Informatics

Approaching Emergent Patterns with Kronecker Algebra in Industrial Agents

DISSERTATION

zur Erlangung des akademischen Grades

Doktor der Technischen Wissenschaften

eingereicht von

Dipl.-Ing. Patrick Denzler, MSc
Matrikelnummer 11743126

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Univ.Prof. Dipl.-Ing. Dr.techn. Wolfgang Kastner
Zweitbetreuung: Ao.Univ.Prof. Dipl.-Ing. Dr.techn. Johann Blieberger

Diese Dissertation haben begutachtet:

Prof. Dr. Moris Behnam

Prof. Dr. Bernd Burgstaller

Wien, 28. Juni 2023

Patrick Denzler

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

TU WIEN Bibliothek Your knowledge hub
Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.

Informatics

Approaching Emergent Patterns with Kronecker Algebra in Industrial Agents

DISSERTATION

submitted in partial fulfillment of the requirements for the degree of

Doktor der Technischen Wissenschaften

by

Dipl.-Ing. Patrick Denzler, MSc
Registration Number 11743126

to the Faculty of Informatics

at the TU Wien

Advisor: Univ.Prof. Dipl.-Ing. Dr.techn. Wolfgang Kastner
Second advisor: Ao.Univ.Prof. Dipl.-Ing. Dr.techn. Johann Blieberger

The dissertation has been reviewed by:

Prof. Dr. Moris Behnam | Prof. Dr. Bernd Burgstaller

Vienna, 28th June, 2023

Patrick Denzler

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Erklärung zur Verfassung der Arbeit

Dipl.-Ing. Patrick Denzler, MSc

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 28. Juni 2023

Patrick Denzler

TU Bibliothek WIEN Your knowledge hub
Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Abstract

Cyber-physical systems (CPSs) integrate distributed physical components, software, and monitors. Their behaviour results from the interactions between the parts and is not deducible from the components. In schools of fish or ant colonies, nature knows similar phenomena and is commonly summarised as self-organisation or emergent behaviour. A systematic literature review revealed properties required in multi-agent systems (MASs) to show emergent pattern formations. The dynamically interacting agents need to create, without external control, a robust pattern that is novel w.r.t. the individual parts of the system over time. Nevertheless, there are limited methods to identify such patterns. A potential solution is a formal language approach based on a cooperating array grammar system and Kronecker Algebra.

Kronecker Algebra manipulates matrices, representing state machines capable of executing formal language grammars. The new Kronecker Synthesise and Symmetric Skip operations enable scenario synthesis to identify unexpected behaviour while ensuring consistency. Adding execution priorities allow pinpointing priority inversions between agents sharing a common resource. Moreover, it enables worst-case execution time (WCET) analysis of processes executing on one central processing unit (CPU).

Applying the operations to a publish-subscribe communication system model results in pattern formations that individual agents cannot execute. Adding priorities affects the pattern formation and the execution time of agent interactions. Experimentations with a time-predictable publish-subscribe environment confirm the findings and the suitability of the proposed approach. Limiting the results is the absence of multiple experiments. Future work includes extending Kronecker Algebra to handle concurrent prioritised agent interactions and linear time model checking.

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

TU WIEN Bibliothek Your knowledge hub
Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.

Zusammenfassung

Cyber-Physische Systeme (CPSs) integrieren, verteilte physische Komponenten, Software und Monitore. Ihr Verhalten resultiert aus den Wechselwirkungen zwischen den einzelnen Teilen und ist nicht aus den Komponenten ablesbar. Die Natur kennt ähnliche Phänomene in Fischschwärmen oder Ameisenkolonien und welche gemeinhin als Selbstorganisation oder Emergenzverhalten zusammengefasst werden. Eine systematische Literaturrecherche ergab Eigenschaften, die in Multiagentensystemen (MASs) erforderlich sind, um solche Musterbildungen zu erzeugen. Die dynamisch interagierenden Agenten müssen ohne externe Kontrolle ein robustes Muster erstellen, das in Bezug auf die Funktionsweise und der einzelnen Teile des Systems im Laufe der Zeit neu ist. Dennoch gibt es nur begrenzte Methoden, um solche Muster zu identifizieren. Eine mögliche Lösung ist ein formaler Ansatz, der auf einem kooperierenden Array-Grammatiksystem und der Kronecker-Algebra basiert.

Die Kronecker-Algebra manipuliert Matrizen, die Zustandsmaschinen darstellen, die in der Lage sind, formale Sprachgrammatiken auszuführen. Die neuen Kronecker-Synthese- und Symmetric-Skip Operationen ermöglichen die Szenariosynthese, um unerwartetes Verhalten zu identifizieren und gleichzeitig Konsistenz sicherzustellen. Durch das Hinzufügen von Ausführungsprioritäten können Prioritätsumkehrungen zwischen Agenten ermittelt werden, die sich eine gemeinsame Ressource teilen. Darüber hinaus ermöglicht es die worst-case execution time (WCET) analysis von Prozessen, die auf einem zentralen Prozessor ausgeführt werden.

Die Anwendung der Operationen auf ein Publish-Subscribe Kommunikationssystemmodell führt zu Musterbildungen, die einzelne Agenten nicht ausführen können. Das Hinzufügen von Prioritäten wirkt sich auf die Musterbildung und die Ausführungszeit von Agenteninteraktionen aus. Experimente mit einer zeitvorhersehbaren Publish-Subscribe Umgebung bestätigen die Ergebnisse und die Eignung des vorgeschlagenen Ansatzes. Die Ergebnisse werden durch das Fehlen mehrerer Experimente eingeschränkt. Zukünftige Arbeiten umfassen die Erweiterung der Kronecker-Algebra, um gleichzeitige priorisierte Agenteninteraktionen und die Überprüfung linearer Zeitmodelle zu verarbeiten.

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Acknowledgements

This dissertation reflects five years of research education through my eyes. However, in addition to my own world, it also represents the worlds of other interesting individuals I have had contact with. First, I would like to express my gratitude to my two supervisors, Professor Wolfgang Kastner and Professor Johann Blieberger. Wolfgang, thank you for these few years of continuous support and optimism, "Alles wird gut", should be in every supervisor's vocabulary. Hans, thank you for all the patience and the introduction to Kronecker Algebra. I would also like to especially thank Professor Dr. Moris Behnam and Professor Dr. Bernd Burgstaller for the final reviews of this dissertation, which have undoubtedly improved its quality.

A large portion of these five years was spent in the company of my colleagues at the Automation Systems Group, to whom I am grateful. Your companionship, Thomas, Daniel, Jürgen, Dieter and Philip, made this process much more enjoyable. Especially noted Dr. Thomas Frühwirth, co-author and a formidable opponent of where to place commas. Not to forget Ruth Fochtner, our wizard handling the depths of TU Wien's bureaucracy.

I should also note my colleagues from the Marie Skłodowska-Curie Fellowship FORA. Especially Cosmin and Basil for the many exciting discussions in a traditional Viennese coffee house. And my dear friend Zeinab, thank you for continuous motivational aid when there was much frustration.

The people I met at Mälardalen University should not be forgotten, as they enabled this journey in the first place. Dr. Anna Granlund, thank you for your outstanding help and care when things turned difficult. Moreover, Christer, Philip, Mohammed, Gita, Leo, Inés and all the others, thank you for the cheerful Fika's and discussions.

Alongside this journey, I received much backing from my closest friends. Marco, Michéle, Mika, Mahnaz, Farhad, Stefano, Martin, Nina and Sarah, I am sure I will be around more again. Gerhard Storz, the work at COMACON AG, definitely shaped my understanding of computers. Moreover, Dr. Michael Gnoth, thank you for all the valuable learnings during my time in Dubai.

Neda, thank you for all your support and patience during these years.

Finally, a note to my very first supporters, my parents: thank you for your unconditional love and encouragement. Mam and Dad, you showed me that I can achieve my goals despite all the obstacles in life. Without you, all this would have been impossible. Last but not least, to Manuel, I am grateful to have you as my brother.

Patrick, Vienna, 28.06.2023

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

„Out beyond ideas of wrongdoing

and rightdoing, there is a field.

I'll meet you there.“

Rumi

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Contents

1 Introduction **1**
1.1 Emergent Pattern . . . 3
1.2 Relation to Industrial Systems . . . 4
1.3 Formalisation and Kronecker Algebra . . . 4
1.4 Problem Statement and the Research Questions . . . 5
1.5 Structure of the Dissertation . . . 5
1.6 Contributions within this Dissertation . . . 6
1.7 Publications . . . 6
1.8 Reference to FORA . . . 8

2 Complex Systems, Self-Organisation and Emergence **9**
2.1 Complex Systems . . . 9
2.1.1 Measures of Complexity . . . 10
2.2 Self-Organisation . . . 12
2.2.1 A Bit of History about Self-Organisation . . . 12
2.2.2 Defining Self-Organisation . . . 13
2.2.3 Characteristics of Self-Organisation . . . 13
2.2.4 Self-Organization From an Information Perspective . . . 15
2.3 Emergence . . . 15
2.3.1 A Bit of History about Emergence . . . 15
2.3.2 Defining Emergence . . . 16
2.3.3 Where to find Emergence . . . 17
2.3.4 Characteristics of Emergence . . . 18
2.3.5 Classifications of Emergence . . . 20
2.3.6 The Design-observed Discrepancy . . . 21
2.3.7 The Information Dynamics of Emergence . . . 21
2.3.8 Macro-Properties, Scope and Resolution . . . 23
2.4 Differences Between Self-Organisation and Emergence . . . 23
2.4.1 Similarities . . . 23
2.4.2 Differences . . . 24
2.4.3 Combining Emergence and Self-Organisation . . . 25
2.5 Unwanted Emergent Behaviour . . . 25
2.5.1 Types of Methods to Detect Emergent Behaviour in Systems . . . 26
2.5.2 How to confirm Emergent Behaviour . . . 27
2.5.3 How to Simulate and Model Emergent Behaviour . . . 28
2.5.4 Influencing Emergent Behaviour . . . 30
2.5.5 Predicting Emergent Behaviour . . . 31

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

2.6 Summary . . . 32

3 Methodology 33
3.1 A Short Excursion in Testing vs. Proving Programs Correct . . . 33
3.1.1 A Quest for Final Truths . . . 33
3.1.2 Formal Verification, the Only Solution . . . 34
3.1.3 The Two Debates Compared . . . 36
3.1.4 Conclusion . . . 37
3.2 Choosing Desing and Creation as Research Strategy . . . 37
3.3 Systematic Literature Review . . . 38
3.3.1 The Guiding Research Questions . . . 39
3.3.2 Search Strings and Databases . . . 39
3.3.3 Acceptance/Rejection Criteria . . . 40
3.3.4 Data Extraction . . . 40
3.3.5 Results . . . 40
3.3.6 A Comment on the Conducted Literature Review . . . 41

