$.

– Formula (A.13): $(x_{i_0-k}, o_{-k-1}) = \begin{cases} (0,0), & \text{if } k \text{ is even} \\ (0,1), & \text{otherwise} \end{cases}$

□

Example A.1 – Consider the ECA rule $\alpha = 34$ and a size $n = 6$. Given the two distinct update schedules $\Delta = (\{3,4\},\{5\},\{2\},\{0,1\})$ and $\Delta' = (\{3,4\},\{5\},\{0,1,2\})$. Let $i_0 = 1$ and $\ell = 2$. The following is a witness sequence (see the proof of Theorem A.10): $x_{i_0-1} = 1, x_{i_0} = 0, o_1 = 1, x_{i_0+1} = 0, o_2 = 1, x_{i_0+2} = 0, o_3 = 1$, and finaly $x_{i_0+3} = o_3 = 1$. By construction it ensures (the value of x_{i_0+4} is set, arbitrarily, to 0)

$$f^{(\Delta')}(100010)_{i_0} = r(1,0,r(0,0,r(0,0,1))) = r(1,0,r(0,0,1)) = r(1,0,1) = 1$$
$$\neq f^{(\Delta)}(100010)_{i_0} = r(100) = 0.$$

Exploiting patterns in the update digraph

In this subsection, we are going to develop a proof technique, which characterises the number of non-equivalent update schedules according to the presence of specific patterns in their interaction digraph. We will show in which cases the differences between two update schedules lead to differences in the dynamics and in which cases they do not. Consequently, the proofs for the different rules will be similar in structure but different in the details depending on which rule is considered. This will concern ECA rules $28, 32, 44$ and 140.
We begin with the ECA Rule 32, which is based on the Boolean function $r_{32}(x_1, x_2, x_3) = x_1 \wedge \neg x_2 \wedge x_3$.

Lemma A.11. *Fix $n \in \mathbb{N}$. For any update schedule $\Delta \in P_n$, for any configuration $x \in \{0,1\}^n$ and for any $i \in [\![n]\!]$, the following holds:*

$$f_{32}^{(\Delta)}(x)_i = 1 \iff lab_\Delta((i+1,i)) = lab_\Delta((i-1,i)) = \oplus \text{ and } (x_{i-1}, x_i, x_{i+1}) = (1,0,1).$$

Proof.
($\Leftarrow$) Since $lab_\Delta((i+1,i)) = lab_\Delta((i-1,i)) = \oplus$ (see Figure A.17) this means that cells $i-1$ and $i+1$ are updated at the same time or after the cell i, therefore $f_{32}^{(\Delta)}(x)_i = r_{32}(x_{i-1}, x_i, x_{i+1}) = r_{32}(1,0,1) = 1$.
($\Rightarrow$) Choose $x \in \{0,1\}^n$ such that $f_{32}^{(\Delta)}(x)_i = 1$. Assume that $lab_\Delta((i+1,i)) = lab_\Delta((i-1,i)) = \oplus$ but $(x_{i-1}, x_i, x_{i+1}) \neq (1,0,1)$. By the same reasoning as above, we have $f_{32}^{(\Delta)}(x)_i = r_{32}(x_{i-1}, x_i, x_{i+1}) = 0$, since $(x_{i-1}, x_i, x_{i+1}) \neq (1,0,1)$. Now, assume $lab_\Delta((i+1,i)) = \ominus$ or $lab_\Delta((i-1,i)) = \ominus$. Then,

$$f_{32}^{(\Delta)}(x)_i = \begin{cases} r_{32}(0,0,1) = 0, \text{if } lab_\Delta((i-1,i)) = \ominus \text{ and } lab_\Delta((i+1,i)) = \oplus \\ r_{32}(1,0,0) = 0, \text{if } lab_\Delta((i-1,i)) = \oplus \text{ and } lab_\Delta((i+1,i)) = \ominus \\ r_{32}(0,0,0) = 0, \text{if } lab_\Delta((i-1,i)) = \ominus \text{ and } lab_\Delta((i+1,i)) = \ominus \end{cases}$$

which contradicts the hypothesis. □

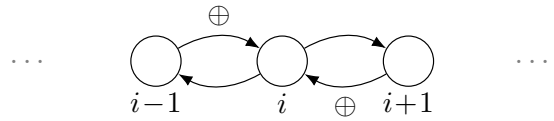


Figure A.17 – Labeling presented in the Lemma A.11. This is the only situation in which we can obtain a cell updated to 1.

Corollary A.12. *Fix $n \in \mathbb{N}$. For any update schedule $\Delta \in P_n$, for any configuration $x \in \{0,1\}^n$ and $i \in [\![n]\!]$, if $lab_\Delta((i-1,i)) = \ominus$ or $lab_\Delta((i+1,i)) = \ominus$, then $f_{32}^{(\Delta)}(x)_i = 0$.*

Lemma A.13. *For any $n \in \mathbb{N}$. Consider a pair of update schedules $\Delta, \Delta' \in P_n$. Then, $D_{F_{32,n}^{(\Delta)}} \neq D_{F_{32,n}^{(\Delta')}}$ if and only if there exists $i \in [\![n]\!]$ such that: $lab_\Delta((i+1,i)) = lab_\Delta((i-1,i)) = \oplus$ and either $lab_{\Delta'}((i+1,i)) = \ominus$ or $lab_{\Delta'}((i-1,i)) = \ominus$, or the symmetric condition (on Δ', Δ).*

Proof.
($\Leftarrow$) Without loss of generality, suppose that $lab_\Delta((i+1,i)) = lab_\Delta((i-1,i)) = \oplus$ and $lab_{\Delta'}((i+1,i)) = \ominus$ or $lab_{\Delta'}((i-1,i)) = \ominus$ (the other case is the same where Δ and Δ' are exchanged). Then, by Lemma A.11 one can find $f_{32}^{(\Delta)}(x)_i = 1$ and by Corollary A.12, $f_{32}^{(\Delta')}(x)_i = 0$. Therefore, $D_{F_{32,n}^{(\Delta)}} \neq D_{F_{32,n}^{(\Delta')}}$.
($\Rightarrow$) Suppose that for every $i \in [\![n]\!]$ one of the following holds:

(Case 1) $lab_\Delta((i+1,i)) = lab_\Delta((i-1,i)) = lab_{\Delta'}((i+1,i)) = lab_{\Delta'}((i-1,i)) = \oplus$

(Case 2) $(lab_\Delta((i+1,i)), lab_\Delta((i-1,i))) \neq (\oplus,\oplus) \neq (lab_{\Delta'}((i+1,i)), lab_{\Delta'}((i-1,i)))$.

We will show that in both cases $D_{F_{32,n}^{(\Delta)}} = D_{F_{32,n}^{(\Delta')}}$. Let $j \in [\![n]\!]$ and consider a configuration $x \in \{0,1\}^n$ such that: $(x_{j-1}, x_j, x_{j+1}) \neq (1,0,1)$, then by Lemma A.11, $f_{32}^{(\Delta)}(x)_j = f_{32}^{(\Delta')}(x)_j = 0$. Now suppose $(x_{j-1}, x_j, x_{j+1}) = (1,0,1)$. If we are in Case 1, then $f_{32}^{(\Delta)}(x)_j = f_{32}^{(\Delta')}(x)_j = 1$. If we are in Case 2, then $f_{32}^{(\Delta)}(x)_j = f_{32}^{(\Delta')}(x)_j = 0$. By the generality of j, $F_{32,n}^{(\Delta)}(x) = F_{32,n}^{(\Delta')}(x)$ and by the generality of x, $D_{F_{32,n}^{(\Delta)}} = D_{F_{32,n}^{(\Delta')}}$. □

For the ECA rules 28, 44 and 140 we are going to develop a similar construction as the one for ECA rule 32 but before let us recall the Boolean functions which they are based on. We start with ECA rule 44, which is based on the Boolean function $r_{44}(x_1,x_2,x_3) = (\neg x_1 \wedge x_2) \vee (x_1 \wedge \neg x_2 \wedge x_3)$, which implies that $r_{44}(1,0,1) = r_{44}(0,1,1) = r_{44}(0,1,0) = 1$.

Notation. *Let us call $\star$ a possible value of a cell in the configuration that has no effect on the result of the update procedure over the cells under consideration. At the same time, we will use a letter to represent the value of a cell in the configuration that is unknown and that affects the result of the update procedure over cells under consideration.*

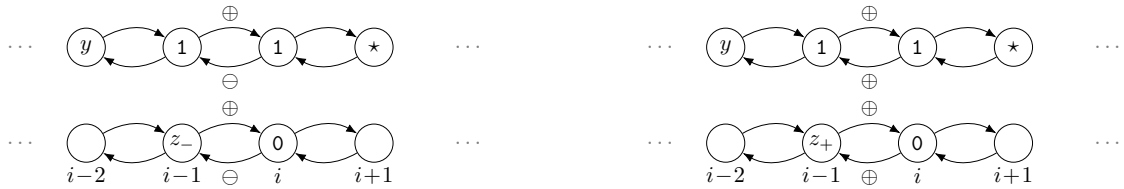


Figure A.18 – On the left, starting from a configuration $(y, 1, 1, \star)$, the schedule Δ updates the cell i before cell $i-1$. Therefore, the cell i becomes 0 and the cell $i-1$ is updated to $z_- = r_{44}(y, 1, 0)$. On the right, according to Δ', the cells i and $i-1$ are updated at the same time and $z_+ = r_{44}(y, 1, 1)$.

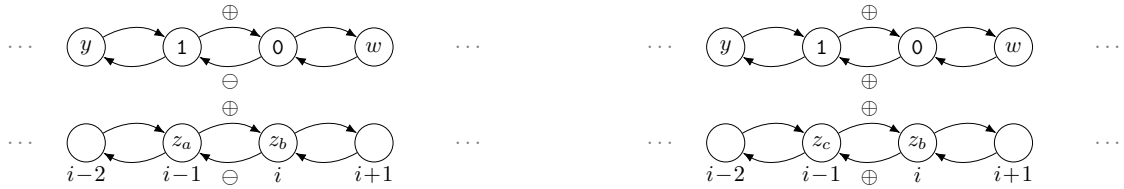


Figure A.19 – On the left, starting from a configuration $(y, 1, 0, w)$, the schedule Δ updates the cell i before cell $i-1$. Therefore, the cell i becomes $z_b = r_{44}(1, 0, w)$, and the cell $i-1$ is updates to $z_a = r_{44}(y, 1, z_b)$. On the right, according to Δ', the cells i and $i-1$ are updated at the same time. Therefore, z_c is equal to $r_{44}(y, 1, 0)$ and z_b is equal to $r_{44}(1, 0, w)$.

Lemma A.14. *Given two update schedules Δ and Δ', if there exists $i \in [\![n]\!]$ such that*

- $lab_\Delta((i, i-1)) = \ominus$ *and* $lab_{\Delta'}((i, i-1)) = \oplus$,
- $lab_\Delta((i-1, i)) = lab_{\Delta'}((i-1, i)) = \oplus$,
- $lab_\Delta((j, j-1)) = lab_{\Delta'}((j, j-1))$ *and* $lab_\Delta((j-1, j)) = lab_{\Delta'}((j-1, j))$ *for each* $j \neq i, j \in [\![n]\!]$,

then $D_{F^{(\Delta)}_{44,n}} = D_{F^{(\Delta')}_{44,n}}$.

Proof. Given two update schedules Δ and Δ', we prove that $(f^{(\Delta)}_{44}(x)_{i-1} = f^{(\Delta')}_{44}(x)_{i-1})$ and $(f^{(\Delta)}_{44}(x)_i = f^{(\Delta')}_{44}(x)_i)$ for every possible starting configuration x.
Starting from the case with $x_{i-1} = x_i = 1$ (see Figure A.18), one obtains cells $i-1$ and i updated to states $r_{44}(y, 1, 0)$ and 0 (respectively) according to the Δ update schedule and to states $r_{44}(y, 1, 1)$ and 0 (respectively) according to the Δ' update schedule. According to the rule, we know that $r_{44}(0, 1, 0) = r_{44}(0, 1, 1) = 1$ and $r_{44}(1, 1, 0) = r_{44}(1, 1, 1) = 0$, as a consequence, the equivalence holds in the case of $x_{i-1} = x_i = 1$.
If we consider $x_{i-1} = 1$ and $x_i = 0$ (see Figure A.19), one obtains cells $i-1$ and i updated to states $r_{44}(y, 1, r_{44}(1, 0, w))$ and $r_{44}(1, 0, w)$ (respectively) according to the Δ update schedule and to states $r_{44}(y, 1, 0)$ and $r_{44}(1, 0, w)$ (respectively) according to the Δ' update schedule. Like in the previous case, the result of the update procedure depends only on the y value, which will be the same in Δ and in Δ', as a consequence, the equivalence holds in this case. If we consider the opposite case $x_{i-1} = 0$ and $x_i = 1$ (see Figure A.20), one obtains cells $i-1$ and i updated to states $r_{44}(y, 0, 1)$ and 1 (respectively) according to the Δ update schedule and to states $r_{44}(y, 0, 1)$ and

Figure A.20 – On the left, starting from a configuration $(y, \mathtt{0}, \mathtt{1}, \star)$, the schedule Δ updates the cell i before cell $i-1$. Therefore, the cell i becomes $r_{44}(\mathtt{0}, \mathtt{1}, \star) = \mathtt{1}$, and the cell $i-1$ is updated to $z_- = r_{44}(y, \mathtt{0}, \mathtt{1})$. On the right, according to Δ', the cells i and $i-1$ are updated at the same time. Therefore, x_i becomes $\mathtt{1}$ and x_{i-1} is updated to $z_+ = r_{44}(y, \mathtt{0}, \mathtt{1})$.

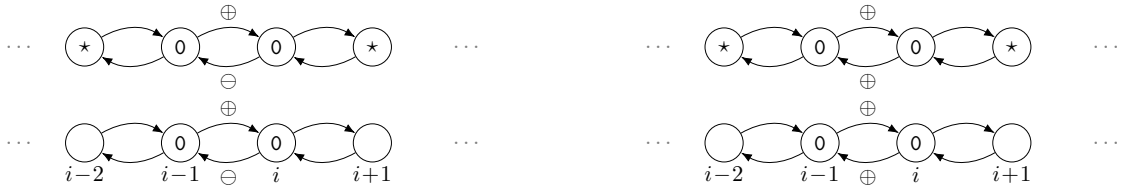


Figure A.21 – On the left, starting from a configuration $(\star, \mathtt{0}, \mathtt{0}, \star)$, the schedule Δ updates updates the cell i before cell $i-1$. Therefore, the cell i becomes $\mathtt{0}$, and the cell $i-1$ is updated to $r_{44}(\star, \mathtt{0}, \mathtt{0})$. On the right, according to Δ', the cells i and $i-1$ are updated at the same time. Remark that $r_{44}(\mathtt{0}, \mathtt{0}, \star) = r_{44}(\star, \mathtt{0}, \mathtt{0}) = \mathtt{0}$.