4 Emergent Behaviour in Industrial Systems 43
4.1 Seamless Communication in Industry . . . 43
4.2 Emergence in Industry 4.0, IIoT and Multi-Agent Systems . . . 44
4.3 A Little bit Critique and Some Clarifications . . . 45
4.4 What Could Cause Emergent Behaviour in a Multi-Agent System? . . . 46
4.4.1 Informal Definition of Basic Emergence and the Connection to Formal Languages . . . 47
4.5 A Formal Language Proposal . . . 48
4.5.1 Formal Languages . . . 48
4.5.2 Cooperating Grammar Systems . . . 48
4.5.3 Cooperating Array Grammar Systems . . . 49
4.5.4 Modifications on Cooperating Grammar System . . . 51
4.5.5 Summing up the Agents Behaviour . . . 51
4.5.6 A Formal Definition of a Basic Emergence . . . 51
4.5.7 Some Comments . . . 52
4.6 Finding a Suitable Tool . . . 52
4.7 Summary and Limitations . . . 54

5 Kronecker Algebra — A Matrix Calculus 55
5.1 A little History of Kronecker Algebra . . . 55
5.2 Finite State Machines and their Matrix Representation . . . 56
5.2.1 Matrix Representation . . . 57
5.2.2 State Transitions (Finding Successors) . . . 59
5.3 Definition of Kronecker Product . . . 60
5.3.1 Further Properties . . . 61
5.3.2 Our Semiring . . . 61
5.3.3 Applying Kronecker Product . . . 61
5.4 Verifying Programs with Kronecker Product . . . 62
5.4.1 Control Flow Graphs . . . 63
5.4.2 Usage Scenarios . . . 64
5.4.3 Applying Kronecker Product . . . 64

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

5.4.4 Isomorphism . . . 64
5.5 Kronecker Skip . . . 65
5.6 Kronecker Sum . . . 66
5.7 Concurrent Programs and Semaphores . . . 67
5.8 Implementing Kronecker Algebra Operations (Lazy Algorithm) . . . 68
5.8.1 Expression Trees . . . 68
5.8.2 Lazy Evaluation of Kronecker Expressions . . . 68
5.9 Related work in the field of Kronecker Algebra . . . 70
5.10 Concluding Remarks . . . 70

6 Finding Implied Scenarios **71**
6.1 The Idea Behind Implied Scenarios . . . 71
6.2 Message Sequence Charts . . . 71
6.2.1 Problem Formulation . . . 73
6.2.2 Limitations . . . 73
6.3 Finding Basic 'Emergence' . . . 73
6.3.1 Scenario Collection . . . 74
6.3.2 Synthesizing message sequence charts (MSCs) to control-flow graphs (CFGs) 74
6.3.3 Combine all Scenarios and Create $\mathcal{S}_{\text{Total}}$ . . . 78
6.3.4 Analyze $\mathcal{S}_{\text{Total}}$ with Kronecker Algebra . . . 79
6.3.5 Evaluate New Scenarios . . . 79
6.4 Evaluation Example . . . 79
6.4.1 Synthesising MSCs to CFGs . . . 81
6.4.2 Combine all Scenarios and Create $\mathcal{S}_{\text{Total}}$ . . . 81
6.4.3 Analysing the Graph with Kronecker Operations . . . 82
6.4.4 Evaluate New Scenarios . . . 86
6.5 Discussion . . . 87
6.6 Related Work . . . 88
6.7 Concluding Remarks . . . 88

7 Consistency Checking **89**
7.1 A Motivating Example . . . 89
7.1.1 Consistency Between an MSC and State Machines . . . 90
7.1.2 A Desired and a Negative Scenario . . . 91
7.2 The Consistency Problem . . . 91
7.2.1 Defining the Problem . . . 91
7.2.2 Limitations . . . 91
7.3 Confirming Consistency . . . 92
7.3.1 Kronecker Symmetric Skip Operation . . . 92
7.3.2 Reusing Kronecker Synthesize Operation . . . 93
7.3.3 Bringing it All Together . . . 93
7.4 Evaluation . . . 94
7.4.1 Preparation of the finite state machine (FSM) . . . 94
7.4.2 A First Simple Example . . . 96
7.4.3 Focusing on the MSC Synthesis . . . 97
7.4.4 The Error Scenario . . . 97
7.5 Discussion . . . 98
7.6 Related Work . . . 99

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

7.7 Concluding Remarks . . . 99

8 Priority **101**
8.1 Process Prioritisation . . . 101
8.2 Bounded/Unbounded Priority Inversion . . . 102
8.3 Priority Ceiling and Inheritance Protocol . . . 103
8.4 Problem Definition . . . 104
8.5 Kronecker Priority . . . 105
8.5.1 Conditionals . . . 107
8.5.2 Branches (If and Else Statements) . . . 110
8.5.3 Sync Pattern . . . 111
8.6 Spotting Priority Inversion . . . 113
8.6.1 Introducing the Lock . . . 113
8.6.2 Introducing the Synchronisation . . . 113
8.6.3 Adjusting the Processes . . . 114
8.6.4 Apply Kronecker Algebra . . . 114
8.6.5 How to Spot Priority Inversion in the Graph? . . . 117
8.6.6 Introduce the Priority Ceiling Protocol . . . 117
8.7 Identify Starvation . . . 118
8.8 Discussion . . . 120
8.9 Related Work . . . 120
8.10 Concluding Remarks . . . 120

9 Worst Case Execution Time Analysis **121**
9.1 Introduction to timing analysis . . . 121
9.2 Problem Definition . . . 122
9.3 WCET Analysis of Shared Memory Concurrent Programs Running on a Multi-Core Architecture . . . 123
9.3.1 Modelling All Interleavings . . . 124
9.3.2 Introduce Synchronisation . . . 125
9.3.3 WCET Analysis on RCPGs . . . 125
9.4 WCET Analysis with Priorities . . . 126
9.5 Discussion and Related Work . . . 128
9.6 Concluding Remarks . . . 128

10 Evaluation **129**
10.1 Returning to the Blackboard . . . 129
10.2 Developing an Example . . . 130
10.2.1 Creating the Basic Structure . . . 130
10.2.2 Introducing Interaction . . . 131
10.2.3 A Pattern Emerges . . . 132
10.3 A Second Pattern . . . 135
10.3.1 Analysing the Paths . . . 136
10.3.2 Adding Another Subscriber . . . 137
10.4 Change the Basic Setup . . . 140
10.5 Apply Priorities to the First Example . . . 143
10.6 Adding Execution Time . . . 145
10.6.1 OPC Unified Architecture . . . 146

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

10.6.2 End-to-end latency 146
10.6.3 WCET Analysis Process 147
10.6.4 Adjusting the OPC UA Publisher 148
10.6.5 Adjusting the OPC UA Subscriber 148
10.6.6 Evaluation Setup 149
10.6.7 Execution Time Measurements 149
10.6.8 End-to-End Latency Analysis 152
10.6.9 Further Work 154
10.6.10 Connection to Kronecker Algebra 154
10.7 Time Complexity of Kronecker Operations 155
10.7.1 Optimisations of Kronecker Algebra 156
10.8 Concluding Remarks 156

11 Discussion **157**
11.1 Research Question 1 157
11.2 Research Question 2 158
11.3 Research Question 3 159
11.4 Research Question 4 161
11.5 Limitations and Related Work 162
11.6 Reliability, Reproducibility and Generalisation 163

12 Conclusion **165**
12.1 Future Work 166

13 Bibliography **167**

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

Chapter 1

Introduction

The topic of this dissertation covers several subjects not obviously related at first sight. Nevertheless, a good starting point is the behaviour of complex and distributed systems, specifically cyber-physical systems (CPSs) [LS17]. Such systems usually combine physical components with software controllers and observers, like in vehicles or robots. A specific characteristic of such systems is that their behaviour surfaces from the combination of all elements, which cannot be deduced from the individual components alone. However, let us not dive overly deep into the technical abyss.

Instead, let us explore a specific behaviour of a fish, namely the male bluegill sunfish (Lepomis Macrochirus). This fish species usually nests in a group of 150 individuals, and each builds a polygonal formed nest on the sea floor [GM81]. Each bluegill constructs its territory in the soft sand like a pit with rims around it to create borders to the adjoined nests. The individual fish acts as a selfish-heard member, most likely evolved to prevent broad predators [Ham64]. By joining the group and surrounding its nest with other nests, each member increases its protection against predators. However, while the group building is intended, each fish fiercely protects its patch of sand or borders against the other fish. This behaviour is also seen in other species of fish like the Tilapia. The result of the nest building is a colony with a beautiful pattern showing a polygonal array lined up, as shown in Figure 1.2. The pattern does not have any direct function; it is a product of individually interacting agents (fish) following a simple set of rules: Do not come too close but still be as close to protect against the enemy. In the literature, the described behaviour is either classified as a natural self-organising or an emergent system [CDF+20].

The terms self-organisation and emergence describe behaviours of systems that cannot be easily explained by the sum of the behaviour of the system elements [Sha01]. Both phenomena are reoccurring research topics that have fascinated researchers repeatedly since the 1970s. While there are strong connections to complex systems theory, the underlying concepts of self-organisation and emergence found their way into different research fields such as robotics, swarm, biology, social sciences or physics [For90, CM95, KMRF+03].

[1] Source: https://commons.wikimedia.org/wiki/File:Lepomis_macrochirus.jpg.

Figure 1.1: The male bluegill sunfish (Lepomis Macrochirus)[1]

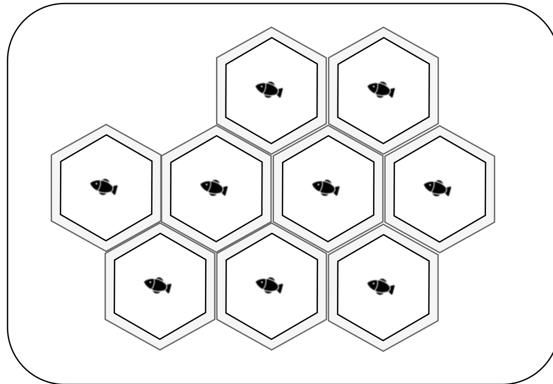


Figure 1.2: Several male bluegill sunfish (Lepomis Macrochirus) nests that form a polygonal array.

Self-organisation refers to a wide range of pattern-formation processes in physical and biological systems [Sha01, CDF+20]. Examples are sand grains assembling into rippled dunes, fish joining in schools, chemical reactants creating swirling spirals or cells assembling highly structured tissues. One essential commonality of these systems is how they produce and hold their order and structure. In such self-organising systems, the pattern formation process happens through internal interactions without any external influences to provide direction or intervention. The author Haken [Hak77, p.191] describes the formation process in an example: "Consider, for example, a group of workers. We then speak of organization or, more exactly, of organized behavior if each worker acts in a well-defined way on given external orders, i.e., by the boss. We would call the same process as being self-organized if there are no external orders given but the workers work together by some kind of mutual understanding." In the example, the boss does not contribute to the pattern formation and is therefore considered external, i.e., the boss is not part of the system.