$\mathtt{1}$ (respectively) according to the Δ' update schedule, as a consequence, the equivalence holds also in this case. The last case corresponds to $x_{i-1} = x_i = \mathtt{0}$ (see Figure A.21), one obtains cells $i-1$ and i updated to states $\mathtt{0}$ and $\mathtt{0}$ according to Δ and Δ', as a consequence, the equivalence holds. The two different update schedules give the same configurations independently from the initial configuration, in other words $D_{F^{(\Delta)}_{44,n}} = D_{F^{(\Delta')}_{44,n}}$. □

Lemma A.15. *Given two update schedules Δ and Δ', $D_{F^{(\Delta)}_{44,n}} \neq D_{F^{(\Delta')}_{44,n}} \iff \exists i \in [\![n]\!]$ such that $lab_\Delta((i-1,i)) = \ominus$ and $lab_{\Delta'}((i-1,i)) = \oplus$, or such that the symmetric condition (on Δ', Δ) is valid.*

Proof. We can consider $lab_\Delta((i, i-1)) = lab_{\Delta'}((i, i-1)) = \oplus$ because according to Lemma A.14 the value of $lab_{\Delta'}((i, i-1))$ cannot change the dynamics that we are considering and the value of $lab_\Delta((i, i-1))$ must be $\oplus$ given the $\ominus$ in the opposite sense.
We can consider equal labelings over the other transitions.
Let j be a cell such that $lab_{\Delta'}((j, j+1)) = \oplus$ and $lab_{\Delta'}((j+k, j+k+1)) = \ominus$ for all $1 \leq k \leq i-j-1$. Such a j must exist since otherwise we would have a $\ominus-$cycle of length n. Now, let $x \in \{\mathtt{0}, \mathtt{1}\}^n$ be any configuration of length n such that

$$x_{[j,i+1]} = \begin{cases} \mathtt{1}(\mathtt{1})^{(i-1)-j-1}\mathtt{011}, \text{ if } (i-1)-j-1 \mod 2 = 0 \text{ or } j = i-2 \\ \mathtt{0}(\mathtt{1})^{(i-1)-j-1}\mathtt{011}, \text{ otherwise} \end{cases} . \quad \text{(A.14)}$$

Then we have

$$\left(f_{44}^{(\Delta)}(x)_{[j+1,i]}\right) = \begin{cases} \mathtt{0(10)}^{\lfloor \frac{(i-1)-j-2}{2} \rfloor}\mathtt{110}, \text{ if } (i-1)-j-1 \mod 2 = 0 \\ \mathtt{10}, \text{ if } j = i-2 \\ \mathtt{1(01)}^{\lfloor \frac{(i-1)-j-2}{2} \rfloor}\mathtt{10}, \text{ otherwise} \end{cases}.$$

In general we can always obtain $f_{44}^{(\Delta)}(x)_i = \mathtt{0}$ (see Figure A.22). The update schedule Δ' gives $f_{44}^{(\Delta')}(x)_i = \mathtt{1}$. Therefore, $f_{44}^{(\Delta)}(x)_i \neq f_{44}^{(\Delta')}(x)_i$ and $D_{F_{44,n}^{(\Delta)}} \neq D_{F_{44,n}^{(\Delta')}}$. □

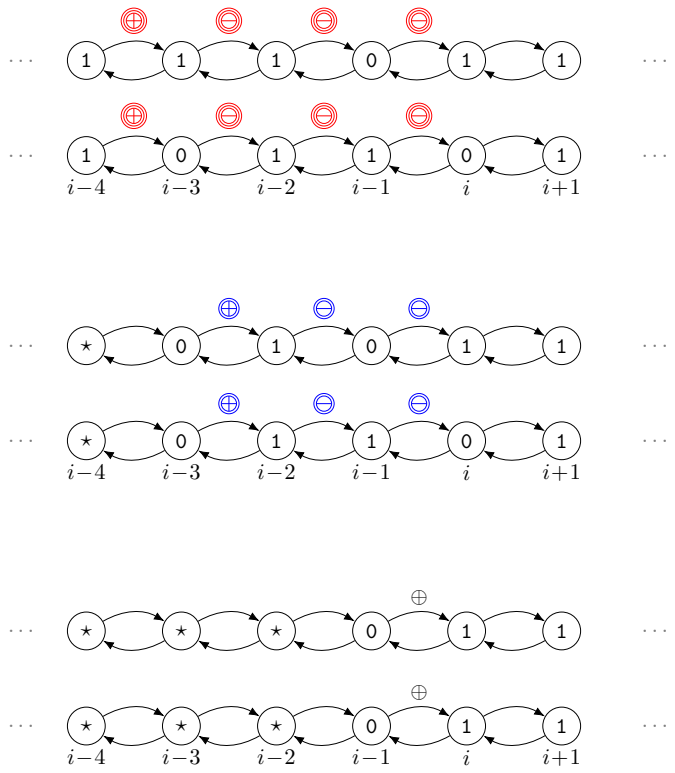


Figure A.22 – Blue double-circled and red triple-circled labels show the different cases of Equation A.14 and the black ones represent Δ'. Recall that we cannot have a $\ominus$-cycle in the labeled interaction digraph of update schedule Δ' and then there must be a cell j such that $lab_{\Delta'}((j, j+1)) = \oplus$ and $lab_{\Delta'}((j+k, j+k+1)) = \ominus$ for all $1 \leq k \leq i-j-1$.

Consider now the ECA rule 28, it is based on $r_{28}(x_1, x_2, x_3) = (\neg x_1 \wedge x_2) \vee (x_1 \wedge \neg x_2 \wedge \neg x_3)$ and hence $r_{28}(\mathtt{1},\mathtt{0},\mathtt{0}) = r_{28}(\mathtt{0},\mathtt{1},\mathtt{1}) = r_{28}(\mathtt{0},\mathtt{1},\mathtt{0}) = \mathtt{1}$. Remark that Lemma A.14 holds also for this rule. The only difference is in the proof, for completeness we show the equivalence that holds for every possible starting configuration (see Figure A.23).
For this rule also Lemma A.15 can be applied. The main idea is the same. In fact, let $x \in \{\mathtt{0},\mathtt{1}\}^n$ be any configuration of length n such that

$$x_{[j,i+1]} = \begin{cases} \mathtt{1(1)}^{(i-1)-j-1}\mathtt{100}, \text{ if } (i-1)-j-1 \mod 2 = 0 \text{ or } j = i-2 \\ \mathtt{0(1)}^{(i-1)-j-1}\mathtt{100}, \text{ otherwise} \end{cases}. \quad \text{(A.15)}$$

Then we have

$$\left(f_{28}^{(\Delta)}(x)_{[j+1,i]}\right) = \begin{cases} (01)^{\lfloor \frac{(i-1)-j}{2} \rfloor}00, \text{ if } (i-1)-j-1 \mod 2 = 0 \\ 00, \text{ if } j = i-2 \\ 1(01)^{\lfloor \frac{(i-1)-j-2}{2} \rfloor}00, \text{ otherwise} \end{cases}.$$

In general we obtain $f_{28}^{(\Delta)}(x)_i = 0$. The update schedule Δ' gives $f_{28}^{(\Delta')}(x)_i = 1$. Therefore, $f_{28}^{(\Delta)}(x)_i \neq f_{28}^{(\Delta')}(x)_i$ and $D_{F_{28,n}^{(\Delta)}} \neq D_{F_{28,n}^{(\Delta')}}$.
Let us now focus our attention on ECA rule 140, which is based on the Boolean function $r_{140}(x_1, x_2, x_3) = (\neg x_1 \vee x_3) \wedge x_2$, that is to say $r_{140}(1,1,1) = r_{140}(0,1,1) = r_{140}(0,1,0) = 1$.

Lemma A.16. *For any $n > 3$, given two update schedules $\Delta, \Delta' \in P_n$, if there exists $i \in [\![n]\!]$ such that*

- $lab_\Delta((i, i+1)) = \ominus$ *and* $lab_{\Delta'}((i, i+1)) = \oplus$
- $lab_\Delta((i+1, i)) = lab_{\Delta'}((i+1, i)) = \oplus$
- $lab_\Delta((j, j-1)) = lab_{\Delta'}((j, j-1))$ *and* $lab_\Delta((j-1, j)) = lab_{\Delta'}((j-1, j))$ *for each* $j \neq i+1, j \in [\![n]\!]$

then $D_{F_{140,n}^{(\Delta)}} = D_{F_{140,n}^{(\Delta')}}$.

Proof. Given the two update schedules $\Delta, \Delta' \in P_n$, using the same reasoning as for Lemma A.14, one can prove that $f_{140}^{(\Delta)}(x)_i = f_{140}^{(\Delta')}(x)_i$ and $f_{140}^{(\Delta)}(x)_{i+1} = f_{140}^{(\Delta')}(x)_{i+1}$ for every possible starting configuration $x \in \{0,1\}^n$. It is easy to see from the Figures A.24, A.25, A.26 and A.27 that the equivalence holds for every possible initial configuration. The two different update schedules give the same configurations independently from the initial configuration, in other words $D_{F_{140,n}^{(\Delta)}} = D_{F_{140,n}^{(\Delta')}}$. □

Remark A.1. The ECA rule 140 is such that $r_{140}(x_1, 0, x_2) = 0$ for any $x_1, x_2 \in \{0,1\}$, hence for any given update schedule Δ a cell that is in state 0 will remain in such a state throughout the whole evolution.

Lemma A.17. *Given two update schedules Δ and Δ', $D_{F_{140,n}^{(\Delta)}} \neq D_{F_{140,n}^{(\Delta')}} \iff \exists i \in [\![n]\!]$ such that $lab_\Delta((i+1, i)) = \ominus$ and $lab_{\Delta'}((i+1, i)) = \oplus$, or the symmetric condition (on Δ', Δ).*

Proof. Choose n, Δ and Δ' as in the hypothesis. We are going to prove that there exists a configuration such that $f_{140}^{(\Delta)}(x)_i \neq f_{140}^{(\Delta')}(x)_i$. Consider the following initial configuration $(x_{i-1}, x_i, x_{i+1}, x_{i+2}) = (1,1,1,0)$ and assume that $lab_\Delta((i-1, i)) = \oplus$ (according to Lemma A.16, this is not changing the dynamics). Moreover, assume that $lab_\Delta((i+1, i)) \neq lab_{\Delta'}((i+1, i))$ is the only difference between the two update schedules. According to Δ', i and $i+1$ are updated together, therefore the final configuration is $(1,1,0,0)$. In the case of Δ, the cell $i+1$ is updated before than i holding $r_{140}(1,1,0) = 0$. In a second moment, the cell i is updated and $r_{140}(1,1,0) = 0$. It follows that $f_{140}^{(\Delta)}(x)_i \neq f_{140}^{(\Delta')}(x)_i$ and $D_{F_{140,n}^{(\Delta)}} \neq D_{F_{140,n}^{(\Delta')}}$. Remark that the cell $i+1$ can be influenced from a $\ominus$-chain, but a cell with value 0 is frozen at this state. □

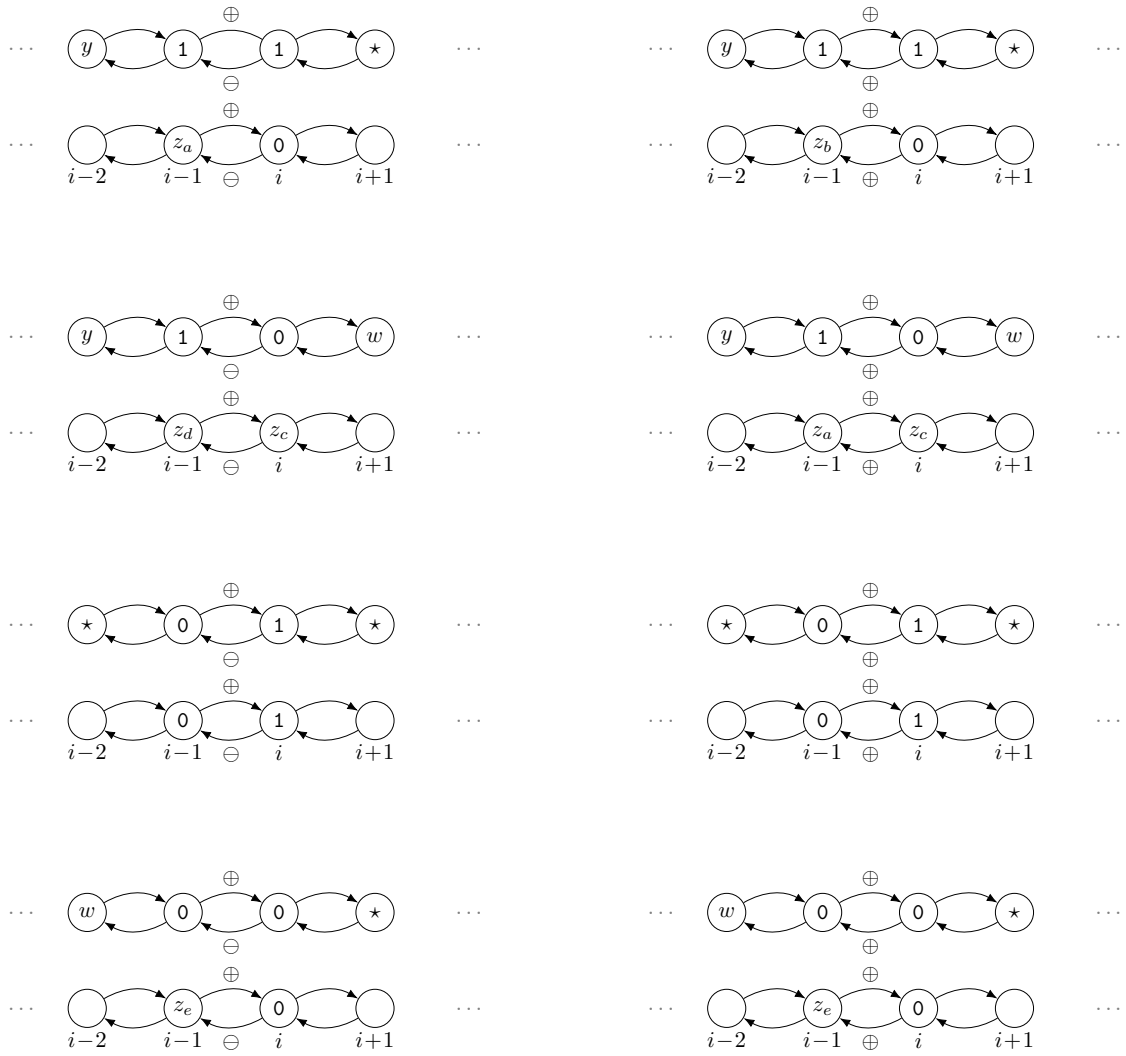


Figure A.23 – Starting from every possible configuration, a difference in a label over the edge between $i+1$ and i is insufficient to obtain different final configurations for rule 28. In the figure: $z_a = r_{28}(y, 1, 0)$, $z_b = r_{28}(y, 1, 1)$, $z_c = r_{28}(1, 0, w)$, $z_d = r_{28}(y, 1, z_c)$ and $z_e = r_{28}(w, 0, 0)$. We need also to consider that $z_a = z_b = y$ and $z_c = z_d = 0$ if $y = 1$, $z_c = z_d = 1$ otherwise.

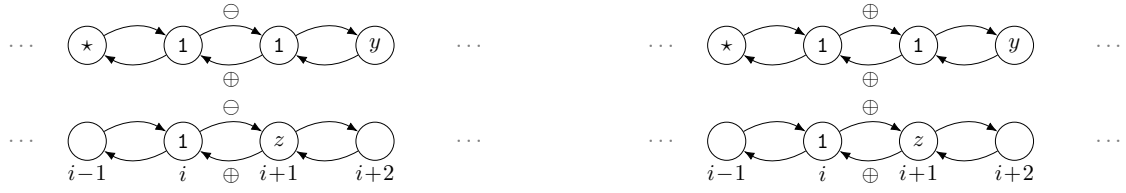


Figure A.24 – On the left, starting from a configuration $(\star, 1, 1, y)$, the schedule Δ updates the cell i before cell $i+1$. Therefore, the cell i becomes $r_{140}(\star, 1, 1) = 1$, and the cell $i+1$ becomes $z = r_{140}(1, 1, y)$. On the right, according to Δ', the cells i and $i+1$ are updated at the same time.

Figure A.25 – On the left, starting from a configuration $(y, 1, 0, \star)$, the schedule Δ updates the cell i before cell $i+1$. Therefore, the cell i becomes $z_a = r_{140}(y, 1, 0)$, and the cell $i+1$ s updated to $r_{140}(r_{140}(y, 1, 0), 0, \star) = 0$. On the right, according to Δ', the cells i and $i+1$ are updated at the same time. Remember that $r_{140}(1, 0, \star)$ is equal to 0.

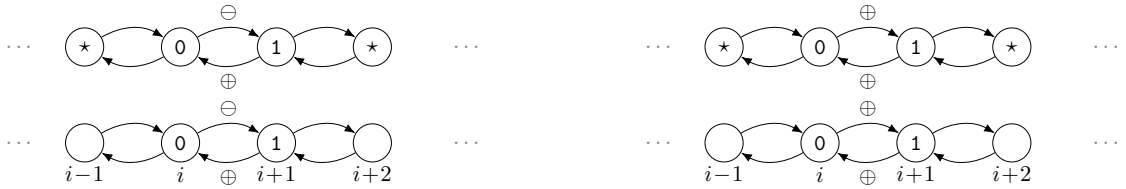


Figure A.26 – On the left, starting from a configuration $(\star, 0, 1, \star)$, the schedule Δ updates the cell i before cell $i+1$. Therefore, the cell i becomes 0 (in fact $r_{140}(\star, 0, 1) = 0$), and the cell $i+1$ is updated to $r_{140}(0, 1, \star) = 1$. On the right, according to Δ', the cells i and $i+1$ are updated at the same time.

The previous lemma is sufficient to determine that two update schedules that differ in at least one cell i such that $lab_{\Delta}((i+1, i)) = \ominus$ and $lab_{\Delta'}((i+1, i)) = \oplus$ generate two different dynamics. Indeed, one can focus on one of these cells to build a configuration in which the cell updated produces different values.
We are now ready to state the result regarding rules 28, 32, 44, 140.

Theorem A.18. *For any $n > 3$ and for all $\alpha \in \{28, 32, 44, 140\}$, $\mu_s(F_{\alpha,n}) = \frac{2^n-1}{3^n-2^{n+1}+2}$.*

Proof. Given a configuration of length $n > 3$, we have necessary and sufficient conditions to count the number of different dynamics according to the update schedules only (Lemma A.13 for ECA rule 32, Lemma A.15 for ECA rules 28 and 44, and Lemma A.17 for ECA rule 140). Intuitively, these conditions are expressed in terms of local labeling patterns, which may be present at k cells

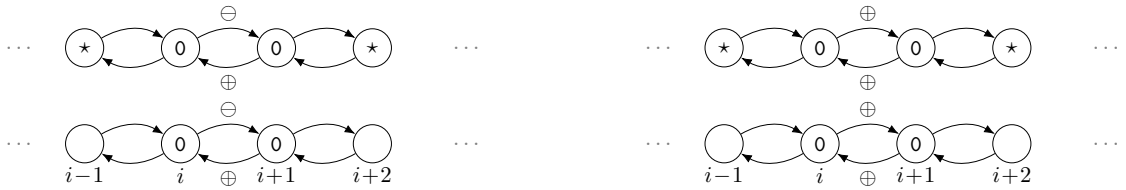


Figure A.27 – On the left, starting from a configuration $(\star, 0, 0, \star)$, the schedule Δ updates the cell i before cell $i+1$. Therefore, the cell i becomes 0, and the cell $i+1$ becomes $r_{140}(0, 0, \star) = 0$. On the right, according to Δ', the cells i and $i+1$ are updated at the same time. Remark that $r_{140}(\star, 0, 0) = 0$ and $r_{140}(0, 0, \star) = 0$.

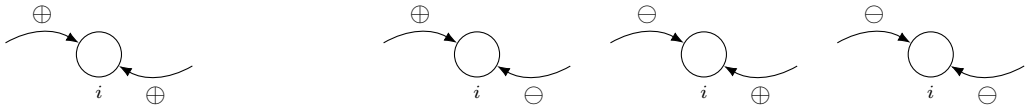


Figure A.28 – Lemma A.13 defines two equivalence classes of patterns at each position $i \in [\![n]\!]$: the class $\{A\}$ and the class $\{B, C, D\}$.

out of n. Therefore, there are $\sum_{k=0}^{n} \binom{n}{k} = 2^n$ possible different dynamics (one of which actually corresponds to a forbidden labeling).
Let us start with rule 44. Lemma A.15 states that it is necessary and sufficient to have a different labeling on an arc of the form $(i-1, i)$ for some $i \in [\![n]\!]$. As a consequence, there are at most 2^n possibilities. However, all-$\ominus$-labels gives a forbidden cycle, and any other combination of at most $n-1$ $\ominus$-labels on arcs of the form $(i-1, i)$ can be encountered on a valid labeling (by labeling all arcs of the form $(i, i-1)$ with $\oplus$, and if there is at least one $\ominus$-label on $(i-1, i)$ then label $\ominus$ one arc $(j, j-1)$ such that $(j-1, j)$ is labeled $\oplus$). Hence the exact count is $2^n - 1$. Rule 28 has the same caracterisation (Lemma A.15 again), and rule 140 has a symmetric caracterisation on arcs of the form $(i+1, i)$ (Lemma A.17).
Regarding rule 32, let A, B, C, D be the patterns defined on Figure A.28, then Lemma A.13 states there are two equivalence classes of patterns at each position $i \in [\![n]\!]$: class $\{A\}$ and class $\{B, C, D\}$, and that it is necessary and sufficient to have patterns from different classes at some $i \in [\![n]\!]$. As a consequence, there are at most 2^n possibilities. However, any combination of n patterns from the class $\{B, C, D\}$ gives a forbidden cycle:

- a B at $i \in [\![n]\!]$ can only be followed by a B at $i+1$ to avoid a forbidden cycle of length two, and by induction it gives a forbidden cycle of length n (counter-clockwise),
- a C at $i \in [\![n]\!]$ can only be preceded by a C at $i-1$ to avoid a forbidden cycle of length two, and by induction it gives a forbidden cycle of length n (clockwise),
- a D at $i \in [\![n]\!]$ can only be followed by a B at $i+1$ and by the first case this leads to an impossibility.

We deduce that the number of different dynamics is at most $2^n - 1$, and now argue that it is at least $2^n - 1$:

- if there is no pattern from the class $\{B, C, D\}$ it gives a valid labeling (all $\oplus$),
- if there is one pattern from the class $\{B, C, D\}$ then choose D and it is valid,
- if there are between two and $n-1$ patterns from the class $\{B, C, D\}$, then there is $i \in [\![n]\!]$ such that we can set pattern A at i and pattern C at $i+1$, and for the other positions always choose B from the class $\{B, C, D\}$ (and of course A from the class $\{A\}$): this is valid.

We conclude that the exact count is again $2^n - 1$. □

A.2.3 Class III: Medium-sensitivity rules

This subsection is concerned uniquely with ECA Rule 8, which is based on the following Boolean function $r_8(x_1, x_2, x_3) = \neg x_1 \wedge x_2 \wedge x_3$. As we will see, finding the expression of sensitivity

function for this rule is somewhat peculiar and require to develop specific techniques. Although the trajectory of $\lim_{n\to+\infty} \mu_s(F_{8,n})$ is different from the low-sensitive rules, it also tends to 0 for rule 8.

Remark A.2. For any $x_1, x_3 \in \{0,1\}$, it holds that $r_8(x_1, 0, x_3) = 0$. Hence, for any update schedule a cell that is in state 0 will remain in state 0 forever.

We will first see in Lemma A.19 that as soon as two update schedules differ on the labeling of an arc $(i, i-1)$, then the two dynamics are different. Then, given two update schedules Δ, Δ' such that $lab_\Delta((i,i-1)) = lab_{\Delta'}((i,i-1))$ for all $i \in [\![n]\!]$, Lemmas A.20 and A.21 will respectively give sufficient and necessary conditions for the equality of the two dynamics.

Lemma A.19. *Consider two update schedules $\Delta, \Delta' \in P_n$ for $n \geq 3$. If there exists $i \in [\![n]\!]$ such that $lab_\Delta((i,i-1)) \neq lab_{\Delta'}((i,i-1))$, then $D_{F_{8,n}^{(\Delta)}} \neq D_{F_{8,n}^{(\Delta')}}$.*

Proof. Choose $n \geq 3$ and fix some $i \in [\![n]\!]$. Without loss of generality, assume that $lab_\Delta((i,i-1)) = \oplus$ and $lab_{\Delta'}((i,i-1)) = \ominus$ and take $x \in \{0,1\}^n$ such that $(x_{i-2}, x_{i-1}, x_i) = (0,1,1)$. Cell $i-2$ will not change its state, hence when it is time for cell $i-1$ to be updated it will have a 0 at its left (cell $i-2$) in both cases. For Δ, when cell $i-1$ is to be updated, its neighbourhood will be $(0,1,1)$, and its state will become 1 after the iteration. As for Δ', when cell i is to be updated, cell $i-1$ is still in state 1, therefore its state will become 0 and when its time for cell $i-1$ to be updated, it will have a 0 at its right (cell i) and its state will become 0 after the iteration. We conclude that $f_8^{(\Delta)}(x)_{i-1} \neq f_8^{(\Delta')}(x)_{i-1}$ and the result follows. □

Now consider two update schedules Δ, Δ' whose labelings are equal on all *counter-clockwise* arcs (*i.e.* of the form $(i, i-1)$). Lemma A.20 states that, if Δ and Δ' differ only on one arc $(i-1, i)$ such that $lab_\Delta((i+1,i)) = lab_{\Delta'}((i+1,i)) = \ominus$, then the two dynamics are identical. By transitivity, if there are more differences but only on arcs of this form, then the dynamics are also identical.

Lemma A.20. *Suppose Δ and Δ' are two update schedules over a configuration of length $n \geq 3$ and there is $i \in [\![n]\!]$ such that*

- $lab_\Delta((i+1,i)) = lab_{\Delta'}((i+1,i)) = \ominus$;
- $lab_\Delta((i-1,i)) \neq lab_{\Delta'}((i-1,i))$;
- $lab_\Delta((j_1,j_2)) = lab_{\Delta'}((j_1,j_2))$, *for all* $(j_1,j_2) \neq (i-1,i)$.

Then $D_{F_{8,n}^{(\Delta)}} = D_{F_{8,n}^{(\Delta')}}$.

Proof. Fix $n \geq 3$ and choose $i \in [\![n]\!]$. Without loss of generality, suppose that $lab_\Delta((i-1,i)) = \oplus$ and $lab_{\Delta'}((i-1,i)) = \ominus$. By Theorem A.2 and the fact that $lab_\Delta((i+1,i)) = lab_{\Delta'}((i+1,i)) = \ominus$, it follows that $lab_\Delta((i,i+1)) = lab_{\Delta'}((i,i+1)) = \oplus$, otherwise a forbidden cycle of length two is created. See Figure A.29 for an illustration of the setting.
The two update schedules Δ and Δ' are very similar. Indeed, for any cell $j \in [\![n]\!] \setminus \{i\}$ the chain of influences are identical, *i.e.* $\overleftarrow{\mathtt{d}}_\Delta(j) = \overleftarrow{\mathtt{d}}_{\Delta'}(j)$ and $\overrightarrow{\mathtt{d}}_\Delta(j) = \overrightarrow{\mathtt{d}}_{\Delta'}(j)$. We deduce from Lemma A.4 that for any configuration $x \in \{0,1\}^n$ and any $j \neq i$ the images under update schedules Δ and Δ', *i.e.* $f_8^{(\Delta)}(x)_j = f_8^{(\Delta')}(x)_j$. As a consequence, it only remains to consider

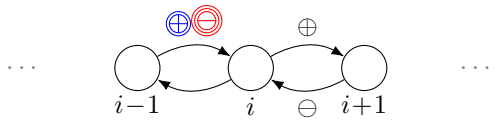


Figure A.29 – Illustration of lab_Δ (blue double-circled label) and $lab_{\Delta'}$ (red triple-circled label) in Lemma A.20. All other labels are equal (the label of arc $(i+1,i)$ is $\ominus$ in both update schedules by hypothesis).

cell i. Let $x \in \{0,1\}^n$ be any configuration (if $n \leq 2$ then $i-1 = i+1$, but $lab_\Delta((i-1,i)) = \oplus$ whereas $lab_\Delta((i+1,i)) = \ominus$).
By Remark A.2, if $x_i = 0$, then $f_8^{(\Delta)}(x)_i = f_8^{(\Delta')}(x)_i = 0$. Now suppose $x_i = 1$. Since $lab_\Delta((i,i+1)) = lab_{\Delta'}((i,i+1)) = \oplus$, by the time cell $i+1$ is updated, there is a 1 at its left (cell i) in both cases, hence $f_8^{(\Delta)}(x)_{i+1} = f_8^{(\Delta')}(x)_{i+1} = 0$. Then, when cell i is updated in both cases, there will be a 0 at its right (cell $i+1$), therefore $f_8^{(\Delta)}(x)_i = f_8^{(\Delta')}(x)_i = 0$.
We conclude that for all $x \in \{0,1\}^n$ and all $j \in [\![n]\!]$ we have $f_8^{(\Delta)}(x)_j = f_8^{(\Delta')}(x)_j$, *i.e.* $D_{F_{8,n}^{(\Delta)}} = D_{F_{8,n}^{(\Delta')}}$. □

Lemma A.21 states that, as soon as Δ and Δ' differ on arcs of the form $(i-1,i)$ such that $lab_\Delta((i+1,i)) = lab_{\Delta'}((i+1,i)) = \oplus$, then the two dynamics are different (remark that in this case we must have $lab_\Delta((i,i-1)) = lab_{\Delta'}((i,i-1)) = \oplus$ otherwise one of Δ or Δ' has an invalid cycle of length two between the nodes $i-1$ and i). This lemma can be applied if at least one cell of the configuration contains the pattern.

Lemma A.21. *For $n \geq 5$, consider two update schedules $\Delta, \Delta' \in P_n$ such that $lab_\Delta((j,j-1)) = lab_{\Delta'}((j,j-1))$, for all $j \in [\![n]\!]$. If there exists at least one cell $i \in [\![n]\!]$ such that*

- $lab_\Delta((i+1,i)) = lab_{\Delta'}((i+1,i)) = \oplus$;
- $lab_\Delta((i-1,i)) \neq lab_{\Delta'}((i-1,i))$;

then $D_{F_{8,n}^{(\Delta)}} \neq D_{F_{8,n}^{(\Delta')}}$.

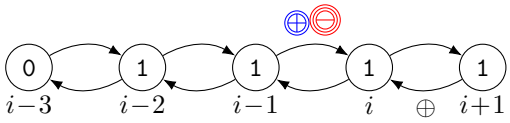


Figure A.30 – Illustration of lab_Δ (blue double-circled label) and $lab_{\Delta'}$ (red triple-circled label) in Lemma A.21. Other labels on arcs of the form $(j-1,j)$ are *a priori* unknown (they may be equal or different in Δ and Δ'), however, labels on arcs of the form $(j,j-1)$ are equal by hypothesis. States inside the nodes correspond to a configuration x such that the image of cell i under update schedule Δ is 0, whereas under update schedule Δ' it is 1.

Proof. Choose n, Δ and Δ' as in the hypothesis. Without loss of generality, assume that $lab_\Delta((i-1,i)) = \oplus$ and $lab_{\Delta'}((i-1,i)) = \ominus$ for $i \in [\![n]\!]$. See Figure A.30 for an illustration of the setting. We are going to construct a configuration $x \in \{0,1\}^n$ such that $f_8^{(\Delta)}(x)_i = 0$ whereas $f_8^{(\Delta')}(x)_i = 1$, *i.e.* such that the two dynamics differ in the image of cell i.
The construction of $x \in \{0,1\}^n$ only requires to set the pattern $(x_{i-3}, x_{i-2}, x_{i-1}, x_i, x_{i+1}) = (0,1,1,1,1)$. Regarding Δ, from the $\oplus$-labels of arcs $(i-1,i)$ and $(i+1,i)$ we have $f_8^{(\Delta)}(x)_i = r_8(x_{i-1}, x_i, x_{i+1}) = r_8(1,1,1) = 0$. Regarding Δ', let us deduce by denoting y the image of x (*i.e.* $y_i = f_8^{(\Delta')}(x)_i$) that whatever the value of $\overleftarrow{\mathtt{d}}_{\Delta'}(i)$ we have $f_8^{(\Delta')}(x)_i = 1$ (*i.e.* $y_i = 1$).

- If $\overleftarrow{\mathtt{d}}_{\Delta'}(i) = 2$ then cell $i-1$ is first updated and then cell i,
  - $y_{i-1} = r_8(x_{i-2}, x_{i-1}, x_i) = r_8(1,1,1) = 0$,
  - $y_i = r_8(y_{i-1}, x_i, x_{i+1}) = r_8(0,1,1) = 1$.
- if $\overleftarrow{\mathtt{d}}_{\Delta'}(i) = 3$ then cell $i-2$ is updated then cell $i-1$ and then cell i,
  - $y_{i-2} = r_8(x_{i-3}, x_{i-2}, x_{i-1}) = r_8(0,1,1) = 1$,
  - $y_{i-1} = r_8(y_{i-2}, x_{i-1}, x_i) = r_8(1,1,1) = 0$,
  - $y_i = r_8(y_{i-1}, x_i, x_{i+1}) = r_8(0,1,1) = 1$.
- if $\overleftarrow{\mathtt{d}}_{\Delta'}(i) \geq 4$ then cell $i-3$ is updated then cell $i-2$ then cell $i-1$ and then cell i,
  - $y_{i-3} = 0$ by Remark A.2 since $x_{i-3} = 0$,
  - $y_{i-2} = r_8(y_{i-3}, x_{i-2}, x_{i-1}) = r_8(0,1,1) = 1$,
  - $y_{i-1} = r_8(y_{i-2}, x_{i-1}, x_i) = r_8(1,1,1) = 0$,
  - $y_i = r_8(y_{i-1}, x_i, x_{i+1}) = r_8(0,1,1) = 1$.

Remark that $n \geq 5$ is required by the consideration of cells $i-3$ to $i+1$ in the third case. □

For rule 8, Lemmas A.19, A.20 and A.21 completely characterise the cases when two update schedules Δ, Δ' lead to

- the same dynamics, *i.e.* $\mathcal{D}(F_{8,n}^{(\Delta)}) = \mathcal{D}(F_{8,n}^{(\Delta')})$, or
- different dynamics, *i.e.* $\mathcal{D}(F_{8,n}^{(\Delta)}) \neq \mathcal{D}(F_{8,n}^{(\Delta')})$.