The example illustrates why systems that do not have self-organising properties can produce high-order patterns. The order is imposed on the system by instructions from a leader, directives coming from blueprints or recipes or pre-existing patterns in the environment (templates) [CDF+20]. An example of an imposed pattern is a marching band forming big letters on a football field. The band's members are guided in their behaviour by a set of externally imposed instructions. Each member gets specific movement instructions, so the final pattern emerges. While each musician does not receive instructions and ignores the other members while marching, the steps and directions are predefined and, therefore, not self-organised.

Camazine et al. [CDF+20, p.8] provide a well-formulated definition for self-organisation in the context of pattern formation: "Self-organisation is a process in which a pattern at a global level of a system emerges solely from numerous interactions among the lower-level components of the system. Moreover, the rules specifying interactions among the system's components are executed using only local information without reference to the global pattern." In other words, the pattern itself is an emergent property of the system. Any external entity or influence does not impose the pattern. The differences between self-organisation and emergent properties be defined in later chapters. For now, it is enough to understand that emergent properties are system features that arise unexpectedly from interactions among the system's components. At the same time, such a property cannot be deducted by simply examining the properties of the systems' components in isolation. As a side note, system components do not necessarily have to interact directly.

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

1.1 Emergent Pattern

The definition of Camazine et al. [CDF+20] contains the important term pattern. In this context, a pattern is a distinct, organised arrangement of objects in space and time [Bal01]. Such patterns are a school of fish, the synchronous flashing of fireflies, the complex architecture of a termite mound or a raiding column of army ants, to continue with examples from biology. The pigmentation patterns of shells, fish and mammals or lichen growth are not less spectacular [Mur88].

Comprehending the pattern formation process is essential for understanding self-organisation and emergence. Staying in the biological context, the building blocks of patterns are sometimes living units (fish, ants or nerve cells) or, in other cases, inanimate objects such as bits of dirt and faecal cement that make up the termite mound. In both cases, a system of living cells or organisms is the pattern formation source without any external directing influence, such as the environment or a leader. The system's entities interact to produce the pattern based on local, not global, information [Hak77]. A simple example is a school of fish where each fish bases its behaviour on the position and velocity of its nearest neighbours (cf. Figure 1.3 on the left). No fish knows the global behaviour of the whole school [HW92], but all fish together can perform complex formations such as avoiding predators (cf. Figure 1.3 on the right). Similarly, a detachment of army ants bases its activity on local concentrations of pheromones by other ants. No "general" ant gives directions.

The literature combines self-organisation, emergent properties, complexity, dissipative structures, and chaos under the umbrella term nonlinear systems [Pri78]. The actual meanings of chaos and dissipative structures highly differ between the scientific and everyday definitions. Let us start with the term complex. The before exemplified organisms use relatively simple behavioural rules to generate structures and patterns on the global system level. Those structures are more complex than the components and processes they emerge [Pag88]. Therefore, systems are not complex because they involve many behavioural rules and large numbers of different components but because of the complex structure and pattern that the system generates. The terms complexity and complex systems describe systems build upon interacting units that create global properties not present at the lower level [Sha01]. These systems usually have diverse responses, often sensitively dependent on the system's initial state and nonlinear interactions among its components. Moreover, those nonlinear interactions involve amplification or cooperativity [Ode02a]. It becomes possible for complex behaviours to emerge, although the system components are similar and follow simple rules. In other words, complexity is not dependent on complicated components or rules [Sha01].

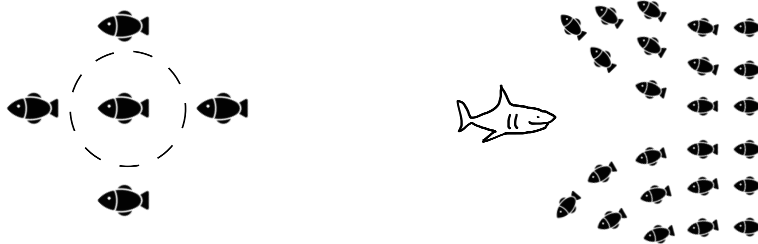


Figure 1.3: On the left, each fish orients itself on the position and velocity of its nearest neighbours. On the right is a school of fish, showing self-organised pattern formation by avoiding a predator.

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

1.2 Relation to Industrial Systems

What do CPSs or industrial systems have in common with the self-organisation or emergent behaviour of a school of fish? That question was the starting point of this dissertation and was rooted in the unexpected behaviour of an industrial communication system built for a research project. The system consisted of several agents communicating and producing high-level interactions not intended in the original design. Despite investigating the single agents, the root cause could only be found by analysing the message pattern created during the system's operation. To answer the above question, CPSs or industrial systems can, like their biological counterparts, create patterns that qualify as emergent properties. Such systems fulfil some basic requirements needed for emergent patterns such as interacting parts [CDF+20], decentralised control [Ode02a], autonomy [Sha01], adaptability or robustness w.r.t. changes [Gol99] and can create a micro-macro effect [Gol99]. Moreover, the possibility for systems to show emergent behaviour increased with the ongoing trend to seamlessly interconnect systems over the Internet, as proposed by Kevin Ashton in 1999, when he coined the term Internet of Things (IoT). The paradigms Industrial Internet of Things (IIoT) and Industry 4.0 (I4.0) [LFK+14] apply the idea of IoT into the industrial context and therefore inherit the same potential.

A large body of academic research focuses on agent systems, specifically self-organising robots or other interacting agents. However, little is done to provide functional methods or tools in the context of emergent pattern generation in such systems. The predominant tools are simulations, where the agents show emergent patterns over time. Questions about why specific behaviour occurs often remain unanswered or explained by elements of surprise, accompanied by limited examples. One reason might be that multi-agent system (MAS) agents are sometimes capable of highly complex tasks, and deducting the system's behaviour backwards is almost impossible [LS17]. Based on the experiences made with the research industrial communication system, this dissertation aims to find a possibility or method to identify emergent patterns in MAS and industrial systems.

1.3 Formalisation and Kronecker Algebra

Parallel research activities on Kronecker Algebra raised the question of whether it is possible to utilise this type of algebra to identify patterns in agent interactions. Johann Georg Zehfuss introduced the primary operator by the symbol $\otimes$ in 1858 [Zeh58]. This operator manipulates matrices and had, throughout history, various names, such as the Zehfuss product, the Hurwitz product, the producttransformation, the conjunction, the tensor product, the direct product, or Kronecker product [Gra18]. Other academics added further operations to Kronecker Algebra. For example, Gerhard Küster [Küs91] proved that Kronecker Sum generates all interleavings of concurrently executing automata and Mittermayr and Blieberger [MB11] applied operations (i.e., Kronecker product and Kronecker Sum) to automata within the Kuich-Salomaa notation.

As indicated above, some research treats self-organisation and emergence rather informally, while others provide formal approaches without appropriate tools. However, searching for methods to identify emergent behaviour also yielded more tangible approaches. Kubí [Kub03] proposes using formal languages to identify emergent behaviours in MAS. Each agent has its language; they generate a new system language when interacting. If the system language differs from the sum of the agent languages, the system may show basic emergent behaviour. The connecting elements with Kronecker Algebra are finite state machines (FSMs). FSMs can represent formal languages on the lower levels of the Chomsky hierarchy [Cho56]. By transforming the FSMs into their matrix representations, we can use Kronecker operations to identify emergent patterns in MAS.

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

The following steps included extending Kronecker Algebra operations to be suitable for finding emergent patterns. At the same time, the operations also contribute to other research issues and represent standalone contributions.

1.4 Problem Statement and the Research Questions

The problem statement and aim of this dissertation narrowed down is as follows. The probability of modern MAS and industrial systems showing emergent behaviours or patterns increases with their steady interconnection to other systems. Research in self-organisation and emergent properties focus on simulation and provide little to no alternative methods or tools to predict emergent patterns. Therefore, this dissertation aims to explore the applicability of Kronecker Algebra to identify emergent patterns in MAS or industrial systems. Moreover, it aims to extend Kronecker Algebra with operators suitable for enabling system designers to predict or prevent unexpected emergent behaviour.

The following research questions guided the research activities to fulfil the aim of this dissertation.

> **RQ1**: How are the various definitions of emergence and self-organisation in the different research disciplines interconnected and express the critical characteristics of these phenomena to identify emergent behaviour in the context of multi-agent systems (MASs)?
>
> **RQ2**: How can interacting agents in a MAS be formally represented to enable the detection of emergent patterns within the entire system while preserving the essential characteristics for pattern formation?
>
> **RQ3**: Can the combination of agent interactions (scenarios) with Kronecker Algebra create a system representation retaining the information essential to detect emergent patterns while ensuring consistency?
>
> **RQ4**: How do factors such as interaction priorities between agents, message transmission times and execution times of the agents influence the emergent pattern formation of a MAS?

Each chapter contributes the answers to the research questions. Moreover, there are some delimitations. Firstly, the aim is not to identify the root causes for emergent behaviour or pattern formation in MAS nor the classification, pattern recognition or proving that the found patterns are emergent. Secondly, the aim is not to provide extensive experiments but instead to provide the foundation for future research. Furthermore, the presented methods or implementations are research prototypes and do not aim for any use in industry or professional settings.

1.5 Structure of the Dissertation

The structure of the dissertation aligns with the research questions for better guidance. Chapter 2 provides a broader overview of self-organisation and emergence based on the outcomes of a conducted literature review. The focus lies on the connections between self-organisation and emergence and presenting the critical characteristics of these phenomena and clear definitions. The following Chapter 3 presents the reasons why a Design and Creation as Research Strategy was chosen and

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

how the systematic literature was conducted. Chapter 4 narrows the focus on self-organisation and emergence in MAS and introduces the possibility of formally representing MAS and connecting it to Kronecker Algebra. Chapter 5 introduces Kronecker Algebra operations and the lazy algorithm. Chapters 6 and 7 show how Kronecker Algebra can find new scenarios in MAS and ensure consistency between message sequence charts (MSCs) and FSMs. The following Chapter 8 introduces the possibility of priorities in Kronecker operations, which are essential in the worst-case execution time (WCET) analysis presented in Chapter 9. Chapter 10 contains the evaluation where the newly introduced Kronecker operations are applied to an example, showing that it is possible to identify patterns in MAS. Chapters 11 and 12 conclude the dissertation by answering the research questions and presenting future work.

1.6 Contributions within this Dissertation

As described beforehand, the dissertation connects different research conducted over five years. Each presented chapter contains contributions, while the entire dissertation provides the connecting thread between the elements. The contributions are shortly summarised in the following:

- A systematic literature review aimed to provide insights into the different aspects of self-organisation and emergence and provide the foundation for the dissertation.
- Identifying the possibility of connecting emergent behaviour in MAS and industrial systems with Kronecker Algebra by utilising formal languages.
- Present a method to identify implied scenarios in MAS based on Kronecker Algebra. In addition, introduce a new Kronecker operation called Kronecker Synthesize.
- Present a method to ensure consistency between state machines and given agent scenarios. In addition, introduce a new Kronecker operation called Kronecker Symmetric Skip.
- Present a method to include priorities in Kronecker Algebra.
- Present a method to calculate the worst-case execution time (WCET) of prioritised processes on a single-core central processing unit (CPU) based on Kronecker Algebra.
- Present first results of an especially build time-predictable real-time capable publish-subscribe communication environment.
- Show that Kronecker Algebra can be utilised to identify pattern formations in MAS.