Indeed, Lemma A.19 shows that counting $|\mathcal{D}(F_{8,n})|$ can be partitioned according to the word given by $lab_\Delta((i,i-1))$ for $i \in [\![n]\!]$. Then, for each labeling of the n arcs of the form $(i,i-1)$, Lemmas A.20 and A.21 provide a way of counting the number of dynamics. We first give an example of application, and then the general counting result establishing a relation to the bisection of Lucas numbers.

Example A.2 – Consider the set of non-equivalent update schedules $\Delta \in \mathcal{P}_{10}$ such that

$$(lab_\Delta((i,i-1)))_{i \in [\![10]\!]} = (\oplus, \ominus, \oplus, \oplus, \oplus, \ominus, \oplus, \ominus, \ominus, \oplus).$$

We have the following disjunction for $i \in [\![n]\!]$ (see Figure A.31):

A- if $lab((i,i-1)) = \ominus$ then $lab((i-1,i)) = \oplus$ according to Theorem A.2, else

B- if $lab((i+1,i)) = \ominus$ then $lab((i-1,i))$ does not change the dynamics according to Lemma A.20,

C- if $lab((i+1,i)) = \oplus$ then the two possibilities for $lab((i-1,i))$ each lead to different dynamics according to Lemma A.21.

Therefore, on overall, there are $2^3 = 8$ different dynamics for such update schedules.

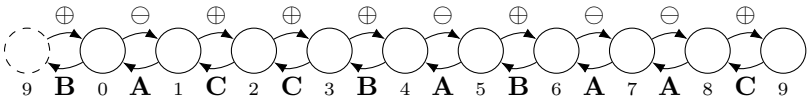


Figure A.31 – Counting the number of different dynamics for ECA rule 8 when the labeling of arcs $(i-1,i)$ for $i \in [\![10]\!]$ is $(\oplus,\ominus,\oplus,\oplus,\oplus,\ominus,\oplus,\ominus,\ominus,\oplus)$. Labels **A** are enforced to be $\oplus$ by Theorem A.2, labels **B** have no influence according to Lemma A.20, and any combination of labels **C** gives a different dynamics according to Lemma A.21.

Theorem A.22. *For any $n \geq 5$, $\mu_s(F_{8,n}) = \frac{\phi^{2n}+\phi^{-2n}-2^n}{3^n-2^{n+1}+2}$ with $\phi = \frac{1+\sqrt{5}}{2}$ the golden ratio.*

Proof. According to Lemma A.19, sets

$$\mathcal{D}_u(F_{8,n}) = \left\{D_{F^{(\Delta)}} \mid \Delta \in P_n \text{ and } \left(lab_\Delta((i,i-1))\right)_{i\in[\![n]\!]} = u\right\} \text{ for } u \in \{\oplus,\ominus\}^n$$

form a partition of $\mathcal{D}(F_{8,n})$. That is, in $\mathcal{D}_u(F_{8,n})$ the labels of arcs of the form $(i,i-1)$ for $i \in [\![n]\!]$ are fixed according to some word $u \in \{\oplus,\ominus\}^n$. Therefore we have

$$|\mathcal{D}(F_{8,n})| = \sum_{u\in\{\oplus,\ominus\}^n} |\mathcal{D}_u(F_{8,n})|.$$

Then, given some word $u \in \{\oplus,\ominus\}^n \setminus \{\oplus^n,\ominus^n\}$, according to Theorem A.2 and Lemmas A.20 and A.21 we have (see Example A.2 for details)

$$|\mathcal{D}_u(F_{8,n})| = 2^{|u|_\oplus - |u|_{\oplus\ominus}}$$

where $|u|_\oplus$ is the number of $\oplus$ in word u, and $|u|_{\oplus\ominus}$ is the number of $\oplus\ominus$-factors in word u considered periodically, *i.e.* $|u|_{\oplus\ominus} = |\{i \in [\![n]\!] \text{ such that } u_i = \oplus \text{ and } u_{i+1} = \ominus\}|$.
According to Theorem A.2, the cases $u \in \{\oplus^n,\ominus^n\}$ are particular. Indeed, for any n:

- all labels $\ominus$ (*i.e.*, $u = \ominus^n$) is an invalid cycle hence $|\mathcal{D}_{\ominus^n}(F_{8,n})| = 0$,
- all labels $\oplus$ (*i.e.*, $u = \oplus^n$) forces the labels of all arcs of the form $(i-1,i)$ for $i \in [\![n]\!]$ to be also labeled $\oplus$ otherwise a forbidden cycle is created, hence $|\mathcal{D}_{\ominus^n}(F_{8,n})| = 1$.

Given that $2^{|\ominus^n|_\oplus - |\ominus^n|_{\oplus\ominus}} = 2^0 = 1$ (instead of 0) and $2^{|\oplus^n|_\oplus - |\oplus^n|_{\oplus\ominus}} = 2^n$ (instead of 1), we deduce that

$$|\mathcal{D}(F_{8,n})| = \left(\sum_{u\in\{\oplus,\ominus\}^n} 2^{|u|_\oplus - |u|_{\oplus\ominus}}\right) - 2^n. \tag{A.16}$$

In order to study the summation term in Equation A.16, let us denote it by $S(n)$. We will consider recurrence relations according to the following partition of the set $\{\oplus,\ominus\}^n$: for $\sigma,\sigma' \in$

$\{\oplus, \ominus\}$ let $L_{\sigma\sigma'}(n)$ be the set of words beginning with label σ and ending with label σ', *i.e.* $L_{\sigma\sigma'}(n) = \{u = u_0 \dots u_{n-1} \in \{\oplus, \ominus\}^n \mid u_0 = \sigma \text{ and } u_{n-1} = \sigma'\}$. Denoting

$$S_{\sigma\sigma'}(n) = \sum_{u \in L_{\sigma\sigma'}} 2^{|u|_\oplus - |u|_{\oplus\ominus}}$$

the recurrence relations are, for all $n \geq 1$ (although Equation A.16 holds only for $n \geq 5$, the value starting from which Lemmas A.19, A.20 and A.21 hold),

- $S_{\oplus\oplus}(n+1) = 2S_{\oplus\oplus}(n) + 2S_{\oplus\ominus}(n)$,
- $S_{\oplus\ominus}(n+1) = \frac{1}{2}S_{\oplus\oplus}(n) + S_{\oplus\ominus}(n)$,
- $S_{\ominus\oplus}(n+1) = 2S_{\ominus\oplus}(n) + S_{\ominus\ominus}(n)$,
- $S_{\ominus\ominus}(n+1) = S_{\ominus\oplus}(n) + S_{\ominus\ominus}(n)$,

and we have $S(n) = S_{\oplus\oplus}(n) + S_{\oplus\ominus}(n) + S_{\ominus\oplus}(n) + S_{\ominus\ominus}(n)$. Indeed, for example regarding $S_{\oplus\oplus}(n+1)$, consider a word $u = u_0 \dots u_{n-1} \in \{\oplus, \ominus\}^n$ and the concatenation of a label $\sigma \in \{\oplus, \ominus\}$ at the end of u, then $u' = u_0 \dots u_{n-1}\sigma) \in L_{\oplus\oplus}(n+1)$ if and only if $\sigma = \oplus$ and $u_0 = \oplus$, *i.e.* $\sigma = \oplus$ and ($u \in L_{\oplus\oplus}(n)$ or $u \in L_{\oplus\ominus}(n)$). It follows that,

- if $u \in L_{\oplus\oplus}(n)$ then $|u'|_\oplus = |u|_\oplus + 1$ and $|u'|_{\oplus\ominus} = |u|_{\oplus\ominus}$,
- if $u \in L_{\oplus\ominus}(n)$ then $|u'|_\oplus = |u|_\oplus + 1$ and $|u'|_{\oplus\ominus} = |u|_{\oplus\ominus}$,

which gives the first recurrence. A similar reasoning leads to the three other recurrence relations. Also remark that by symmetry we always have $S_{\oplus\ominus}(n) = S_{\ominus\oplus}(n)$, though this fact will not be used in the coming proof.
In order to solve the recurrence, we establish a relation to known formulas by remarking that $S(n) = 3S(n-1) - S(n-2)$, which corresponds to the bisection of Fibonacci-like integer sequences (*aka* Lucas sequences):

$$\begin{aligned}
S(n) &= S_{\oplus\oplus}(n) + S_{\oplus\ominus}(n) + S_{\ominus\oplus}(n) + S_{\ominus\ominus}(n) \\
&= 2S_{\oplus\oplus}(n-1) + 2S_{\oplus\ominus}(n-1) + \frac{1}{2}S_{\oplus\oplus}(n-1) + S_{\oplus\ominus}(n-1) \\
&\quad + 2S_{\ominus\oplus}(n-1) + S_{\ominus\ominus}(n-1) + S_{\ominus\oplus}(n-1) + S_{\ominus\ominus}(n-1) \\
&= 3S_{\oplus\oplus}(n-1) + 3S_{\oplus\ominus}(n-1) + 3S_{\ominus\oplus}(n-1) + 3S_{\ominus\ominus}(n-1) \\
&\quad - \frac{1}{2}S_{\oplus\oplus}(n-1) - S_{\ominus\ominus}(n-1) \\
&= 3S(n-1) - S_{\oplus\oplus}(n-2) - S_{\oplus\ominus}(n-2) - S_{\ominus\oplus}(n-2) - S_{\ominus\ominus}(n-2) \\
&= 3S(n-1) - S(n-2).
\end{aligned}$$

Finally, since we have $S(1) = 2$ and $S(2) = 3$ we deduce that $S(n)$ is the bisection of Lucas numbers, sequence `A005248` of OEIS [The Online Encyclopedia of Integer Sequences (1996)b]. The nice closed form involving the golden ratio is a folklore adaptation of Binet's formula to Lucas numbers, and Equation A.16 gives the result. □

We conclude our consideration of ECA rule number 8 with the study of its asymptotic sensitivity (this study is trivial for other rules).

Theorem A.23. $\mu_s(F_8) = 0$.

Proof. According to the definitions and Theorem A.22, with $\phi = \frac{1+\sqrt{5}}{2}$ we have

$$\mu_s(f_8) = \lim_{n\to+\infty} \frac{\phi^{2n} + \phi^{-2n} - 2^n}{3^n - 2^{n+1} + 2}.$$

Dividing both terms of the fraction by 3^n, we observe that the denominator tends to 1, whereas the numerator tends to 0 (because $\frac{\phi^{2n}}{3^n} = (\frac{\phi}{3})^n$ and $\phi^2 < 3$). □

A.2.4 Class IV: Almost max-sensitive rules

This last class contains three ECA rules, namely 128, 160 and 162, for which the sensitivity function tends to 1. The study of sensitivity to synchronism for these rules is based on the characterisation of pairs of update schedule leading to the same dynamics. An unordered pair of update schedules $\{\Delta, \Delta'\} \in P_n \times P_n$ is *isomer for rule* α if $\Delta \not\equiv \Delta'$ but $D_{F^{(\Delta)}_{\alpha,n}} = D_{F^{(\Delta')}_{\alpha,n}}$. We will count the isomer pairs for rules 128, 160 and 162 to quantify the number of different dynamics $|\mathcal{D}(F_{\alpha,n})|$. Indeed, the sensitivity function of these rules is given by $\frac{3^n - 2^{n+1} + 2 - |I_n|}{3^n - 2^{n+1} + 2}$ where I_n is the set of isomer pairs considering n cells.
Given an update schedule $\Delta \in P_n$, let us define the *left rotation* $\sigma(\Delta)$ and the *left/right exchange* $\rho(\Delta)$,such that, $\forall i \in [\![n]\!]$, it holds that $lab_{\sigma(\Delta)}((i,j)) = lab_\Delta((i+1, j+1))$ and $lab_{\rho(\Delta)}((i,j)) = lab_\Delta((j,i))$. It is clear that if a pair of update schedules $\{\Delta, \Delta'\} \in P_n \times P_n$ is isomer then $\{\sigma(\Delta), \sigma(\Delta')\}$ is also isomer. Furthermore, when rule α is left/right symmetric (meaning that $\forall x_1, x_2, x_3 \in \{0,1\}$ we have $r_\alpha(x_1, x_2, x_3) = r_\alpha(x_3, x_2, x_1)$, which is the case of rules 128 and 162, but not 160) then $\{\rho(\Delta), \rho(\Delta')\}$ is also isomer. We say that isomer pairs in a set S are *disjoint* when no update schedule belongs to more than one pair, *i.e.*, if three update schedules $\Delta, \Delta', \Delta'' \in S$ are such that both $\{\Delta, \Delta'\}$ and $\{\Delta, \Delta''\}$ are isomer pairs then $\Delta' = \Delta''$. When it is clear from the context, we will omit to mention the rule relative to which some pairs are isomer.

ECA rule 128

The Boolean function associated with the ECA rule 128 is $r_{128}(x_1, x_2, x_3) = x_1 \wedge x_2 \wedge x_3$. Its simple definition will allow us to better illustrate the role played by isomer pairs.

Remark A.3. When $\mathtt{d}_\Delta(i) = [\![n]\!]$ for some cell i, the only possibility to get $f^{(\Delta)}_{128}(x)_i = 1$ is $x = 1^n$. However, for $x = 1^n$ we have $f^{(\Delta)}_{128}(x)_i = 1$ for any Δ.

The previous remark combined with an observation in the spirit of Lemma A.5, gives the next characterisation. Let us introduce the notation $\mathtt{d}_\Delta = \mathtt{d}_{\Delta'}$ for cases in which $\mathtt{d}_\Delta(i) = \mathtt{d}_{\Delta'}(i)$ holds in every cell $i \in [\![n]\!]$.

Lemma A.24. *For any $n \in \mathbb{N}$, choose $\Delta, \Delta' \in P_n$ such that $\Delta \not\equiv \Delta'$. Then, $\mathtt{d}_\Delta = \mathtt{d}_{\Delta'}$ if and only if $D_{F^{(\Delta)}_{128,n}} = D_{F^{(\Delta')}_{128,n}}$.*

Proof. For $n \in \{1, 2, 3\}$ we have $\mathtt{d}_\Delta(i) = \mathtt{d}_{\Delta'}(i) = [\![n]\!]$ for all $i \in [\![n]\!]$, and only one dynamics, therefore the result holds. Now, consider $n \geq 4$.

($\Rightarrow$) Assume $\mathtt{d}_\Delta = \mathtt{d}_{\Delta'}$. Given $i \in [\![n]\!]$, two cases are possible:

- $\mathtt{d}_\Delta(i) \neq [\![n]\!]$. In this case, one can deduce from $\mathtt{d}_\Delta(i) = \mathtt{d}_{\Delta'}(i)$ that $\overrightarrow{\mathtt{d}}_\Delta(i) = \overrightarrow{\mathtt{d}}_{\Delta'}(i)$ and $\overleftarrow{\mathtt{d}}_\Delta(i) = \overleftarrow{\mathtt{d}}_{\Delta'}(i)$ (see the proof of Lemma A.5). From Formula A.10 it follows $f_{128}^{(\Delta)}(x)_i = f_{128}^{(\Delta')}(x)_i$ for any $x \in \{\mathtt{0},\mathtt{1}\}^n$.
- $\mathtt{d}_\Delta(i) = \mathtt{d}_{\Delta'}(i) = [\![n]\!]$. In this case, in order to have $f_{128}^{(\Delta)}(x)_i \neq f_{128}^{(\Delta')}(x)_i$ one must have one of them equal to $\mathtt{1}$ and the other equal to $\mathtt{0}$. Without loss of generality, assume $f_{128}^{(\Delta)}(x)_i = \mathtt{1}$. From the definition of the ECA rule 128 and Formula A.10, since $\mathtt{d}_\Delta(i) = [\![n]\!]$ the only possibility is $x = \mathtt{1}^n$, but this also implies $f_{128}^{(\Delta')}(x)_i = \mathtt{1}$.

We conclude that $f_{128}^{(\Delta)}(x)_i = f_{128}^{(\Delta')}(x)_i$ for any $x \in \{\mathtt{0},\mathtt{1}\}^n$ and $i \in [\![n]\!]$, which is equivalently formulated as $D_{F_{128,n}^{(\Delta)}} = D_{F_{128,n}^{(\Delta')}}$.