1.7 Publications

In the duration of this PhD, some publications were created, and the following contributed directly to the dissertation:

- P. Denzler, J. Blieberger, and W. Kastner, *"Utilising Kronecker Algebra to Detect Unexpected Behaviour in Distributed Systems,"* in *2022 IEEE 25th International Symposium On Real-Time Distributed Computing (ISORC)*, pp. 1–8, 2022. [DBK22]
- P. Denzler, M. Ashjaei, T. Frühwirth, V. N. Ebirim, and W. Kastner, *"Concurrent OPC UA information model access, enabling real-time OPC UA PubSub,"* in *2022 IEEE 27th International Conference on Emerging Technologies and Factory Automation (ETFA)*, pp. 1–4, 2022. [DAF+22]

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

- P. Denzler, T. Frühwirth, D. Scheuchenstuhl, M. Schoeberl, and W. Kastner, *"Timing Analysis of TSN- Enabled OPC UA PubSub,"* in *2022 IEEE 18th International Conference on Factory Communication Systems (WFCS)*, pp. 1–8, 2022. [DFS+22]

- P. Denzler, T. Frühwirth, A. Kirchberger, M. Schoeberl, and W. Kastner, *Static Timing Analysis of OPC UA PubSub,"* in *2021 17th IEEE International Conference on Factory Communication Systems (WFCS)*, pp. 167–174, 2021. [DFK+21b]

- P. Denzler, T. Frühwirth, A. Kirchberger, M. Schoeberl, and W. Kastner, *"Experiences from Adjusting Industrial Software for Worst-Case Execution Time Analysis,"* in *2021 IEEE 24th International Symposium on Real-Time Distributed Computing (ISORC)*, pp. 62–70, 2021. [DFK+21a]

Other publications below are contributions to FORA or results of research collaborations:

- P. Denzler and W. Kastner, *Reference Architectures for Closing the IT/OT Gap*, pp. 95–123. Berlin, Heidelberg: Springer Berlin Heidelberg, 2023. [DK23]

- P. Denzler, D. Ramsauer, T. Preindl, W. Kastner, and A. Gschnitzer, *"Comparing Different Persistent Storage Approaches for Containerized Stateful Applications,"* in *2022 IEEE 27th International Conference on Emerging Technologies and Factory Automation (ETFA)*, pp. 1–8, 2022. [DRP+22]

- P. Denzler, D. Ramsauer, and W. Kastner, *"Model-driven Engineering of Gateways for Industrial Automation,"* Automation, Robotics & Communications for Industry 4.0, p. 47, 2021. [DRK21a]

- P. Denzler, D. Scheuchenstuhl, D. Ramsauer, and W. Kastner, *"Modelling protocol gateways for cyber-physical systems using Architecture Analysis & Design Language,"* *Procedia CIRP*, vol. 104, pp. 1339– 1344, 2021. 54th CIRP CMS 2021 - Towards Digitalized Manufacturing 4.0. [DSRK21]

- P. Denzler, S. Hollerer, T. Frühwirth, and W. Kastner, *"Identification of security threats, safety hazards, and interdependencies in industrial edge computing,"* in *2021 IEEE/ACM Symposium on Edge Computing (SEC)*, pp. 397–402, 2021. [DHFK21]

- P. Denzler, D. Ramsauer, and W. Kastner, *"Tunnelling and Mirroring Operational Technology Data with IP-based Middlewares,"* in *2021 22nd IEEE International Conference on Industrial Technology (ICIT)*, vol. 1, pp. 1205–1210, 2021. [DRK21b]

- P. Denzler, D. Ramsauer, T. Preindl, and W. Kastner, *"Communication and container reconfiguration for cyber-physical production systems,"* in *2021 26th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, pp. 1–8, 2021. [DRPK21]

- P. Denzler, J. Ruh, M. Kadar, C. Avasalcai, and W. Kastner, *"Towards Consolidating Industrial Use Cases on a Common Fog Computing Platform,"* in *2020 25th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, vol. 1, pp. 172–179, 2020. [DRK+20]

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

1.8 Reference to FORA

FORA—Fog Computing for Robotics and Industrial Automation was a European Training Network (ETN), which funded and trained 15 PhD candidates in the area of Fog Computing, during the period 2017-2021. Fog Computing, also sometimes called Edge Computing, brings the Cloud "closer to the ground", to the edge of the network. FORA was an interdisciplinary, international, intersectoral network that trained the next generation of researchers in Fog Computing with applicability to industrial automation and manufacturing. The wider aim was to pave the way for the a new industrial revolution, Industry 4.0, which will only become a reality through the convergence of Operational and Information Technologies (OT & IT), and this convergence will be achieved through Fog Computing. The funding came from European Union's Marie Skłodowska-Curie Actions (MSCA) European Training Networks (ETN) instrument part of the Horizon 2020 research program (Grant agreement No. 764785). The consortium was formed of 4 universities and 3 companies from Sweden, Denmark, Austria and Germany. Some outputs were a reference system architecture for Fog Computing; resource management mechanisms and middleware for deploying mixed-criticality applications in the Fog; safety and security assurance; service-oriented application modelling and real-time machine learning. FORA's researchers received integrated training across key areas (computer science, electrical engineering, control engineering, industrial automation, applied mathematics and data science) necessary to fully realise the potential of Fog Computing for Industry 4.0 and had moved between academic and industrial environments to promote interdisciplinary and intersectoral learning [BP23].

Figure 1.4: FORA: http://www.fora-etn.eu/

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Chapter 2

Complex Systems, Self-Organisation and Emergence

Within this chapter, we approach the challenge of presenting a coherent summary of the context of self-organisation and emergence. Covering the entire spectrum is unattainable as both phenomena have been recurrently a topic in academic research. Therefore the chapter builds upon the findings of a systematic literature review. It follows the idea of presenting a broader perspective first and later narrowing down on aspects of emergence relevant to this dissertation. Chapter 3 contains more details about how the systematic literature review was conducted. We introduce complex systems and their relation to self-organisation and emergence as a starting point, followed by a broader introduction to self-organisation and emergence and their similarities and differences. The rest of the chapter presents different aspects, viewpoints and research fields of emergence.

2.1 Complex Systems

Finding a universally agreed-upon definition of what constitutes a complex system is a rather challenging endeavour. There is a wide range of definitions and research areas (e.g., cyber-physical systems, system of systems [MRM17]) that focus on different types of complex systems [LLW13, TDM18]. However, there is a common conception among researchers in complex systems science that such a system should have the following features (see, for example [Hol98, BY97]):

- They consist of interacting entities, processes or agents.
- The interacting entities create high-level structures, patterns and behaviours. This particular feature is often summarised as "emergence".
- The higher-level behaviours are not simply predictable from the single entity's behaviour alone. There is also a stronger opinion that it should not be possible at all to predict high-level behaviours based on the behaviour of the system components.
- The interactions between the entities cannot be trivial, as even minor differences in local or initial conditions can create more significant system-wide effects. For example, linear changes at the entity level can give rise to non-linear effects at the system level.

The literature often summarises those features under three key constructs: complexity, self-organisation, and emergence [Mit13].

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

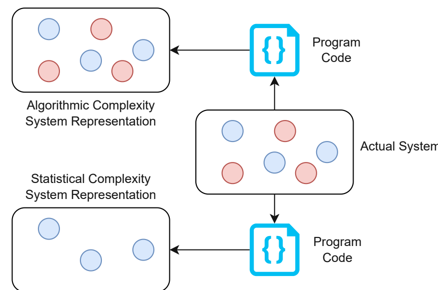


Figure 2.1: Algorithmic complexity is the length of the minimal program that can generate the actual system/object, whereas statistical complexity is the length of the minimal program that can generate the statistically significant aspects of the system. Figure adjusted from [Cha03, SK14].

2.1.1 Measures of Complexity

The term complexity describes a vital construct in complex systems research to address the previously indicated possibility for a discrepancy (expressed as non-linearity) between the summarised complexity of the entities and the complexity of the system. Measuring the complexity of a system is another vast topic that created various attempts to define an accurate measure [SSH04]. In essence, measuring complexity represents the idea that the more complex an entity is, the more information is required to describe or reproduce it. For complex computational systems, two forms of complexity measures are predominant:

1. Algorithmic complexity [Cha66, Cha74, Cha03], which is the length of the minimal Universal Turing Machine program which can describe/reproduce the entity; and

2. Statistical complexity [BCFV02] is the length of the minimum program able to reproduce the statistically significant features of an entity. It is calculated by reconstructing a minimal model containing the collection of all situations with a similar specific probabilistic future and measuring the probability distribution of the states (There are various algorithms for determining this for different numbers of dimensions, e.g., for one-dimensional time series [Sha03] and two-dimensional time series [SK14]).

Other complexity measures, as presented in Table 2.1, include, for example, design or grammar size and connectivity. Those measures could be interpreted as more specific algorithmic and statistical complexity formulations.

Algorithmic and statistical complexity differ in how randomness is treated, i.e., algorithmic complexity defines the information content of an individual sequence. In contrast, statistical complexity refers to an ensemble of sequences generated by a particular source (cf. Figure 2.1). If the source of a system's states is random, even if it can display a more significant number of configurations, their distinction is not statistically significant. Some definitions of emergence (cf. Section 2.3) use a form of complexity definition; however, these seem to be more informal or abstract and, therefore, not classified. The authors Bonabeau and Dessalles [BD97] define complexity as the set of detectors required to detect the entity and the tools that allow the description of the structures to be computed. Transferring that definition into an algorithmic interpretation would mean that the detectors can detect all features of the entity. From a statistical viewpoint, the same interpretation would lead to the detectors only detecting the statistically significant features.

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Table 2.1: System and design complexity measures.

Complexity measure	Definition
Algorithmic Complexity	Number of symbols of the shortest program that produces an object. The algorithmic complexity value is always an approximation since it is impossible to determine with certainty what the minimal program would be, e.g., [Cha66].
Statistical Complexity	Number of symbols of the shortest program that produces the statistically significant features of an object. E.g., [Sha03, SK14].
Connectivity	The number of edges that can be removed before the graph representing the object is split into two separate graphs. E.g., [TSE94, Edm99].
System Structure and Organisation	Function of the degree of connectivity between and within subsets of components. E.g., [TSE94].
Design Size	The length in symbols of the assembly procedure for an object. E.g., [Hor07]
Logical Depth	Computational complexity of an object's assembly procedure, i.e., the time it takes to compute the assembly procedure. E.g., [Ben86]
Sophistication	The number of control symbols in the program that generates the object. E.g., [Kop87].
Grammar Size	The number of production rules in the program required to produce an object. E.g., [Edm99].
Design Structure and Organisation	Function of the number of modules, the reuse of these modules and the degree of nesting. E.g., [Hor07].