($\Leftarrow$) Assume $\mathtt{d}_\Delta(i) \neq \mathtt{d}_{\Delta'}(i)$ for some $i \in [\![n]\!]$. Then, without loss of generality, there exists $j \in [\![n]\!]$ such that $j \in \mathtt{d}_\Delta(i) \setminus \mathtt{d}_{\Delta'}(i)$. The configuration x, where all the cells are in state $\mathtt{1}$ except cell j, gives (again by Formula A.10) that $f_{128}^{(\Delta)}(x)_i \neq f_{128}^{(\Delta')}(x)_i$. Indeed,

- the image of i under the update schedule Δ depends on $x_j = \mathtt{0}$, which (by the definition of the ECA rule 128) ensures that $f_{128}^{(\Delta)}(x)_i = \mathtt{0}$, and
- the image of i under the update schedule Δ' depends only on cells in state $\mathtt{1}$, which (by the definition of the ECA rule 128) ensures that $f_{128}^{(\Delta')}(x)_i = \mathtt{1}$.

As a consequence, $D_{F_{128,n}^{(\Delta)}} \neq D_{F_{128,n}^{(\Delta')}}$. □

Lemma A.24 characterises exactly the pairs of non-equivalent update schedules for which the dynamics of rule 128 differ, *i.e.*, the set of isomer pairs $\{\Delta, \Delta'\} \in P_n \times P_n$ for rule 128 such that $\Delta \not\equiv \Delta'$ but $\mathtt{d}_\Delta = \mathtt{d}_{\Delta'}$. Computing $\mu_s(F_{128,n})$ is now a combinatorial problem of computing the number of possible $\mathtt{d}_\Delta$ for $\Delta \in P_n$.

Remark A.4. Lemma A.24 does not hold for all rules, since some of them are max-sensitive even though there exist $\Delta \not\equiv \Delta'$ with $\mathtt{d}_\Delta(i) = \mathtt{d}_{\Delta'}(i)$ for all $i \in [\![n]\!]$.

We are going to prove that for any $n > 6$, there exist $10n$ disjoint isomer pairs of schedules of size n (Lemma A.27). We will first argue that isomer pairs differ in the labeling of exactly one arc (Lemma A.26), then exhibit $10n$ isomer pairs of schedules of size n (which come down to five cases up to rotation and left/right exchange) and finally argue that these pairs are disjoint. This will lead to Theorem A.28. The coming proofs will make a heavy use of the following lemma (see Figure A.32).

Lemma A.25. *For any $n \geq 4$, consider an isomer pair $\{\Delta, \Delta'\} \in P_n \times P_n$ for rule 128 such that $lab_\Delta((i+1, i)) = \oplus$ and $lab_{\Delta'}((i+1, i)) = \ominus$ for some $i \in [\![n]\!]$. It holds that, for all $j \in [\![n]\!] \setminus \{i, i+1, i+2\}$, $lab_\Delta((j, j+1)) = \ominus$ and $lab_\Delta((j+1, j)) = \oplus$.*

Proof. From Lemma A.24, we must have $\mathtt{d}_\Delta = \mathtt{d}_{\Delta'}$. Hence, in particular, $\mathtt{d}_\Delta(i) = \mathtt{d}_{\Delta'}(i)$. However, from the hypothesis on the labelings of arc $(i+1, i)$, the only possibility is that $\mathtt{d}_\Delta(i) = \mathtt{d}_{\Delta'}(i) = [\![n]\!]$. Indeed, we have $i + 2 \in \mathtt{d}_{\Delta'}(i)$, but on Δ to the right we have $\overrightarrow{\mathtt{d}}_\Delta(i) = 0$ thus for the chain of influences of cell i to contain cell $i + 2$ we must have $\overleftarrow{\mathtt{d}}_\Delta(i) \geq n - 2$,

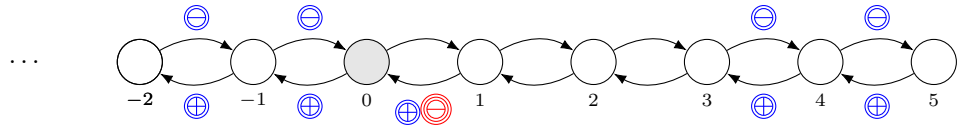


Figure A.32 – Illustration of Lemma A.25, with lab_Δ in blue double-circled labels and $lab_{\Delta'}$ in red triple-circled labels. For Δ, the hypothesis on the label of arc $(1,0)$ forces a chain of $\ominus$-labels on arcs of the form $(j, j+1)$, and a chain of $\oplus$-labels on arcs of the form $(j+1, j)$.

which corresponds to $lab_\Delta((j+1,j)) = \ominus$ for all $j \in [\![n]\!] \setminus \{i, i+1, i+2\}$. It follows that, for these j, we have $lab_\Delta((j,j+1)) = \oplus$ otherwise an invalid cycle of length two is created (Theorem A.2). □

Lemma A.26. *For any $n > 6$, if $\{\Delta, \Delta'\} \in P_n \times P_n$ is an isomer pair for rule 128 then Δ and Δ' differ on the labeling of exactly one arc.*

Proof. First, by definition of isomer pair, we have $\Delta \not\equiv \Delta'$. Hence, Δ and Δ' must differ on the labeling of at least one arc. Up to rotation and right/left exchange, let us suppose without loss of generality that $lab_\Delta((1,0)) = \oplus$ and $lab_{\Delta'}((1,0)) = \ominus$. Now, for the sake of contradiction, assume that they also differ on another arc, and consider the following cases disjunction (remark that the order of the case study is chosen so that cases make reference to previous cases).

(a) If $lab_\Delta((i,i+1)) = \oplus$ and $lab_{\Delta'}((i,i+1)) = \ominus$ for some $i \in [\![n]\!]$, then by applying Lemma A.25 to the two arcs where Δ and Δ' differ leads to a contradiction on the labeling of some arc according to Δ. Indeed, Lemma A.25 is applied to two arcs in different directions, one application leaves three arcs of the form $(j, j+1)$ not labeled $\ominus$ in Δ and three arcs of the form $(j+1, j)$ not labeled $\oplus$ in Δ, the converse for the other application, hence starting from $n = 7$ these labelings overlap in a contradictory fashion.

(b) If $lab_\Delta((i+1,i)) = \ominus$ and $lab_{\Delta'}((i+1,i)) = \oplus$ for some $i \in [\![n]\!] \setminus \{0\}$, then $i \in \{2,3\}$ otherwise there is a forbidden cycle of length two in Δ with some $\ominus$-label given by the application of Lemma A.25 to the arc $(1,0)$. However, for $i \in \{2,3\}$ the application of Lemma A.25 to the arc $(i+1,i)$ gives $lab_{\Delta'}((0,1)) = \ominus$, creating a forbidden cycle of length two in Δ'.

(c) If $lab_\Delta((i+1,i)) = \oplus$ and $lab_{\Delta'}((i+1,i)) = \ominus$ for some $i \in [\![n]\!] \setminus \{1\}$, then applying Lemma A.25 to the two arcs where Δ and Δ' differ leads to a forbidden cycle of length n in Δ (contradiction Theorem A.2). Indeed, if $i \notin \{1,2\}$ then we have $\ominus$-labels on arcs of the form $(j, j+1)$ for all $j \in [\![n]\!]$, and if $i = 2$ then the forbidden cycle contains the arc $(3,2)$ labeled $\oplus$. The case $i = 0$ is not a second difference.

(d) If $lab_\Delta((i,i+1)) = \ominus$ and $lab_{\Delta'}((i,i+1)) = \oplus$ for some $i \in [\![n]\!]$, then applying Lemma A.25 to arc $(1,0)$ gives $lab_\Delta((j+1,j)) = \oplus$ for all $j \in [\![n]\!] \setminus \{0,1,2\}$, and applying Lemma A.25 to arc $(i,i+1)$ gives $lab_{\Delta'}((j+1,j)) = \ominus$ for all $j \in [\![n]\!] \setminus \{i, i-1, i-2\}$. Starting from $n = 7$ we have $([\![n]\!] \setminus \{0,1,2\}) \cap ([\![n]\!] \setminus \{i, i-1, i-2\}) \neq \emptyset$ and, as a consequence, there is an arc $((j+1,j))$ in the case of Item (c).

(e) If $lab_\Delta((2,1)) = \oplus$ and $lab_{\Delta'}((2,1)) = \ominus$, then applying Lemma A.25 to arc $(2,1)$ gives $lab_\Delta((0,1)) = \ominus$, however, since by hypothesis $lab_{\Delta'}((1,0)) = \ominus$ we also have

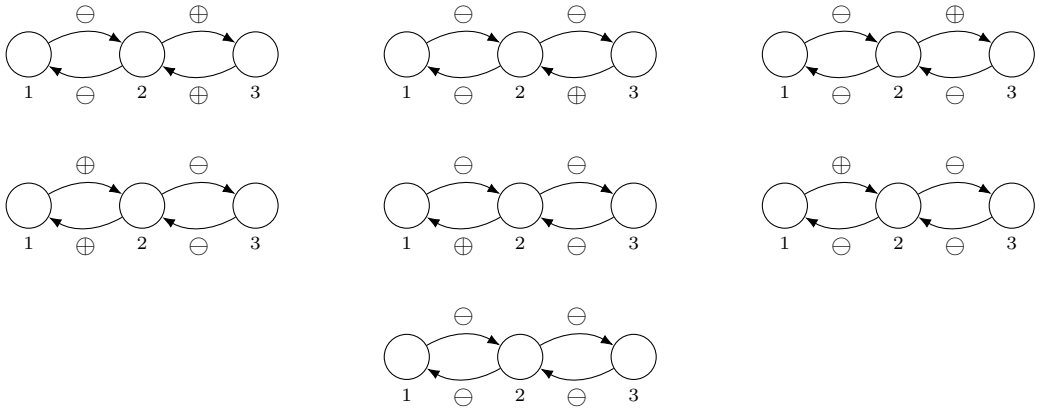


Figure A.33 – Seven labelings of arcs $(1,2)$, $(2,3)$, $(2,1)$ and $(3,2)$ giving a forbidden cycle of length two, in the proof of Lemma A.27.

$lab_{\Delta'}((0,1)) = \oplus$ otherwise there is a forbidden cycle of length two in Δ' (Theorem A.2). As a consequence, the arc $(0,1)$ is in the case of Item (d).

We conclude that in any case a second difference leads to a contradiction, either because an invalid cycle is created, or because repeated applications of Lemma A.25 give contradictory labels (both $\oplus$ and $\ominus$) to some arc for some update schedule. □

Lemma A.27. *For any $n > 6$, there exist $10n$ disjoint isomer pairs of schedules of size n for rule 128.*

Proof. Fix $n \geq 6$ and consider the set of isomer pairs $\{\Delta, \Delta'\} \in P_n \times P_n$ in which we have a difference between Δ and Δ' on the labeling of arc $(1,0)$ (with $lab_\Delta((1,0)) = \oplus$ and $lab_{\Delta'}((1,0)) = \ominus$). Lemma A.25 fixes the labels of many arcs of Δ, and from Lemma A.26 the same labels hold for Δ' since there is already a difference on arc $(1,0)$. Therefore, we have:

$$\text{for all } j \in [\![n]\!] \setminus \{0,1,2\}, lab_\Delta((j,j+1)) = lab_{\Delta'}((j,j+1)) = \ominus$$
$$\text{and } lab_\Delta((j+1,j)) = lab_{\Delta'}((j+1,j)) = \oplus.$$

Furthermore the labeling of arc $(1,0)$ is given by our hypothesis, and from Theorem A.2 (to avoid a forbidden cycle of length two in Δ) and Lemma A.26 (equality of lab_Δ and $lab_{\Delta'}$ except for the arc $(1,0)$) we also have $lab_\Delta((0,1)) = lab_{\Delta'}((0,1)) = \oplus$. As a consequence, it remains to consider 2^4 possibilities for the labelings of arcs

$$(1,2), (2,3), (2,1) \text{ and } (3,2)$$

(which are equal on Δ and Δ', again by Lemma A.26).
Among these possibilities, seven create a forbidden cycle of length two when the labels of the two arcs between cells 1 and 2, or 2 and 3, are both $\ominus$ (see Figure A.33).
Among the remaining possibilities, four create a forbidden cycle of length n in Δ, when the labels of arcs $(2,1)$ and $(3,2)$ are set to $\oplus$ (see Figure A.34).
The five remaining possibilities are presented on Figure A.35, one can easily check that they indeed correspond to isomer pairs. In fact:

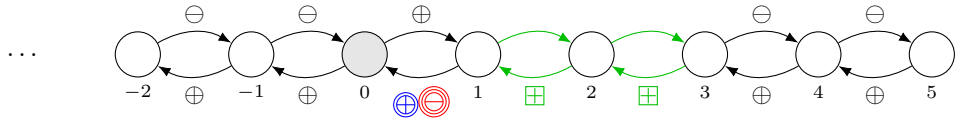


Figure A.34 – Four labelings of arcs $(1,2)$, $(2,3)$, $(2,1)$ and $(3,2)$ giving a forbidden cycle of length n in Δ (see the proof of Lemma A.27). Blue double-circled labels are involved in lab_Δ, and red triple-circled labels are involved in $lab_{\Delta'}$. Squared green labels are on the arcs on which we consider the 2^4 possibilities. The remaining labels are those in common between lab_Δ and $lab_{\Delta'}$. For any combination of $\oplus$ and $\ominus$-labels on arcs $(1,2)$ and $(2,3)$, the forbidden cycle is $0 \overset{\ominus}{\to} -1 \overset{\ominus}{\to} -2 \overset{\ominus}{\to} \dots \overset{\ominus}{\to} 5 \overset{\ominus}{\to} 4 \overset{\ominus}{\to} 3 \overset{\oplus}{\to} 2 \overset{\oplus}{\to} 1 \overset{\oplus}{\to} 0$ (recall that the orientation of $\ominus$-arcs is reversed, Theorem A.2).

- neither Δ nor Δ' contain a forbidden cycle. Hence, they are pairs of non-equivalent update schedule,
- for any $i \in [\![n]\!] \setminus \{0\}$, we have $\overleftarrow{\mathtt{d}}_\Delta(i) = \overleftarrow{\mathtt{d}}_{\Delta'}(i)$ and $\overrightarrow{\mathtt{d}}_\Delta(i) = \overrightarrow{\mathtt{d}}_{\Delta'}(i)$. Hence, $\mathtt{d}_\Delta(i) = \mathtt{d}_{\Delta'}(i)$, and for cell 0 we have $\mathtt{d}_\Delta(0) = \mathtt{d}_{\Delta'}(0) = [\![n]\!]$.