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

2.2 Self-Organisation

The second construct of complex systems, self-organisation, comes as well with many possible definitions. Therefore, understanding the origins of self-organisation might shed some light on it. The following historical background does not aim to be complete; it is a summary of repeatedly mentioned accounts in several papers and PhD theses. For a more detailed historical overview, the interested reader is guided to Schalizi [SK14].

2.2.1 A Bit of History about Self-Organisation

Let us start with an often-quoted intuitive definition of self-organisation by Dempster: *"Self-organization refers to exactly what is suggested: systems that appear to organize themselves without external direction, manipulation, or control."* [Dem98, p.41]. Organisation in this context refers to an increase of structure or order of the system's behaviour; the system grows in organisation [DWH05]. Possible examples of such self-organisation are agent networks that autonomously build their structure because the entities can discover each other.

There is a broad agreement that the notion of spontaneous, dynamically produced organisation has existed for centuries. However, the term "self-organisation" that describes the phenomenon appeared much later. First accounts appeared in the aftermath of the Second World War [SK14], when research areas such as cybernetics and computing machinery introduced the term in their work [YC60, Ash62]. According to Schalizi [SK14], the author W. Ross Ashby [Ash47] coined the term in 1947 in the publication "*Principles of the Self-Organizing Dynamic System*." In this paper, Ashby explained what "organisation" is: namely that the organisation of a system is the functional dependence of its future state on its present state and its (if existing) external inputs. He pointed out that for a system to be self-organising, the system itself changes its organisation, not under the influence of an external entity.

Later, self-organisation was picked up and studied in other research areas such as physics, computer science, and systems theory. Pattern formation [Bal01], spontaneous symmetry breaking [NP77], or cooperative phenomena [Hak78] are only a few examples that have been extensively explored in physics since the 1970s. The intense growth of interest also led to disputes within the community. According to Frisch [Fri95], there is a long-standing dispute about the view of Klimontovich [Kli91], who claimed that the transition from lamellar to turbulent flow is an instance of self-organisation. A central argument of the opponents of this viewpoint is that there is a widespread misconception about "self-organisation". Researchers in computer science focused, for example, on adaptation [Hol75, FP86], "emergent" or distributed computation [For90, CM95] or learning [Sel59, YC60]. In economics [Kru96], and ecology [Art90, CDF+20], self-organisation has also become a trend, including new disputes about if specific processes are self-organising. Table 2.2 provides a brief overview of areas of self-organisation.

Later in 1980, self-organisation merged with other ideas, models and techniques to the "sciences of complexity" [Pag88]. This combination opened the door to other sciences where researchers adopted and applied the idea of self-organisation. The most relevant discipline for this dissertation is multi-agent systems because they can be used to model self-organising systems. A wide range of methods, such as cooperation [Ste90] and group formation [MRS02], can create more organised multi-agent systems. Examples of self-organising applications are found in networks [FH03] and robotics [KMRF+03] or communication between agents [Oud99].

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Table 2.2: Areas of self-organisation found in the literature.

Area	Relevant authors
Biology	[DM11, Fel06]
Physics	[Nic93]
Chemistry	[NF06]
Computer Science	[XWWW03, SJ02, HMS+20, Dut12, MZ06, VM13, MG18, CBdSMP16]
Economics	[Wit97, IMA+14]
Control Science	[KZV95]
Manufacturing	[BEL10]

2.2.2 Defining Self-Organisation

As indicated beforehand, there are many definitions of the concept of self-organisation. After revisiting several proposed definitions, the one given by DeWolf and Holvoet [DWH05] is closest to the one coined initially by W. Ross Ashby [Ash47]. The authors emphasised adding "autonomy" and "increase in structure" to their definition to make a more apparent distinction to emergence.

> *"Self-organisation is a dynamical and adaptive process where systems acquire and maintain structure themselves, without external control." [DWH05, p.7]*

The authors mention that the "structure" part can be spatial, temporal or functional. In contrast, the "without external control" part refers to the absence of any external intervention such as direction, manipulation, interference, pressures or involvement. Moreover, the authors stress that their definition does not exclude inputs outside the system as long as they do not contain control instructions. Another point the authors emphasise is the importance of "identifying" the system's "boundaries" when deciding if a system is self-organising. In addition, other authors specified several characteristics of self-organising systems. Note: Some characteristics are similar to those presented in Section 2.1 but explained in the context of self-organisation.

2.2.3 Characteristics of Self-Organisation

As mentioned by Dempster [Dem98], "organisation" is a vital part of the self-organisation concept. In [CAL], the organisation is *"the arrangement of selected parts to promote a specific function"*. Therefore, the system's behaviour is restricted to a subset of its state space. [CAL] calls this subset or region of the state space an attractor. In other words, organisation is an increase in the order of the system's behaviour that allows the system to have a spatial, temporal, or functional structure. It has to be mentioned that not every increase in order is automatically self-organising; there is a need for autonomy, as explained later on.

Shalizi [Sha01] formally approaches self-organisation. In his work, he uses statistical complexity to define order. He postulates that an increase in statistical complexity is essential for self-organisation. As described in Section 2.1 statistical complexity measures the average amount of historical memory stored in the process. The information-theoretic perspective of Shalizi's approach is presented later on, as it also covers several other definitions in the literature, e.g., "the arrangement of selected parts" implies that the arrangement is a kind of historical memory of the process that becomes bigger when more and more parts are arranged.

Another aspect of order increase is that it implies that a system can start from either a semi-organised or completely random state (i.e., no historical memory) [KMRF+03]. This further im-

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

plies that systems can decrease in order (i.e., lose historical memory) while becoming less ordered. Through the possibility that a system can increase or decrease in order, the process of self-organisation should follow the same notion.

There is more to address in the definition of [CAL], namely the part "as to promote a specific function". As a system without any order cannot exhibit a useful function, the same applies to a system with too much order. The thought that entities organise themselves with such a high complexity that no proper function can result from it is not that difficult. Expressed by the concept of historical memory, there is too much of it. According to [Lan90, Kau93], systems need to be between no order (chaos) and too much order to show proper self-organised behaviour. Other relevant research about the order in systems is done by: [Hak98, Hey02, Hey89, FH03, VDPB01, VDPB04]. Further characteristics mentioned in the literature are:

- **Autonomy**: The characteristic of increased order is tempting to believe that every increase of order equals self-organisation [Sha01, CDF+20, VDPB04, KMRF+03]. That is not the case, as there is a need for the "self", meaning the absence of external control. The system needs to organise itself without the interference of outside entities [Hey02, Hey89].

  As previously mentioned, outside control and autonomy do not exclude other types of inputs [Hak98, CAL, FH03]. A wide range of system inputs is possible as long they do not contain any control instructions [DWH05]. The systems entities should autonomously decide what to do next, i.e., how to process the input and what actions to take. A simplified example could be onboarding a sensor node into a network without user intervention. However, defining the boundaries of the self-organising system becomes relevant, as only parts of systems could be self-organising while others are not, e.g., some robots organise the transportation of parts themselves while others only passively produce parts [DWH05].

- **Adaptability or Robustness w.r.t. Changes**: In the context of self-organising systems, robustness and adaptability refer to the system's ability to react to changes and disturbances. The expectation is that a self-organising system can maintain its organisation autonomously during a change [SK14]. Goldstein [Gol99] expresses this characteristic as a self-generated adaptable behaviour while Foukia and Hassas [FH03] point out the usefulness of knowing past system states. Other authors, such as Mostefaoui et al. [KMRF+03], elaborate on the effects of environmental changes on the systems tasks while not affecting the internal entities.

  An adaptable system, therefore, can or needs to exhibit many behaviours. The system organises itself towards a specific attractor in its state space to provide the respective behaviour. Attractors can provide either one behaviour, periodic behaviour or a wide variety of behaviours [Hey02]. However, an attractor that provides too many possible behaviours makes the system uncontrollable and chaotic, while a too low variety reduces the system's flexibility [Lan90, Kau93]. It is crucial to maintain a balanced adaptability [VDPB04].

- **Dynamical, i.e. Far-From-Equilibrium**: Another neglected aspect of a self-organising system is that every system exists in time. The increase in order happens over time and not suddenly; it needs to be dynamic as it requires time to adjust to the changes [Sha01, Gol99]. In other words, the dynamic characteristic is required as changes permanently force the system to maintain its structure [Hey89, Hey02]. Authors such as Prigogine [GP71] see a far-from-equilibrium system as an essential requirement for self-organisation. Such a system's state makes it more dynamic and capable of reacting to changes, but also very fragile and sensitive.

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

2.2.4 Self-Organization From an Information Perspective

Before turning to emergence, the most crucial characteristic of self-organisation, "organisation" can also be described formally. As shown by [Sha01, SSH04, SK14], organisation entails an increase in statistical complexity. In self-organising systems, this is reflected through the information dynamics themselves.

According to Shalizi [Sha01], the increase in complexity reflects an increase in the predictive information $I_{pred}(T, T')$ within the system. If $P(x_{future})$ is a prior probability distribution for the futures and $P(x_{future}|x_{past})$, the average predictive information is:

$$I_{pred}(T, T') = \left\langle \log_2 \frac{P(x_{future}|x_{past})}{P(x_{future})} \right\rangle \tag{2.1}$$

where $\langle .. \rangle$ denotes an average over the joint distribution of the past and future $P(x_{future}|x_{past})$, T is the length of the observed data stream in the past, and T' is the length of the data stream that will be observed in the future. This quantifies the information that the past provides about the future and captures the reduction in Shannon entropy:

$$I_{pred}(T, T') = H(T') - H(T|T') \tag{2.2}$$

where $H(T')$ is the entropy for the future and $H(T|T')$ is the entropy for the future given the past [SSH04]. An increase in predictive information can be interpreted as an increase in order since it means that knowing how the system has behaved up to this point gives us a better idea of how it will behave in the future, with absolute certainty as the upper limit [Sha01]. Granger [Gra69] formulated similar measures using regression modelling.

The robustness of a self-organised system can be defined in terms of its sensitivity to perturbations. A system is robust if it exhibits coordinated behaviour despite perturbations. In information-theoretic terms, the robustness of a system can be measured by the range of perturbations for which the increase in predictive information (reduction in Shannon entropy) holds.

2.3 Emergence

The third construct of complex systems, emergence, is commonly seen as a phenomenon where a system's global behaviour emerges from its entities' interactions. That is about all most papers state about the term emergence, usually followed by some examples like pheromone paths of ants or the swarming movement of a school of fish [CDF+20]. Indeed not a good definition for a phenomenon that involves many facets and sometimes produces astonishing and even beautiful results. As with self-organisation, a look back on the history of emergence provides a good starting point. The following overview is mainly a summary of accounts given in Goldstein [Gol99]. For more details, we guide the interested reader to the mentioned paper.