We have seen so far that there are exactly five isomer pairs with their unique difference (Lemma A.26) on arc $(1,0)$. Let us call them the five *base* pairs and denote them by $\{\Delta_b, {\Delta'}_b\}$ for $b \in [\![5]\!]$. When we consider the n rotations plus the left/right exchange (recall that rule 128 is symmetric), we obtain $10n$ pairs:

$$\rho^k(\sigma^j(\Delta_b)), \rho^k(\sigma^j({\Delta'}_b)) \text{ for } b \in [\![5]\!], j \in [\![n]\!], k \in [\![2]\!]. \tag{A.17}$$

Let us finally argue that these pairs are disjoint, *i.e.* an update schedule belongs to at most one pair.
First, one can straightforwardly check on Figure A.35 that the ten update schedules with a difference on arc $(1,0)$ are all distinct, hence the five base pairs are disjoint.
Second, the n rotations of these ten update schedules are all distinct when $n > 6$, as can be noticed from n letter words on alphabet $\{\oplus, \ominus\}$ given by

$$(lab_\Delta((i, i+1)))_{i \in [\![n]\!]} \text{ for some } \Delta.$$

Indeed, each of these words contains a unique factor $\ominus \ominus \ominus\oplus$, which allows one to identify the number of left rotations applied to some Δ_b or ${\Delta'}_b$ with $b \in [\![5]\!]$ in order to obtain Δ. As a consequence, two distinct base update schedules remain distinct when some rotation is applied to one of them.
Third, the left/right exchange of these $5n$ update schedules (base plus rotations) give $10n$ distinct update schedules, as can be noticed on the number of $\ominus$-labels on arcs of the form $(i, i+1)$ for $i \in [\![n]\!]$. Indeed, denoting

$$|\Delta|_\ominus = |\{(i, i+1) \mid i \in [\![n]\!] \text{ and } lab_\Delta((i, i+1)) = \ominus\}|,$$

we have for any $n > 6$ that $|\Delta|_\ominus > n-3$ when Δ is a base update schedule, the quantity is preserved by rotation, *i.e.* $|\sigma(\Delta)|_\ominus = |\Delta|_\ominus$, but it holds that $|\Delta|_\ominus > n-3$ if and only if $|\rho(\Delta)|_\ominus < n-3$. As a consequence, two distinct update schedules (among the $5n$ update schedules $\sigma^j(\Delta_b), \sigma^j({\Delta'}_b)$ for $b \in [\![5]\!]$ and $j \in [\![n]\!]$) remain distinct when the left/right exchange is

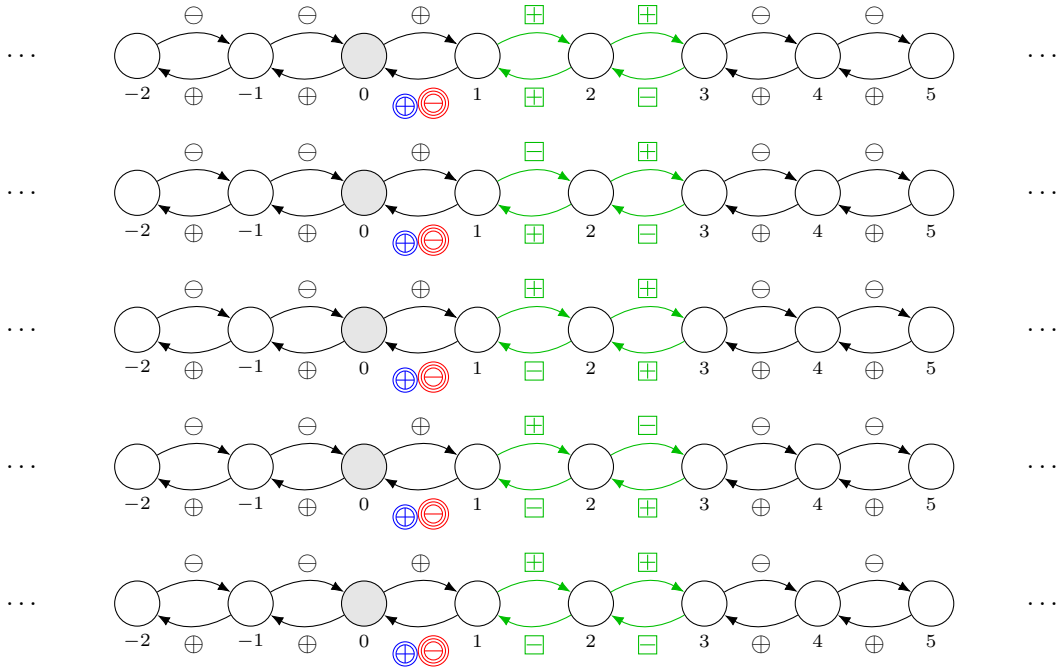


Figure A.35 – Five base isomer pairs $\{\Delta, \Delta'\} \in P_n \times P_n$ for rule 128 (see the proof of Lemma A.27). Blue double-circled labels are involved in lab_Δ, and red triple-circled labels are involved in $lab_{\Delta'}$. Squared green labels are on the arcs on which we consider the five remaining possibilities. The remaining labels are those common between lab_Δ and $lab_{\Delta'}$.

applied to one of them. When the left/right exchange is applied to both of them then the situation is symmetric to the previous considerations.
We conclude that the $10n$ pairs given by Formula A.17 are isomer and disjoint. □

As a consequence of Lemma A.27, we have $|\{\mathtt{d}_\Delta \mid \Delta \in P_n\}| = 3^n - 2^{n+1} - 10n + 2$ for any $n > 6$, and the result follows from Lemma A.24.

Theorem A.28. *For any $n > 6$, $\mu_s(F_{128,n}) = \frac{3^n - 2^{n+1} - 10n + 2}{3^n - 2^{n+1} + 2}$.*

ECA rule 162

The ECA rule 162 is expressed as the Boolean function $r_{162}(x_1, x_2, x_3) = (x_1 \vee \neg x_2) \wedge x_3$. Let F be a shorthand for $F_{162,n}$ when the context is clear.
The structure of the reasoning is to first prove that for any isomer pair $\{\Delta, \Delta'\} \in P_n \times P_n$ for rule 162, the labelings of arcs of the form $(i+1, i)$ for all $i \in [\![n]\!]$ are identical in Δ and Δ' (Lemma A.29). Second, given a difference on the labels of some arc $(i, i+1)$, we will prove that it forces all other labels both in Δ and in Δ' (Lemma A.30). Third, for the remaining case, we will prove that it is indeed an isomer pair, thus generating n disjoint isomer pairs by rotation (for any $n \geq 5$), leading to Theorem A.31.

Lemma A.29. *For any $n \geq 2$, if $\{\Delta, \Delta'\} \in P_n \times P_n$ is an isomer pair for rule 162, then for all $i \in [\![n]\!]$ we have $lab_\Delta((i+1, i)) = lab_{\Delta'}((i+1, i))$.*

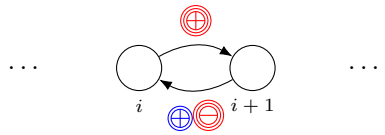


Figure A.36 – Illustration of the setting for the contradiction in Lemma A.29. Blue double-circled labels are part of lab_Δ, and red triple-circled labels are part of $lab_{\Delta'}$.

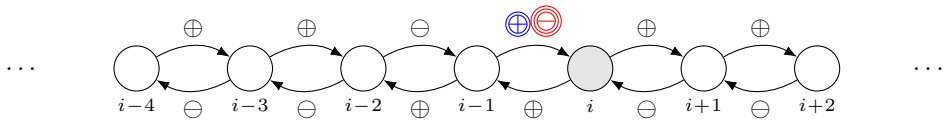


Figure A.37 – The isomer pair $\{\Delta, \Delta'\} \in P_n \times P_n$ for rule 162 in Lemma A.30, with $lab_\Delta((i-1,i)) = \oplus$ and $lab_{\Delta'}((i-1,i)) = \ominus$ for some $i \in [\![n]\!]$. Blue double-circled labels are part of lab_Δ, and red triple-circled labels are part of $lab_{\Delta'}$. The remaining labels are those in common between lab_Δ and $lab_{\Delta'}$.

Proof. By contradiction, assume that there exists $i \in [\![n]\!]$ such that, without loss of generality, $lab_\Delta((i+1,i)) = \oplus$ whereas $lab_{\Delta'}((i+1,i)) = \ominus$. This implies $lab_{\Delta'}((i,i+1)) = \oplus$ otherwise there is a forbidden cycle of length two in Δ' (Theorem A.2). See Figure A.36 for an illustration of the setting.
Consider some $x \in \{0,1\}^n$ with $x_i = 0$ and $x_{i+1} = 1$ (this require $n \geq 2$)). From our knowledge of Δ we have for some unknown $y_{i-1} \in \{0,1\}$ that

$$f^{(\Delta)}(x)_i = r_{162}(y_{i-1}, x_i, x_{i+1}) = r_{162}(y_{i-1}, 0, 1) = 1.$$

From our knowledge of Δ' we have for some unknown $y_{i-1}, y_{i+2} \in \{0,1\}$ that

$$\begin{aligned} f^{(\Delta')}(x)_i &= r_{162}(y_{i-1}, x_i, r_{162}(x_i, x_{i+1}, y_{i+2})) = r_{162}(y_{i-1}, 0, r_{162}(0, 1, y_{i+1})) \\ &= r_{162}(y_{i-1}, 0, 0) = 0. \end{aligned}$$

Thus $f^{(\Delta)}(x)_i \neq f^{(\Delta')}(x)_i$, a contradiction to the fact that $\{\Delta, \Delta'\}$ is an isomer pair. □

From Lemma A.29 and the fact that $\Delta \not\equiv \Delta'$, we will now consider an isomer pair with a difference on some arc $(i, i+1)$ for $i \in [\![n]\!]$, and prove that this first difference enforces all the other labels (both in Δ and in Δ').

Lemma A.30. *For any $n \geq 3$, there is a unique isomer pair $\{\Delta, \Delta'\} \in P_n \times P_n$ for rule 162 with $lab_\Delta((i-1,i)) \neq lab_{\Delta'}((i-1,i))$ for some $i \in [\![n]\!]$, and its labels are those depicted on Figure A.37.*

Proof. Without loss of generality, as on Figure A.37, assume that $lab_\Delta((i-1,i)) = \oplus$ and $lab_{\Delta'}((i-1,i)) = \ominus$. We deduce that $lab_{\Delta'}((i,i-1)) = \oplus$ otherwise there is a forbidden cycle of length two in Δ' (Theorem A.2) and from Lemma A.29 it follows that we also have $lab_\Delta((i,i-1)) = \oplus$.
We are going to prove that this forces all the other labels of Δ and Δ', *i.e.* there is a unique such isomer pair. From the hypothesis that $\{\Delta, \Delta'\}$ is an isomer pair, we will use the fact that for all $x \in \{0,1\}^n$ and for all $j \in [\![n]\!]$ we have $f^{(\Delta)}(x)_j = f^{(\Delta')}(x)_j$.

From Lemma A.29, we deduce that at the time cell i is updated, the state of its right neighbour (cell $i+1$) is identical under update schedules Δ and Δ'. Let us denote by $y_{i+1} \in \{0,1\}$ this state for the rest of this proof.
By contradiction, assume that it is possible to have some configuration $x \in \{0,1\}^n$ such that

$$x_{i-1} = 0 \text{ and } x_i = 1 \text{ and } y_{i+1} = 1 \tag{A.18}$$

(this requires $n \geq 3$). In this case we have

$$f^{(\Delta)}(x)_i = r_{162}(x_{i-1}, x_i, y_{i+1}) = r_{162}(0,1,1) = 0$$

but for some unknown $y_{i-2} \in \{0,1\}^n$ we always have

$$\begin{aligned} f^{(\Delta')}(x)_i &= r_{162}(r_{162}(y_{i-2}, x_{i-1}, x_i), x_i, y_{i+1}) = r_{162}(r_{162}(y_{i-2}, 0, 1), 1, 1) \\ &= r_{162}(1,1,1) = 1 \end{aligned}$$

i.e. $f^{(\Delta)}(x)_i \neq f^{(\Delta')}(x)_i$, which contradicts the hypothesis that $\{\Delta, \Delta'\}$ is an isomer pair. We conclude that it must be impossible to have simultaneously $x_{i-1} = 0$, $x_i = 1$ and $y_{i+1} = 1$. This hints at the fact that the value of $\overrightarrow{\mathtt{d}}_\Delta(i) = \overrightarrow{\mathtt{d}}_{\Delta'}(i)$ must be close to n so that the constraints on x_{i-1} and x_i make it impossible to obtain $y_{i+1} = 1$ when updating the chain of influence to the right of cell i. This is what we are going to prove formally, via the following case disjunction.

- If $\overrightarrow{\mathtt{d}}_\Delta(i) = \overrightarrow{\mathtt{d}}_{\Delta'}(i) < n-1$ then consider $x \in \{0,1\}^n$ with $x_{i-1} = 0$ and $x_i = x_{i+1} = \cdots = x_{i+\overrightarrow{\mathtt{d}}_\Delta(i)} = 1$. From our current hypothesis on $\overrightarrow{\mathtt{d}}_\Delta(i)$ and for $n \geq 3$, such a configuration exists. We deduce from the definition of rule 162 that the updates (in this order, both in Δ and Δ') of cells $i + \overrightarrow{\mathtt{d}}_\Delta(i), i + \overrightarrow{\mathtt{d}}_\Delta(i) - 1, \ldots, i+1$ all give state 1, *i.e.* in particular $y_{i+1} = 1$, leading to a contradiction as developed from Equation A.18.
- If $\overrightarrow{\mathtt{d}}_\Delta(i) = \overrightarrow{\mathtt{d}}_{\Delta'}(i) \geq n$ then there is a forbidden cycle of length n in Δ:

  $$i \overset{\ominus}{\to} i+1 \overset{\ominus}{\to} \ldots \overset{\ominus}{\to} i-2 \overset{\ominus}{\to} i-1 \overset{\oplus}{\to} i \tag{A.19}$$

  (recall that the orientation of $\ominus$-arcs is reversed, see Theorem A.2). As a consequence, we discard this case.
- If $\overrightarrow{\mathtt{d}}_\Delta(i) = \overrightarrow{\mathtt{d}}_{\Delta'}(i) = n-1$ then it means that we have $lab_\Delta((j+1,j)) = lab_{\Delta'}((j+1,j)) = \ominus$ for all $j \in [\![n]\!] \setminus \{i-2, i-1\}$, and $lab_\Delta((i-1,i-2)) = lab_{\Delta'}((i-1,i-2)) = \oplus$, but also $lab_\Delta((j,j+1)) = lab_{\Delta'}((j,j+1)) = \oplus$ for all $j \in [\![n]\!] \setminus \{i-2, i-1\}$ from Theorem A.2.
  Therefore it only remains to consider the labels of arc $(i-2, i-1)$ in schedules Δ and Δ'. To avoid a forbidden cycle of length n in Δ, similar to Equation A.19 with $i-2 \overset{\oplus}{\to} i-1$, we need to set $lab_\Delta((i-2,i-1)) = \ominus$. It only remains to consider $lab_{\Delta'}((i-2,i-1))$. Suppose for the contradiction that $lab_{\Delta'}((i-2,i-1)) = \oplus$, then similarly to our previous reasoning, for some $x \in \{0,1\}^n$ with $x_{i-2} = 0$, $x_{i-1} = 1$ and $x_i = 1$, we have for some unknown $y_{i-3} \in \{0,1\}$ that

  $$\begin{aligned} f^{(\Delta)}(x)_{i-1} &= r_{162}(r_{162}(y_{i-3}, x_{i-2}, x_{i-1}), x_{i-1}, x_i) = r_{162}(r_{162}(y_{i-3}, 0, 1), 1, 1) \\ &= r_{162}(1,1,1) = 1 \end{aligned}$$

  whereas

  $$f^{(\Delta)}(x)_{i-1} = r_{162}(x_{i-2}, x_{i-1}, x_i) = r_{162}(0,1,1) = 0$$

  thus $f^{(\Delta)}(x)_{i-1} \neq f^{(\Delta')}(x)_{i-1}$, contradicting the fact that $\{\Delta, \Delta'\}$ is an isomer pair.

We conclude that there is only one remaining possible isomer pair with a difference on the labelings of arc $(i-1, i)$, and that it is the one given on Figure A.37.
Let us finally prove that this is indeed an isomer pair. One easily checks on Figure A.37 that the update schedules of this pair have no forbidden cycle and are non-equivalent. For any $j \in [\![n]\!] \setminus \{i\}$, we have $\overleftarrow{\mathtt{d}}_{\Delta}(j) = \overleftarrow{\mathtt{d}}_{\Delta'}(j)$ and $\overrightarrow{\mathtt{d}}_{\Delta}(j) = \overrightarrow{\mathtt{d}}_{\Delta'}(j)$, *i.e.* the chain of influences are identical hence, for all $x \in \{\mathtt{0},\mathtt{1}\}^n$, we have $f^{(\Delta)}(x)_j = f^{(\Delta')}(x)_j$.
Regarding cell i, we have $\overrightarrow{\mathtt{d}}_{\Delta}(i) = \overrightarrow{\mathtt{d}}_{\Delta'}(i)$, meaning that at the time cell i is updated, its right neighbour (cell $i+1$) will be in the same state (denoted by y_{i+1}) in both update schedules. Given some $x \in \{\mathtt{0},\mathtt{1}\}^n$, we proceed to a case disjunction.

- If $y_{i+1} = \mathtt{0}$ then

  $$f^{(\Delta)}(x)_i = r_{162}(x_{i-1}, x_i, y_{i+1}) = r_{162}(x_{i-1}, x_i, \mathtt{0}) = \mathtt{0}$$

  and for some unknown $y_{i-1} \in \{\mathtt{0},\mathtt{1}\}$ we have

  $$f^{(\Delta')}(x)_i = r_{162}(y_{i-1}, x_i, y_{i+1}) = r_{162}(y_{i-1}, x_i, \mathtt{0}) = \mathtt{0}$$

  therefore we conclude $f^{(\Delta)}(x)_i = f^{(\Delta')}(x)_i$.

- If $y_{i+1} = \mathtt{1}$ then, from the reasoning we have just made above, and since cell $i+2$ is updated prior to cell $i+1$, we deduce that cell $i+2$ is updated to state $\mathtt{1}$, otherwise cell $i+1$ would be updated to state $\mathtt{0}$ (in both Δ and Δ', contradicting our last hypothesis that $y_{i+1} = \mathtt{1}$). This applies to cell $i+3$, *etc.*, until cell $i-2$ (which must also be updated to state $\mathtt{1}$). Finally, cell $i-1$ must be in state $\mathtt{1}$, *i.e.* $x_{i-1} = \mathtt{1}$.
  We deduce from $y_{i+1} = \mathtt{1}$ and $x_{i-1} = \mathtt{1}$ that

  $$f^{(\Delta)}(x)_i = r_{162}(x_{i-1}, x_i, y_{i+1}) = r_{162}(\mathtt{1}, x_i, \mathtt{1}) = \mathtt{1}.$$

  Regarding cell i in the update schedule Δ', we proceed to a last case disjunction.

  - If $x_i = \mathtt{0}$ then for some unknown $y_{i-1} \in \{\mathtt{0},\mathtt{1}\}$ we have

    $$f^{(\Delta')}(x)_i = r_{162}(y_{i-1}, x_i, y_{i+1}) = r_{162}(y_{i-1}, \mathtt{0}, \mathtt{1}) = \mathtt{1}$$

    and we conclude $f^{(\Delta)}(x)_i = f^{(\Delta')}(x)_i$.
  - If $x_i = \mathtt{1}$ then we can use our prior deduction that cell $i-2$ is updated to state $\mathtt{1}$, therefore

    $$\begin{aligned} f^{(\Delta')}(x)_i &= r_{162}(r_{162}(\mathtt{1}, x_{i-1}, x_i), x_i, y_{i+1}) = r_{162}(r_{162}(\mathtt{1},\mathtt{1},\mathtt{1}),\mathtt{1},\mathtt{1}) \\ &= r_{162}(\mathtt{1},\mathtt{1},\mathtt{1}) = \mathtt{1} \end{aligned}$$

    and we also conclude $f^{(\Delta)}(x)_i = f^{(\Delta')}(x)_i$ in this ultimate case.

We have seen that for any $x \in \{\mathtt{0},\mathtt{1}\}^n$, $F^{(\Delta)}(x) = F^{(\Delta')}(x)$. Thus, $\{\Delta, \Delta'\}$ is an isomer pair, and, from the first part of this proof, it is unique. □

Theorem A.31. *For any $n \geq 3$, $\mu_s(F_{162,n}) = \frac{3^n - 2^{n+1} - n + 2}{3^n - 2^{n+1} + 2}$.*

Proof. From Lemmas A.29 and A.30 there are n pairs of isomer pairs for rule 162 (no pair with a difference on the label of an arc of the form $(i+1,i)$ for some $i \in [\![n]\!]$ by Lemma A.29, and exactly one pair with a difference on arc $(i,i+1)$ for each $i \in [\![n]\!]$ by Lemma A.30). Denoting $\{\Delta,\Delta'\}$ the isomer pair given by Lemma A.30 with a difference on the arc $(0,1)$, the n isomer pairs are $\{\sigma^j(\Delta),\sigma^j(\Delta')\}$ for $j \in [\![n]\!]$. Lemmas A.29 and A.30 hold for any $n \geq 3$, and for any such n one easily checks by considering the word formed by the labels of arcs $(i-1,i)$ for $i \in [\![n]\!]$ (this word is identical for both schedules of each isomer pair, by Lemma A.29) that these pairs are disjoint: these words contain exactly one factor $\oplus\oplus$ whose position differs for any rotation of Δ,Δ'. It follows that among the $3^n - 2^{n+1} + 2$ non-equivalent update schedules, we have $\mathcal{D}(F_{162,n}) = 3^n - 2^{n+1} + 2 - n$, as stated. □

ECA rule 160

The ECA rule 160 is somewhat similar to the ECA rule 128. Indeed, it is based on the Boolean function $r_{160}(x_1,x_2,x_3) = x_1 \wedge x_3$.

Remark A.5. It is clear from the definition of r_{160} that for any update schedule Δ and any configuration $x \in \{0,1\}^n$ such that $x_i = x_{i+1} = 0$ (or $x_{i-1} = x_i = 0$) for some $i \in [\![n]\!]$, it holds $f_{160}^{(\Delta)}(x)_i = 0$.

We are going to adopt a reasoning analogous to rule 128 for the study of the sensitivity to synchronism of rule 160. Lemma A.32 will be the first stone showing that as soon as two update schedules form an isomer pair for rule 160, the position of their difference enforces the labels of many other arcs. Then, Lemma A.33 will use applications of Lemma A.32, according to some carefully crafted case disjunction, in order to prove that an isomer pair with more than one difference among the two update schedules (*i.e.* differences on the labels of at least two arcs) is contradictory. Finally, Lemma A.34 will use these previous results to characterise exactly the isomer pairs of update schedules for rule 160, which comes down to six disjoint base isomer pairs, leading to $12n$ isomer pairs (when considering left/right exchange and rotations). This will give Theorem A.35.

Lemma A.32. *For any $n > 4$, consider an isomer pair $\{\Delta,\Delta'\} \in P_n \times P_n$ for rule 160 such that $lab_\Delta((i+1,i)) = \oplus$ and $lab_{\Delta'}((i+1,i)) = \ominus$ for some $i \in [\![n]\!]$. For all $j \in [\![n]\!] \setminus \{i,i+1,i+2,i+3\}$, it holds that $lab_\Delta((j,j+1)) = \ominus$, $lab_\Delta((j+1,j)) = \oplus$ and also $lab_{\Delta'}((i+2,i+1)) = \ominus$, $lab_{\Delta'}((i+1,i+2)) = \oplus$.*

Proof. Let us prove that, with the hypothesis of the statement, we must have $\overleftarrow{\mathtt{d}}_\Delta(i) \geq n-4$ (which implies the $\ominus$-labels on Δ) and also $lab_{\Delta'}((i+2,i+1)) = \ominus$. The complete result follows by application of Theorem A.2 to get the $\oplus$-labels (in order to avoid any forbidden cycle of length two).

For the first part, if $\overleftarrow{\mathtt{d}}_\Delta(i) < n-3$ then we can construct the following configuration $x \in \{0,1\}^n$ without a contradiction on the states of cells $i - \overleftarrow{\mathtt{d}}_\Delta(i)$ and $i+3$:

- $x_{i+2} = x_{i+3} = 0$
- $x_i = x_{i-1} = \cdots = x_{i-\overleftarrow{\mathtt{d}}_\Delta(i)} = 1$.

This requires $n \geq 5$, see Figure A.38 for an illustration. Regarding Δ', it follows from Remark A.5 that cell $i+2$ remains in state 0, and as a consequence, regardless of the label of arc $(i+2,i+1)$,

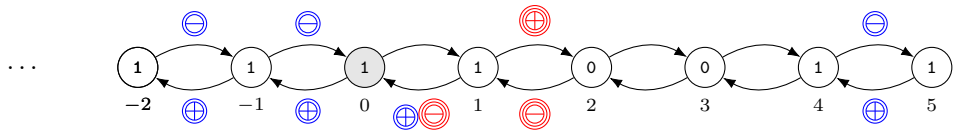


Figure A.38 – Illustration of Lemma A.32 with lab_Δ (blue double-circled labels) and $lab_{\Delta'}$ (red triple-circled label). The hypothesis on the label of arc $(1,0)$ implies: many $\ominus$-labels on arcs of the form $(j, j+1)$ (for Δ), $\oplus$-labels on arcs of the form $(j+1, j)$ (for Δ), and $\ominus$-labels on arcs $(2,1)$ and $(1,2)$ (for Δ'). Inside the cells are depicted the states corresponding to a contradictory configuration (having different images on cell 0) when $\overleftarrow{\mathtt{d}}_\Delta(0) \leq n-4$.

cell $i+1$ is updated to state $\mathtt{0}$, then so is i. However, in Δ, we have $x_j = \mathtt{1}$ for all $j \in \mathtt{d}_\Delta(i)$, *i.e.* cell i depends only on cells in state $\mathtt{1}$, and we deduce that it is updated to state $\mathtt{1}$. Thus $f_{160}^{(\Delta)}(x)_i \neq f_{160}^{(\Delta')}(x)_i$, a contradiction to the fact that $\{\Delta, \Delta'\}$ is an isomer pair.

For the second part, suppose for the contradiction that $lab_{\Delta'}((i+2, i+1) = \oplus$, and consider the configuration $x \in \{\mathtt{0},\mathtt{1}\}^n$ with $x_{i+1} = \mathtt{0}$ and state $\mathtt{1}$ in all other cells. In Δ, at time cell 0 is updated it has a state $\mathtt{0}$ on its right (cell $i+1$, not yet updated), and $f_{160}^{(\Delta)}(x)_i = \mathtt{0}$. In Δ', cell $i+1$ is updated prior to its left and right neighbours (from Theorem A.2 again we have $lab_{\Delta'}((i, i+1)) = \oplus$) thus it goes to state $\mathtt{1}$. We can deduce from this that all the cells will go to state $\mathtt{1}$ because they all have two neighbours in state $\mathtt{1}$ at the time they are updated. Therefore, in particular $f_{160}^{(\Delta')}(x)_i = \mathtt{1}$, again a contradiction. □

Let us recall that rule 160 is symmetric, therefore Lemma A.32 also applies with a left/right exchange.

Lemma A.33. *For any $n > 8$, if $\{\Delta, \Delta'\} \in P_n \times P_n$ is an isomer pair for rule 160 then Δ and Δ' differ on the labeling of exactly one arc.*

Proof. Up to rotation and right/left exchange, let us suppose without loss of generality that $lab_\Delta((1,0)) = \oplus$ and $lab_{\Delta'}((1,0)) = \ominus$. Now, for the sake of contradiction, assume that they also differ on another arc, and consider the following cases disjunction (remark that the order of the case study is chosen so that cases make reference to previous cases).

(a) If $lab_\Delta((i, i+1)) = \oplus$ and $lab_{\Delta'}((i, i+1)) = \ominus$ for some $i \in [\![n]\!]$, then according to Lemma A.32 one obtains a contradiction on the labeling of some arc according to Δ. Indeed, Lemma A.32 is applied twice centred on the two arcs with different labels, one application leaves four arcs of the form $(j, j+1)$ not labeled $\ominus$ in Δ and three arcs of the form $(j+1, j)$ not labeled $\oplus$ in Δ, the converse for the other application, hence starting from $n = 8$ these labelings overlap in a contradictory fashion.

(b) If $lab_\Delta((i+1, i)) = \ominus$ and $lab_{\Delta'}((i+1, i)) = \oplus$ for some $i \in [\![n]\!] \setminus \{0\}$, then $i \in \{1, 2, 3\}$ otherwise there is a forbidden cycle of length two in Δ with some $\ominus$-label given by the labeling of Lemma A.32 with a difference on the arc $(1,0)$. However, for $i \in \{2, 3, 4\}$, the application of Lemma A.32 centred in the arc $(i+1, i)$ gives $lab_{\Delta'}((0,1)) = \ominus$, creating a forbidden cycle of length two in Δ'.

(c) If $lab_\Delta((i+1, i)) = \oplus$ and $lab_{\Delta'}((i+1, i)) = \ominus$ for some $i \in [\![n]\!] \setminus \{1, 2\}$, then applying Lemma A.32, centred in the two arcs where Δ and Δ' differ, leads to a forbidden cycle of

length n in Δ (contradiction Theorem A.2). Indeed, if $i \notin \{1,2,3\}$ then we have $\ominus$-labels on arcs of the form $(j, j+1)$ for all $j \in [\![n]\!]$, and if $i = 3$ then the forbidden cycle contains the arc $(4,3)$ labeled $\oplus$. The case $i = 0$ is not a second difference.

(d) If $lab_\Delta((i, i+1)) = \ominus$ and $lab_{\Delta'}((i, i+1)) = \oplus$ for some $i \in [\![n]\!]$, then applying Lemma A.32 centred in the arc $(1,0)$ gives $lab_\Delta((j+1, j)) = \oplus$ for all $j \in [\![n]\!] \setminus \{0,1,2,3\}$, and applying Lemma A.32 centred in the arc $(i, i+1)$ gives $lab_{\Delta'}((j+1, j)) = \ominus$ for all $j \in [\![n]\!] \setminus \{i, i-1, i-2\}$. Starting from $n = 9$ we have $([\![n]\!] \setminus \{0,1,2,3\}) \cap ([\![n]\!] \setminus \{i, 1-1, i-2\}) \neq \emptyset$, and, as a consequence, there is an arc $((j+1, j))$ in the case of Item (c).

(e) If $lab_\Delta((2,1)) = \oplus$ and $lab_{\Delta'}((2,1)) = \ominus$, then applying Lemma A.32 centred in the arc $(2,1)$ gives $lab_\Delta((0,1)) = \ominus$, however, since by hypothesis $lab_{\Delta'}((1,0)) = \ominus$ we also have $lab_{\Delta'}((0,1)) = \oplus$ otherwise there is a forbidden cycle of length two in Δ' (Theorem A.2). As a consequence, the arc $(0,1)$ is in the case of Item (d). The arc $(3,2)$ is involved in the same situation.

We conclude that in any case a second difference leads to a contradiction, either because an invalid cycle is created, or because repeated applications of Lemma A.32 give contradictory labels (both $\oplus$ and $\ominus$) to some arc for some update schedule. □

Lemma A.34. *For any $n > 8$, there exist $12n$ disjoint isomer pairs of schedules of size n for rule 160.*

Proof. The structure of this proof is similar to Lemma A.27. Fix $n > 8$ and consider the set of isomer pairs $\{\Delta, \Delta'\} \in P_n \times P_n$ (with a difference between Δ and Δ' on the labeling of arc $(1,0)$, and $lab_\Delta((1,0)) = \oplus$ and $lab_{\Delta'}((1,0)) = \ominus$). Lemma A.32 fixes the labels of many arcs of Δ, and from Lemma A.33 the same labels hold for Δ' since there is already a difference on arc $(1,0)$. Therefore, we have:

$$\begin{aligned} \text{for all } j \in [\![n]\!] \setminus \{0,1,2,3\}, lab_\Delta((j, j+1)) = lab_{\Delta'}((j, j+1)) = \ominus, \\ lab_\Delta((j+1, j)) = lab_{\Delta'}((j+1, j)) = \oplus, \\ lab_\Delta((1,2)) = lab_{\Delta'}((1,2)) = \oplus, \\ \text{and } lab_\Delta((2,1)) = lab_{\Delta'}((2,1)) = \ominus. \end{aligned}$$

Furthermore, the labeling of the arc $(1,0)$ is given by our hypothesis, and from Theorem A.2 (to avoid a forbidden cycle of length two in Δ) and Lemma A.33 (equality of lab_Δ and $lab_{\Delta'}$ except for the arc $(1,0)$) we also have $lab_\Delta((0,1)) = lab_{\Delta'}((0,1)) = \oplus$. As a consequence, it remains to consider 2^4 possibilities for the labelings of arcs

$$(2,3), (3,4), (3,2) \text{ and } (4,3)$$

(which are equal on Δ and Δ', again by Lemma A.33).
Among these possibilities, seven create a forbidden cycle of length two when the labels of the two arcs between cells 1 and 2, or 2 and 3, are both $\ominus$ (see Figure A.33 relative to rule 128; the seven possibilities for rule 160 are analogous with the four respective arcs we are now considering).
Among the nine remaining possibilities, three do not correspond to isomer pairs, as we will prove now by exhibiting for each of them a configuration $x \in \{0,1\}^n$ such that the images at cell 0 differ in Δ and Δ'. These three possibilities are depicted on Figure A.39, let us denote them by $\{\Delta_e, \Delta'_e\}$ for $e \in [\![3]\!]$.

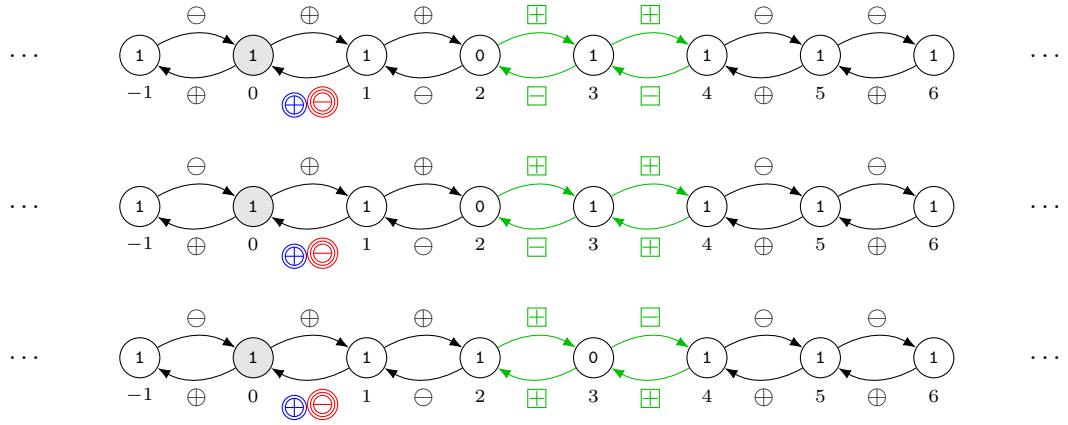


Figure A.39 – Three pairs $\{\Delta_e, \Delta'_e\}$ for $e \in [\![3]\!]$ for rule 160 ($\{\Delta_0, \Delta'_0\}$ on the top; $\{\Delta_1, \Delta'_1\}$ in the middle; $\{\Delta_2, \Delta'_2\}$ on the bottom) not corresponding to isomer pairs because for each of them there exists a configuration $x \in \{0,1\}^n$ such that $f_{160}^{(\Delta_e)}(x)_0 = 1 \neq 0 = f_{160}^{(\Delta'_e)}(x)_0$. The states of configuration x are given inside the cells.