2.3.1 A Bit of History about Emergence

As self-organisation, emergence is an old research topic [Gol99]. The first similar notions to emergence are the "whole before its parts" concept (i.e., the global behaviour is more important than explaining how the system works based on local behaviour) or Gestalt (i.e., a pattern of elements unified as a whole that it cannot be described as the sum of its parts) and are as old as ancient Greek philosophy [Gol99]. However, emergence is not a pre-given entity; it dynamically arises over time. In late 1875, over 100 years ago, the English philosopher G.H. Lewes noted the difference between 'resultant' and 'emergent' chemical compounds and the dynamics of the latter [Lew75].

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

> "(...) although each effect is the *resultant* of its components, we cannot always trace the steps of the process, so as to see in the product the mode of operation of each factor. In the latter case, I propose to call the effect an *emergent*. It arises out of the combined agencies, but in a form which does not display the agents in action (...)." (italics added) [DWH04, p.2]

In the 1920s, emergence found ground in a movement known as emergent evolutionism [Gol99]. The movement used emergence as an argument against reductionism, which states that every system can be reduced to the sum of its parts. Nevertheless, there was little understanding of the required processes to transform lower-level inputs into higher-level outputs. A following movement, commonly known as complexity theory [Gol99], addressed this apparent lack of understanding. Emergence in complex systems has a long tradition and is found in several scientific and mathematical disciplines such as cybernetics, solid state/condensed matter physics, evolutionary biology, artificial intelligence or artificial life.[1] DeWolf and Holvoet [DWH05] mention four central schools of research in this context. Each school studies emergence from a different perspective:

- Complex adaptive systems theory, created at the Santa Fe Institute. Emergence is the underlying process that creates macro-level patterns (see [Hol98, Kau95] and [Lan86]).
- Nonlinear dynamical systems theory and Chaos theory use attractors to describe the system's behaviour towards its evolves [New96].
- The synergetic school studies emergence in physical systems and represents the idea of an order parameter to asses macro-level behaviour [Hak84].
- Far-from-equilibrium thermodynamics, created by Ilya Prigogine. Emergent phenomena are the dissipative structures arising at far-from-equilibrium conditions [Nic93].

In all schools, the concept of emergence has two relevant characteristics: A global behaviour that is intractable to its individual parts but still arises out of the interaction of even those parts.

2.3.2 Defining Emergence

In the literature, there is quite a joint agreement on defining emergence as such, compared to self-organisation. Therefore, we resort to the definition proposed by DeWolf and Holvoet [DWH05]:

> *"A system exhibits emergence when there are coherent emergents at the macro-level that dynamically arise from the interactions between the parts at the micro-level. Such emergents are novel w.r.t. the individual parts of the system." [DWH05, p.3]*

DeWolf and Holvoet's definition encompasses several aspects. First, emergence is the process that creates the conceptual term "emergent", which represents properties, behaviour structure or patterns. The second is, as with self-organisation, the importance of micro-macro-levels. The macro-level comprises the whole system, while the micro-level is the viewpoint of the individual entities representing the system's composition.

[1] The map of complexity sciences by Brien Castellani, provides an excellent overview and deep insights about complexity.

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

2.3.3 Where to find Emergence

Before going into the characteristics of emergence, the following overview lists research activities in the various scientific disciplines in which emergence occurs.

- **Complexity sciences:** As introduced before, complexity sciences are linked with emergence. Some more specific accounts for research in complexity science concerning emergence are found in systems thinking [Sta00, CF02]. According to Monat and Gannon [MG15], emergence is essential for systems thinking. Referring to Stacey et al.[Sta00] systems thinking is related to systems engineering and provides perspective, language and tools to describe the relationships between entities of a system. Other accounts in systems thinking mention the existence of emergent and self-organising behaviours in systems with multiple feedback loops [MG15]. In the paper from Checkland [Che00b], the author uses emergent properties in real-world situations when introducing a soft systems methodology.

- **Nature and biology:** Other prominent research fields are nature and biology. While not going too deep into it, some phenomena in nature cannot be deduced by exploration [HW06] and show emergent characteristics. Organisation presents itself in various forms and facets in the natural world. For instance, we find organisational structures and processes in living organisms that can be classified as emergence. That includes biological functions, such as the ones of organs, that only function as part of the whole organism. It is challenging to deduct the processes in one organ without knowing the environment [KC06]. Another instance of emergence is Darwinian evolution, while the question of "what is life", approached by Schrödinger in 1944, points towards self-organising and emergent processes [NF06, Pro11]. Also, Pross [Pro11] mentions that the question of life can only be answered if one understands the principles governing its emergence.

  Protein folding is another example where self-organising and emergence properties appear. Simplified proteins are the building blocks of organelles, cells, organs, and organisms. Those linear strings of amino acids form folded three-dimensional structures. The interactions between acids lead to the emergence of these structures [NF06].

  The book of Camazine et al. [CDF+20] presents several examples of emergent pattern formation in non- and living biological systems. Camazine et al. mention that living biological systems' patterns arise without genetic coding but have developed through natural selection. The authors bring up the example of the stripes of a zebra and present within the book several groupings of natural systems, divided in:

  - Non-living systems: Everything that does not involve life, such as Benard convection or dune ripples.
  - Biological systems: Patterns that occur primarily in animals, such as the stripes of the zebra or leopard.
  - Social systems (interactions): Patterns based on "social" interactions of animals such as ants, bees or wasps and mammalians like humans.
  - Social systems (movement): Patterns based on collective movements of animals such as fish swarms, insects or birds.

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

- **Sociology:** The phenomenon of emergence is an essential element in sociology. In this context, emergence encompasses the collective effects created by individuals that cannot be traced back to the actions of the individuals [Saw01]. The relationship between individuals and the community is called a micro-macro link, a fundamental element in sociology. Emergence illustrates downward causation in communities in the context of computational sociology [SG13]. The phenomenon only exists if the agents react and act on each other [Saw05] and also depends on the agents' communication language. Other authors report the effects of emergence in human and natural societies interactions [Cor02].

- **Economy:** Similarly, as in sociology, emergence can be found in economic processes. Research in this field mainly concerns the novelty aspect of emergence and proposes various definitions of the phenomenon. One example is Harper and Lewis [HL12] that describe the emergence of four groups of evolutionary economics. The authors see emerging skills, abilities, companies, networks, or customer priorities over time that originate within the economic system as examples of emergence [HE12].

- **Cognitive sciences and psychology:** Another closely related field is psychology. A central element in the roots of psychology is the concept/theory of organicism. According to the organicism theory, an organism differs from its components' sum [Saw02]. There is a larger body of research. Wundt [Wun12] presents one compelling thought; the author connects creativity with emergence. The author explains that each created product does not equal the sum of its elements separately; it also includes innovation and creativity [Saw02, Wun12].

- **Physics and chemistry:** Emergence appears in several sub-areas in physics and chemistry, as almost everything has a molecular or chemical basis. There is research concerning life, as in biology, with a different scope, such as investigating the role of RNA in Earth's life [Szo09]. Other research focuses on near-equilibrium systems, like gases in a closed container, exhibiting emergent properties not present in the components. Systems that are far from thermodynamic equilibrium can also show elements of self-organisation. Such systems reach a quasi-stable or meta-stable state and display a certain order level, resulting in structures like regular hexagonal convection cells with emergent properties [HW06].

  Furthermore, authors such as Fernández, Maldonado, and Gershenson [FMG14] argue that emergence has a spatial or temporal scale, for example, the colour and flexibility of gold that do not exist in single atoms [And72, GH05]. The examples continue that ammonia and hydrogen chloride are gases, but the combination results in a solid structure, i.e., the resulting material has a different feature. Sugar has a particular taste, but the building blocks, carbon, hydrogen, and oxygen, undoubtedly not [Ash56].

2.3.4 Characteristics of Emergence

Some of the examples above already indicated some characteristics of emergence. Nevertheless, the following characteristics are repeatedly mentioned and accepted in the literature.

- **Micro-Macro effect**: Certainly, the most mentioned characteristic of emergence is the Micro-Macro effect. This effect describes the properties, behaviours, structures, or patterns only present at a higher macro level that arises from the micro-level interactions of the system's entities [Gol99]. In the above definition, the term "emergent" represents those properties. The macro-level effect is also known as the global behaviour of the system. The authors in [Hol98, Cru94a, Cru94b, Hey89, Hey02, Ode02a, Ode02b, VDPB01, CDF+20, VDPBS01, VDPB04, CAL, Luc97] describe the effect consistently.

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

- **Radical Novelty**: The characteristic of radical novelty is closely related to the previous one. In essence, it describes the novelty of the global behaviour w.r.t. the individual behaviours at the micro-level. This means that the micro-level entities do not explicitly represent the global behaviour; by reductionist terms, the macro-level emergents are not reducible to the system's entities. Crutchfield [Cru94a, Cru94b] uses the formulation "not directly described by" or Heylighen [Hey89] "can not be reduced to". In contrast, Goldstein [Gol99] uses "neither predictable nor deducible from" or the all-time favourite of most papers, "the whole is greater than the sum of its parts" by Odell [Ode02a].

  Nevertheless, according to BarYam [BY97], the idea that the behaviour of the parts cannot capture emergents is misleading. Rather the emergence arises because there is no understanding of the collective behaviour of the parts. BarYam states that collective behaviour is implicitly contained in the behaviour of the parts if studied in the context of the system. Emergent properties cannot be studied by taking the system apart and examining the parts; they have to be studied in the context of the system as a whole. Other relevant authors are: [CDF+20, VDPB04, CAL, Luc97].

- **Coherence**: 'Organisational closure' or coherence refers to the logical and consistent correlation of parts [Hey02]. Emergents are a new whole and, according to [BY97], maintain over time their identity. In order words, emergents are persistent over time. Therefore the correlations between components are needed to reach a coherent whole [Gol99, Ode02b, Ode02a].

- **Interacting Parts**: For emergence to exist, the parts must interact [Ode02a, CDF+20, VDPBS01, Ode02b, Hey02].

- **Dynamical**: Emergents have a time component; they arise over time. Therefore, the system's behaviour changes to a new behaviour not right away. It is similar to dynamical systems where new attractors appear, i.e., bifurcations [Gol99, Hey02, Hol98, Ode02a, Cru94b].

- **Decentralised Control**: Any control must be limited to local mechanisms to influence the global behaviour of the system [Ode02a]. No centralised control unit (or a single entity) influences the system's macro-behaviour. It is only possible to control each unit but not the system as a whole. Accordingly to Odell [Ode02a] and Heylighen [Hey02], the decentralised control characteristic directly results from the previously described radical novelty.

- **Two-Way Link**: In essence, there is a need for the macro-level and the micro-level to interact bidirectionally [Ode02a]. The micro-level parts create an emergent structure, and the emergent structure influences its parts, i.e., the higher level has a downward causation [CAL, Luc97]. Camazine et al. [CDF+20] use the example of ant paths that influence the ants while being an emergent structure.