- For $\{\Delta_0, \Delta'_0\}$ we have $x \in \{0,1\}^n$ with $x_2 = 0$ and all other cells in state 1,
- For $\{\Delta_1, \Delta'_1\}$ we have $x \in \{0,1\}^n$ with $x_2 = 0$ and all other cells in state 1,
- For $\{\Delta_2, \Delta'_2\}$ we have $x \in \{0,1\}^n$ with $x_3 = 0$ and all other cells in state 1.

One can check that in these three cases $e \in [\![n]\!]$ with these three respective configurations, we have $f_{160}^{(\Delta_e)}(x)_0 = 1$ but $f_{160}^{(\Delta'_e)}(x)_0 = 0$, because in both update schedules of each pair the left neighbor of cell 0 (cell -1) will be updated to state 1, and the right neighbor of cell 0 (cell 1) will be updated to state 0 before the update of cell 0 in Δ'_e whereas it is still in state $x_1 = 1$ when cell 0 is updated in Δ_e.
The six remaining possibilities are presented on Figure A.40. Let us argue that they indeed correspond to isomer pairs:

- neither Δ nor Δ' contain a forbidden cycle. Hence, they are pairs of non-equivalent update schedule,
- for any $i \in [\![n]\!] \setminus \{0\}$, we have $\overleftarrow{\mathtt{d}}_\Delta(i) = \overleftarrow{\mathtt{d}}_{\Delta'}(i)$ and $\overrightarrow{\mathtt{d}}_\Delta(i) = \overrightarrow{\mathtt{d}}_{\Delta'}(i)$. Hence, $f_{160}^{(\Delta)}(x)_i = f_{160}^{(\Delta')}(x)_i$ for any $x \in \{0,1\}^n$ (Lemma A.4). For cell 0, let us show that $f_{160}^{(\Delta)}(x)_0 = f_{160}^{(\Delta')}(x)_0$ for any $x \in \{0,1\}^n$. In order to have a difference in the update of cell 0, one of the two update schedules must update it to state 1. Now, remark that, given the definition of rule 160, the only possibility for cell 0 to be updated to state 1 in some update schedule (recall that $\overleftarrow{\mathtt{d}}_\Delta(0) = \overleftarrow{\mathtt{d}}_{\Delta'}(0)$) is that $x_0 = x_{-1} = x_{-2} = \cdots = x_{-\overleftarrow{\mathtt{d}}_\Delta(0)-2} = 1$, and $x_{-\overleftarrow{\mathtt{d}}_\Delta(0)} = 1$. Indeed, if any of these cells is in state 0, then at some point in the update of the chain of influence to the left of cell 0 (in this order: cell $-\overleftarrow{\mathtt{d}}_\Delta$ then $-\overleftarrow{\mathtt{d}}_\Delta + 1$ then $\ldots$ then -1 and finally 0) some cell will be updated to state 0, and then all subsequent cells will be updated to state 0 as well. Given that $\overleftarrow{\mathtt{d}}_\Delta(0) = \overleftarrow{\mathtt{d}}_{\Delta'}(0) \geq n-3$, this would enforce the states of all cells in x except (in the order of Figure A.40):
  - cells 1 and 3 for the first and third pairs,

– cells $1, 2$ and 4 for the second, fourth and fifth pairs,

– cell 2 for the sixth pair.

A straightforward exhaustive analysis of these $2 \times 2^2 + 3 \times 3^2 + 2$ cases would convince the reader that, for any configuration $x \in \{0,1\}^n$ where cell 0 may be updated to state 1 in Δ or in Δ' (otherwise $f_{160}^{(\Delta)}(x)_0 = f_{160}^{(\Delta')}(x)_0 = 0$), it turns out that $f_{160}^{(\Delta)}(x)_0 = f_{160}^{(\Delta')}(x)_0$ (this is tedious but reveals the nice combinatorics of green labels on Figure A.40).

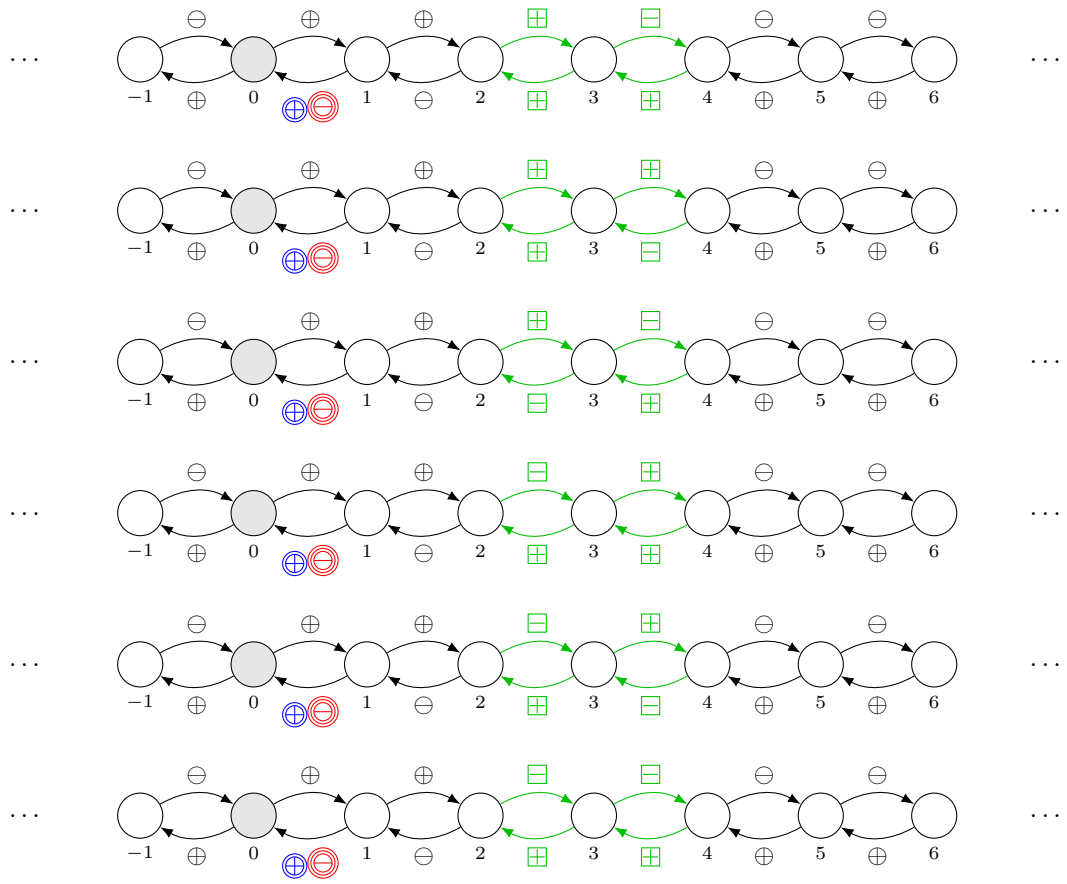


Figure A.40 – Six base isomer pairs $\{\Delta, \Delta'\}$ for rule 160 in the proof of Lemma A.34. Blue double-circled labels are part of lab_Δ, and red triple-circled labels are part of $lab_{\Delta'}$. Squared green labels are on the arcs on which we consider the six remaining possibilities. The remaining labels are those commons between lab_Δ and $lab_{\Delta'}$.

We have seen so far that there are exactly six isomer pairs with their unique difference (Lemma A.33) on the arc $(1, 0)$. Let us finally argue that these six *base* pairs for rule 160 give $12n$ distinct pairs when considering their rotations and left/right exchange, *i.e.* an update schedule belongs to at most one pair.

It is clear from Figure A.40 that all the base pairs are all disjoint. Moreover, considering the pattern $\ominus\ominus\ominus\ominus\oplus\oplus$ and any of the $24n$ update schedules, for any $n > 8$ either it appears exactly once on arcs of the form $(i, i+1)$, or its mirror appears exactly once on arcs of the form $(i+1, i)$,

but not both. This allows us to uniquely determine the left/right exchange and rotations applied to some base pair, and the remaining labelings straightforwardly allow to determine one of the six base isomer pair, and one of Δ or Δ'. Therefore all isomer pairs are disjoint. □

As a consequence of Lemma A.34, we have the following result.

Theorem A.35. *For any* $n > 8$, $\mu_s(F_{160,n}) = \frac{3^n - 2^{n+1} - 12n + 2}{3^n - 2^{n+1} + 2}$.

A.3 Conclusion and perspectives

Asynchrony highly impacts the dynamics of CA and new original dynamical behaviours are introduced. However, not all schedules produce original dynamics. For this reason, a measure to quantify the sensitivity of ECA *w.r.t.* to changes of the update schedule has been introduced [Ruivo et al. (2018)]. All ECA rules were then classified into two classes: max-sensitive and non-max sensitive.

This appendix provided a finer study of the sensitivity measure with respect to the size of the configurations. We found four classes (see Table 2). In particular, it is interesting to remark that each class is characterised by a different base of the exponential in the numerator of the sensitivity measure. Class I has base 1 (and indeed it contains the simplest dynamical behaviours); class II and IV have base 2 and 3, respectively. Finally, class III contains only the ECA rule 8 and presents a contrast between the simple dynamical behaviour of ECA rule 8 and ϕ (the golden ratio) as a base of the numerator of the sensitivity function.

Moreover, remark that in the classical case, the limit set of the ECA rule 8 (class III) is the same as ECA rule 0 (class I) and it is reached after just two time steps. This suggests that further research should be performed to understand what are the relations between the limit set (both in the classical and in the asynchronous cases) and the sensitivity to synchronism. A similar idea has been investigated in works on *block-invariance* [Goles et al. (2018), Goles et al. (2015)], with the difference that it concentrates only on the set of configurations in attractors, and discards the transitions within these sets.

In the literature one can find a notion similar to sensitivity to synchronism called π-independence [Macauley et al. (2011), Macauley et al. (2007)]. However, π-independence is defined on asynchronous updates that are permutations of the set of cells. Consequently, these are only some of the possible asynchronous updates that can be introduced thanks to block-sequential update schedules. Remark that a max-sensitive rule can be π-independent. This implies that each block-sequential update schedule with blocks of size 1 generates at least a difference in the non-periodic dynamics. The four classes introduced here allow to further distinguish the 56 not max-sensitive rules, which are reduced to 21 if one considers topological conjugacy. Of these 21 rules, 13 are π-independent and 6 are not. Among the π-independent rules, we can now distinguish four insensitive $(0, 51, 200, 204)$, six low-sensitive $(12, 28, 32, 60, 136,$ and $140)$, one medium-sensitive (rule 8), and two almost max-sensitive (128 and 160) rules. Furthermore, the remaining insensitive rules $(3, 15, 34, 44,$ and $170)$ and the remaining almost max-sensitive rules (rule 162) are not π-independent. This implies that even with an update schedule with one cell per block forces differences even in the periodic dynamics.

We stress that this study focuses on block-sequential updating schemes. A promising research direction consists in investigating how the sensitivity functions change when different updating schemes are considered such as block-parallel ones [Demongeot and Sené (2020)].

Another interesting research direction would consider the generalisation of our study to larger classes of CA in order to verify if a finer grained set of classes appears or not. From experiments, it seems that the set of possible sensitivity functions is tightly related to the structure of the neighbourhood.
An ambitious research program would try to extend the analysis performed here to other sets of configurations. A good starting point might be bi-periodic configurations, *i.e.* infinite configurations that admit an integer $z \in \mathbb{Z}$ such that all cells with index greater than z have different periodic pattern than all cells with index less than or equal to z (consider for example a configuration which is 0 for all $i \leq 0$ and 1 for all $i > 0$). An even more ambitious research would consider arbitrary infinite configuration in the spirit of [Ruivo et al. (2020)]. We think that the approach by "fair" measures could be a good starting point [Dennunzio et al. (2012), Dennunzio et al. (2013)].

Factorisation de Systèmes Dynamiques Discrets

Sara RIVA

Résumé

Un Système Dynamique Fini à temps Discret (SDD) est constitué d'un ensemble fini $\mathcal{X}$, dit espace des états, et d'une fonction f, dite fonction de mise à jour (associant à un état v l'état $f(v)$). Les SDD sont un outil formel pour modéliser de nombreux phénomènes en physique, en mathématique, en biologie, et, bien sûr en informatique. Si la formalisation mathématique et les résultats qui en découlent sont élégants et parlants, souvent, ces résultats sont peu applicables en pratique à cause de leur coût computationnel élevé. Dans la littérature, il est connu que les SDD équipés d'opérations de somme et de produit appropriées forment un semi-anneau commutatif. Cette structure algébrique nous permet d'écrire des équations polynomiales dans lesquelles les coefficients et les inconnues sont des SDD. En particulier, si nous sommes intéressés par une certaine dynamique dérivée de données expérimentales, nous pouvons écrire une équation avec celle-ci comme terme de droite constant et modéliser des hypothèses sur la fonction f (ou ses propriétés) dans un terme de gauche polynomial. Trouver des solutions à cette équation permet de mieux comprendre le phénomène et ses propriétés. Cette approche est intéressante mais pose des limites computationnelles importantes. En effet, résoudre une équation polynomiale (à plusieurs variables) est, en général, indécidable et même en se concentrant sur le cas de la validation des hypothèses, le coût computationnel reste élevé. L'idée est alors de chercher des approximations donnant des informations pertinentes sur les solutions de l'équation originale. Trois abstractions (équations plus simples) sont introduites afin d'identifier : le nombre d'états des variables, le comportement asymptotique ou le comportement transient (comportement avant que le système se stabilise). Chaque abstraction est construite d'un point de vue théorique et algorithmique dans le but d'introduire une méthode pour effectuer la validation d'hypothèses sur SDD. Dans cette thèse, il est montré qu'au moyen de transformations algébriques, il est possible d'énumérer les solutions d'une équation polynomiale avec un terme droit constant par l'énumération d'un nombre fini d'équations plus simples. Enfin, le lien entre la résolution ces équations simples et le problème de la cancellation connu en théorie des graphes est exploré. Cela a permis de trouver une borne supérieure linéaire sur le nombre de solutions.

Mots-clés : Systèmes Dynamiques Discrets, Complexité, Vérification.

Abstract

A Finite Discrete-time Dynamical System (DDS) consists of a finite set $\mathcal{X}$, called state space, and a function f, called next-state map (which associates to a state v the state $f(v)$). DDS are a formal tool for modelling phenomena that appear in Physics, Mathematics, Biology, and, of course, in Computer Science. While the mathematical formalisation and the results that have been found up to nowadays are elegant and meaningful, often they are not very suitable in practice because of their high computational cost. In the literature, it is known that DDS equipped with appropriate sum and product operations form a commutative semiring. This algebraic structure allows us to write polynomial equations in which the coefficients and unknowns are DDS. In particular, if we are interested in some dynamics derived from experimental data, we can write an equation with this as a constant right-hand term and model assumptions about the function f (or its properties) in a polynomial left-hand term. Finding solutions to this equation allow us to better understand the phenomenon and its properties. This approach is interesting but it has important limitations from a computational point of view. Solving a polynomial equation (with several variables) is, in general, undecidable, and even if we focus on the case of hypothesis validation, the computational cost remains high. The idea is then to look for approximations that give relevant information about the solutions of the original equation. It is possible to introduce three abstractions (simpler equations) to identify: the number of states of the variables, the asymptotic behaviour, or the transient behaviour (what happens before the system stabilises). Each one is built from a theoretical and algorithmic point of view to introduce a method to perform hypothesis validation on DDS. In this thesis, it is shown that through algebraic transformations, it is possible to enumerate the solutions of a polynomial equation with a constant term by enumerating a finite number of simpler equations. Finally, the connection between the solution of these simple equations and the cancellation problem known in graph theory is explored. This allowed us to find a linear upper bound on the number of solutions.

Keywords: Discrete Dynamical Systems, Complexity, Formal Verification.