- **Robustness and Flexibility**: As no single entity has an idea or plan for the emergent, it cannot be a single point of failure. According to Odell [Ode02a], emergents are relatively robust regarding errors. There is no sudden loss of function but rather a slight decrease in performance and quality, correlating to the damage. That allows an emergent to remain even if an entity fails or is replaced. For example, birds in a flock or cars in a traffic jam can be replaced by other birds or cars, yet the flock and traffic jam phenomena remain [Ode02a, Hey02].

In summary, most researchers in the different disciplines accept those characteristics, which allows the differentiation to self-organisation. A topic discussed in Section 2.4.

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

2.3.5 Classifications of Emergence

The previously given characteristics are relatively general and give room for interpretation. Unsurprisingly, many have strived to introduce other means to measure or express an emergent phenomenon's size, extent or dimension. As a first delimitation, the following measures only include suggestions that can be empirically studied and computable [Bed03]. This decision is necessary to avoid getting trapped in discussions about emergence's ontological or metaphysical status. Also, we exclude emergent properties that involve substrate-specifics. They might be fascinating and can be empirically studied, but they are out of scope.[2] In addition, most of those phenomena cannot be computationally modelled [BG07] by the conventional Turing model of computation [Tur50]. Most of the metrics fit into one of the following two categories.

- Observer dependency:
  - **Design-subjective:** The emergent property/phenomenon is judged solely by the observer and his understanding of the design or set of rules underlying the system/phenomenon. Ronald et al. [RSC99] accept the occurrence of emergence if: (a) the language of design L_1 and the language of observation L_2 are distinct, and (b) the causal link between the elementary interactions programmed in L_1 and the behaviours observed in L_2 is nonobvious to the observer. Nevertheless, the elements L_2 and the non-obviousness to the observer are subjective. Further examples: Optimal means of prediction is a simulation in [Car89, Dar94].
  - **Partial-analytical:** The emergent property/phenomenon is judged relative to an observation or point of view, e.g., level, scale, scope, or resolution. The property is emergent if the previously set specific criteria can be analytically determined. For example, the flocking behaviour of Boids might be emergent because, by definition, it involves more than one Boid. [3] Further examples: A "whole" language is not reducible to the "sum of parts" of a language [Kub03] and deriving a macro-state from its micro-dynamic and system's external conditions but only by simulation [Bed03].
  - **Partial-empirical:** The emergent property/phenomenon is judged relative to an observation or point of view, e.g., level, scale, scope, or resolution. The property is emergent if the previously set specific criteria can be empirically determined at a particular resolution. For example, the flocking behaviour of Boids might be emergent because a single Boid's behaviour becomes easier to predict. Further examples: Redundancy of lower-level detectors in [BD97] and greater predictive efficiency [Cru94b, SC01, SSH04].
- Quantifiability (Measures):
  - **Categorical:** A property is either emergent or not. Examples: Redundancy of lower-level detectors in [BD97], and a "whole" language is not reducible to the "sum of parts" of a language [Kub03].
  - **Continuous-unquantifiable:** Properties can exhibit different degrees of emergence, but no established quantification method exists. Examples: Optimal means of prediction is a simulation in [Car89, Dar94], deriving a macro-state from its micro-dynamic and system's external conditions but only by simulation [Bed03] and Language of design L_1 and language of observation L_2 are distinct, and causal link between interactions programmed in L_1 and behaviours observed at L_2 is nonobvious [RSC99].

[2] The reader will agree that it has nothing to do with the intended industrial context of this dissertation.
[3] Boids is an artificial life simulation initially developed by Craig Reynolds.

TU WIEN Bibliothek Your knowledge hub
Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.

- **Continuous-quantifiable:** Properties can exhibit different degrees of emergence, and there is an established method for quantification: Greater predictive efficiency (Information Theory) [Cru94b, SC01, SSH04].

Other types of classifications sometimes overlap. For example, some authors classify how extensive or complex the emergent phenomena is as simple, spooky, complex, positive, negative, static dynamic, deterministic, stochastic, synchronic, diachronic and nominal emergence. It would go too far to elaborate on each of them; the interested reader is guided to Kalantari et al. [KNM20, p.13] for an overview of the various classifications with examples. The authors also include other classifications, such as time dependency, certainty, reducibility or self-organisation-based. Chapter 4 will introduce the classification of basic emergence as proposed by [Bed03].

2.3.6 The Design-observed Discrepancy

Section 2.3.5 introduced the characteristic of observer dependency, which needs further explanation. As mentioned beforehand, a significant characteristic of emergents is their novelty. It is possible to design the entities, but it is not easy to design emergent behaviour. To approach this difficulty, researchers propose different measurements depending on the viewpoint. In [RSC99], the components have a language L_1 they use to interact, and the system designer uses this language to design the system. An observer knows the design but uses a different L_2 language to describe the system's behaviour. The two languages, L_1 and L_2, are distinct, and if there is a non-obvious causal link between L_1 and L_2, Ronald et al. [RSC99] state that this is the proof for a behaviour to be emergent. However, there is a catch. How does the observer assess whether or not there is a non-obvious causal link between the L_1 and L_2 properties? Darley [Dar94] brings up a more objective criterion for an emergent property. According to this author, the only way to deduce if a behaviour is emergent is by stepping through the execution of the system that creates it. In essence, simulation is the solution to predict the system's behaviour.

Another possibility is to describe the discrepancy in terms of design complexity [MB89] and system complexity [Kin10] (see Figure 2.2). The complexity of a system in traditional design and engineering linearly reflects the complexity of the design. At the top of Figure 2.2 shows that the system's complexity can be established analytically from the design complexity. If C_{design} stands for the design complexity, C_{system} stands for the system complexity, and $C_{system} = f(C_{design})$, it is possible to define the function f based on the design.

In complex systems design that exploits emergent properties and behaviours, however, the connection between design and system complexity is not that simple [Kin10]. It is not possible to establish the relationship analytically from the design. In other words, f cannot be found based on C_{design}. The reason is that the relationships and interactions between the components are dynamic and changing in the way of the original design. The lower part of Figure 2.2 illustrates the dynamic changes of the system over time.

2.3.7 The Information Dynamics of Emergence

The information-theoretic approaches look at the system with different resolutions to identify emergence [PBR09]. It has been shown that emergence can occur when lower-resolution dynamics have greater predictive efficiency than higher-resolution dynamics. According to [PBR09] that should make it possible to predict the statistically significant features of the system's future. Predictive efficiency is defined as [CF03]:

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

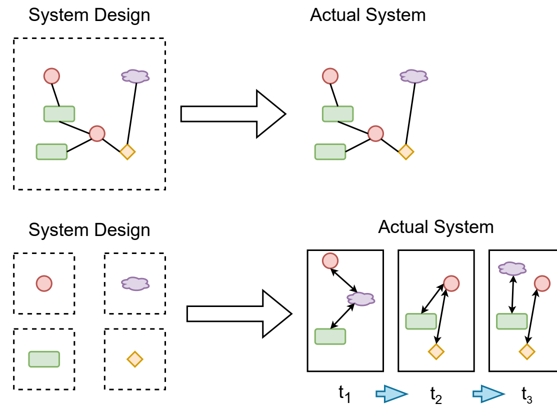


Figure 2.2: Design and system complexity.

$$\epsilon = \frac{E}{C_\mu}, \tag{2.3}$$

where ϵ is the predictive efficiency, E the excess entropy and C_μ the statistical complexity. The term excess entropy is a measure for the total apparent memory in a source [CF03]:

$$E = \sum_{L=1}^{\infty} (h_\mu(L) - h_\mu), \tag{2.4}$$

where the average uncertainty about the Lth symbol $h_\mu(L)$, provided the $(L-1)$ previous ones are given is:

$$h_\mu(L) = H(L) - H(L-1), L \geq 1 \tag{2.5}$$

for the entropy H(L) of length-L sequences, and

$$h_\mu = \lim_{h \to \infty} \frac{H(L)}{L} \tag{2.6}$$

is the source (per-symbol) entropy rate. [FC03] and [CF03] define the excess entropy E as the mutual information between the source's past and future. The excess entropy E is the amount of information observed in the past to predict the future. Continuing this definition, the authors see the best level of observation as the one that optimises *(relative to the particular problem at hand)* the trade-off between:

- reducing E, but losing predictability and gaining simplicity, i.e., reducing C_μ; and
- increasing E and losing simplicity (increasing C_μ).

Therefore, the information dynamics of a system can differ depending on the resolution of the observation. Each resolution has its own E, C_μ and ϵ. Returning to the statement of [PBR09] that emergence occurs when lower resolution dynamics have a greater ϵ than higher resolution dynamics. A system with a random behaviour, C_μ, directly proportional to E (ϵ is always constant) cannot produce emergent behaviour. For emergence, the difference in C_μ between higher and lower

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

resolution levels of description shall not linearly relate to the differences in E. E is disproportionately low relative to C_μ at low resolutions. Those interpretations of emergence and complexity lead to the following implications: The statistical information-theoretic interpretations of emergence and complexity have three important implications [CF03].

1. Emergence depends on both the observed system and the resolution of observation (A function).
2. The predictive efficiency ϵ can be determined for any two resolutions concerning the other.[4]
3. Theoretically, it is possible to quantify the degree of emergence by the value ϵ.

2.3.8 Macro-Properties, Scope and Resolution

Section 2.3.7 considers only one resolution and ignores the spatial dimension of the properties. [Rya07] describes the scope of a system as the set of components within the boundary between the associated system and its environment. Moreover, the author distinguishes between next to the (physical) spatial and temporal dimensions. The temporal dimension is the set of moments over the system resolution defined between two alternative system configurations. Therefore, it is possible to distinguish between (physical) spatial and temporal resolutions. Spatial resolution defines the size or distance between the system's different locations, and temporal resolution defines the duration of a moment in time. Using the Shannon entropy, a higher resolution can distinguish a more significant number of possibilities, n and hence has a greater value for H. The spatial scope is the set of location points occupied by the system.[5]

$$H = -\sum_{i=n}^{n} p_i \log(p_i) = \log(n) \tag{2.7}$$

2.4 Differences Between Self-Organisation and Emergence

Before discussing more details about methods proposed to identify emergent behaviours, it makes sense to point out the differences and similarities between self-organisation and emergence. *"Self-organisation is an adaptable behaviour that autonomously acquires and maintains an increased order (i.e., statistical complexity, structure, ...)"*, while *"emergence is the existence of a global behaviour that is novel w.r.t. the constituent parts of the system"* [DWH05, p.9].

2.4.1 Similarities

There are only a few similarities between self-organisation and emergence, as each focuses on different aspects of the system's behaviour. The main similarity is that both are dynamic processes created over time. They both share the capability of robustness, while self-organisation is robust w.r.t. change and the ability to maintain the increased order. On the other hand, emergence is more robust w.r.t. flexibility of its parts [CF03]. A failure of one part will not cause the emergent to disappear. Many authors explain the differences between the two concepts by combining them as they complement each other quite well [DWH05].

[4] If possible to measure E and C_μ.
[5] Spatial scope is not explicitly defined by [Rya07].

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

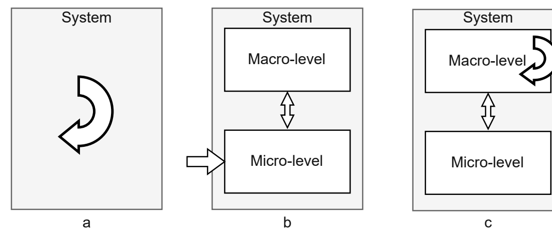


Figure 2.3: Different systems with either (a) only self-organisation, (b) emergence without self-organisation and (c) the combination of both. Adjusted from [DWH05].

2.4.2 Differences

This section first presents self-organisation without emergence and second, emergence without self-organisation, as both concepts can exist in isolation. The combination of both is presented later. There is a slight repetition from the previous sections; however, as there is always confusion about the differences, the benefits overweight the redundancy.

- **Self-Organisation without Emergence**: As a single concept, self-organisation has no control outside the system's boundaries or micro-macro effect. Figure 2.3 (a) represents this circumstance using a curved arrow, as only internal interactions exist. As mentioned in Section 2.3.4, there is a need for radical novelty, micro-macro effect, flexibility w.r.t. the entities, and decentralised control for emergence to arise. However, those requirements are not needed for self-organisation; thus, self-organisation can exist. Potential examples of such stand-alone self-organising systems are "classical" multi-agent systems [VDPB04]. Those systems are autonomous and increase their order through interactions. No novelty involved in such interactions, especially if each agent has a model of the global behaviour, i.e., the properties are present in the agents and not new.

  Systems without decentralised control, e.g., one agent with a plan of global behaviour and controls the interactions, cannot create any radical novelty either [DWH05]. Even the self-organising system dynamically re-elects the leader agent. Due to failure or task changes, the agent needs a plan; thus, no emergent property can arise. A leader agent, however, represents a single point of failure and therefore violates the ability of emergent systems to act in "graceful degradation" when an entity fails [DWH05]. One entity is not essential for an emergent system.

- **Emergence without Self-Organisation**: A pure emergent system would require increased order, no external control and adaptability [FH03]. Figure 2.3 (b) represents this circumstance by two internal fields, the micro and the macro level. There is a micro-macro effect but no self-organisation [DWH05].

  Examples of emergence without self-organisation are rare. However, some examples in physics and thermodynamics emerge from statistical mechanics in a stationary (and non-self-organising) system [Sha01]. The stationary system arises out of a "time-translation invariant process" without an increase in order [Sha01]. An often-presented example is a gas with a specific volume in space. The volume is the emergent property resulting from the interactions (i.e., attraction and repulsion) between the individual particles, but the gas itself is stationary. Also, the statistical complexity remains the same over time [Sha01].

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

2.4.3 Combining Emergence and Self-Organisation

One of the reasons there is some confusion about whether a system is self-organising or emergent is that most examples found in the literature are a combination of both. The multi-agent and adaptive systems community primarily focuses on such systems that are, for example, highly distributed, open, large, and situated in a dynamic context [VDPB04]. As for such systems, the single entity should be relatively simple (scalability), and the aim is to combine emergence and self-organisation to obtain a specific functionality. The reason is that self-organisation requires an increase in order to fulfil a specific functionality [DWH05]. However, a single entity cannot control the complex system; therefore, new global behaviour must emerge from the entities' interactions.

There is also the other way around, where a multi-agent system must exhibit emergent behaviour, but creating the necessary initial structure is impossible. In this case, the behaviour has to arise and organise autonomously. Such an approach is auspicious for engineering a coherent behaviour for complex (multi-agent) systems [DWH05].

If combined, Figure 2.3 (c) visualises the relations between self-organisation and emergence. According to Parunak [VDPB01, p.1], "... self-organising behaviour occurs at the macro-level". In other words, there is an increase in order on the global level [VDPB04], i.e., the emergents become increasingly organised. However, the author states that the system as a whole decreases its order. A surprise, but apparently, the reason is that at the micro-level of emergent systems, the dynamics are often very complicated and disordered. Also, Shalizi [Sha01, p.118] states that "... self-organization increases [statistical] complexity, while emergence, generally speaking, reduces it ...". Many authors see emergence in complex systems as the result of a self-organising process [Hey89, CDF$^+$20, HVZ$^+$04, Kau93]. Moreover, one more characteristic seems to appear when combining self-organisation and emergence, namely "nonlinearity".

Authors such as Glansdorff and Prigogine [GP71], Goldstein [Gol99] and Heylighen [Hey02] see a requirement "small cause, large effect" for a system without a priori order and self-organised emergence. This nonlinearity enables the secondary effects at the macro level (emergents) and is achieved through positive feedback that amplifies an initial change. The initial change then gets amplified again by the positive feedback of the previous amplification. This amplification will not continue forever, as most entities will have "aligned" themselves with the initial configuration created by the first change. The system has reached a point where no more resources are available, and the newly created alignment of all entities is the new emergent property [CDF$^+$20].

The mechanisms behind this nonlinear process are related to the properties of self-organisation. Glansdorff and Prigogine [GP71] state that one of the components in the system must exhibit auto-catalysis. Auto-catalysis describes the causal influence of one component with another to result in its increase. It is positive feedback causing a nonlinear effect. An example is the pheromone reinforcements by ants [CDF$^+$20]. Camazine et al. [CDF$^+$20] consider positive and negative feedback important for adaptive behaviour. This does not mean a negative feedback loop should complement every positive feedback loop. Sometimes, positive feedback can be needed to fight a never-ending leak that makes the structure disappear as soon as the positive feedback stops.

2.5 Unwanted Emergent Behaviour

Another critical issue is whether emergent behaviours in self-organising systems are desirable or undesirable. Predicting and determining the type (desirable or undesirable) of emergent behaviours in systems is essential as the positive effects can be harnessed, and undesirable outcomes can be prevented. Johnson [Joh06] mentioned that emergent properties could be beneficial or harmful in some contexts.

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

2.5.1 Types of Methods to Detect Emergent Behaviour in Systems

The methods proposed for identifying emergence seem to fall into three general groups: requirement based, mathematical and statistical concepts, and artificial intelligence based. In most of them, at least one external observer is necessary to identify emergence. In natural systems, the external observer identifies emergents by specifying changes in the behaviour and structure of natural agents (such as humans, insects, and birds) and swarms. Therefore, we also need measurements or requirements for evaluating the system's behaviour in artificial systems. Identifying emergence that occurs over time is done by examining changes in these measurements or requirements.

- **Requirement based**: Early systems design is one area where preventing unexpected emergent behaviour is crucial. Preventing unexpected emergent behaviour will reduce the effort in the later development stages. A typical method is defining system requirements or scenarios that can be checked while modelling the system. The developers identify and evaluate the system requirements [CLW92].

  The authors in Moshirpour et al. [MMBH10] focus in their paper on presenting an example where they identify potential flaws in the system by using implied scenarios. A scenario represents one system's behaviour, and several scenarios of the same system can create an implied scenario, i.e., a scenario not foreseen by the designer. Such implied scenarios are a potential cause for unwanted emergent behaviour. Another possibility to create implied scenarios is during transforming scenarios into state machines [MMEF12]. The same authors propose an algorithm that identifies emergent behaviour and prevents over-generalisation. However, they do not provide any example that the implied scenario creates emergent behaviour.

  Fard et al. [Hen13] follow a similar track and aim to identify emergent behaviours by storing message labels into interaction matrices. The matrices contain the component name, the sender, the receiver identifier (ID), and what are the time dependencies. The actual message content is irrelevant to the proposed method. The authors use a Markov chain to model the system's behaviour and identify paths among components of the scenarios. If a path is absent in the message sequence matrix, the system develops new behaviours through unwanted transitions from one state to another.

- **Mathematical**: Other authors propose methods based on mathematical and statistical techniques to identify systems' emergence. A benefit of such methods is that they are replicable but are limited to one specific set of emergent behaviour.

  O'Toole, Nallur, and Clarke [ONC17] describe an algorithm to detect emergent events in complex adaptive systems. They utilise a form of distributed consensus, where several agents detect and decide whether a systems change is an emergent event. The same authors [ONC14] analyse the behaviour of complex adaptive systems in a decentralised manner. Here is the system feedback from the macro and micro levels of the components, the relevant measure for determining emergent behaviour. In detail, the macro-level emergent behaviour affects the micro-level by limiting the components' behaviour or environment. Several variables are recorded and statistically analysed for correlations using each agent's local information. The authors argue that emergence creates those statistical correlations as they are not observed before.

  Another approach proposes the use of Semi-Boolean algebra. In Haglich et al. [HRP10], the authors use Semi-Boolean algebra to identify and predict emergent or self-organising behaviours in extended social networks, such as money laundering or smuggling networks. The authors argue that combining several social networks increases total complexity and the

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

potential for emergent behaviour to appear. In a later paper [HPR10], the authors present an adjusted version for multi-sensory situations and mention that operational complement, Partially Ordering Set (POSET), equivalent classes, and order axioms are methods suitable to describe emergent behaviours.

Chen et al. [CCN10] propose a formal method approach to characterise and examine emergent behaviours in complex agent-based simulations. The authors characterise various behaviours according to different abstraction levels, which allows the examination of relations between the levels, i.e., between higher-level behaviours and lower-level events. The relatively complex framework utilises an X-machine similar to Petri nets, state diagrams, and sequence diagrams to model higher-level emergent behaviours and the transitions of agent-level states.

The mathematical methods also encompass model checking and runtime verification. These methods will be introduced in Chapter 7.

- **AI based**: The last group contains methods based on artificial intelligence techniques such as clustering and machine learning. Clustering refers to methods focusing on visual or content-based separation of emergent patterns. Pattern detection can either be a spatial, temporal or content distribution. Machine learning makes it possible to find emergence based on previous events or data. For example, Grossman et al. [GSG+09], a system called "Angle", identifies emergent behaviours in distributed Internet Protocol (IP) data packets. The system clusters the data packages chronologically, and changes in the clusters indicate emergence. In addition, the statistical analysis identifies new types of behaviours.

  Other authors, such as Denzinger et al. [DK06], use evolutionary algorithms to identify unwanted emergent behaviours in multi-agent systems. In essence, the system contains attack agents and test agents to create events in the system. The attack agents create sequential activities that lead to a sequence of environmental states. The test agents then apply environmental state conditions to find unwanted emergent behaviour. According to the authors, the evolutionary algorithm represents a form of desired emergent behaviour. In Villani et al. [VFB+13], the authors look for dynamic structures of emergence in dynamic networks. Their method uses a cluster index measure to identify clusters in biological neural networks. Gomez et al. [GSZ17] introduce a quantitative definition of emergence, where subsystems of a complex system are observed. The proposed algorithm automatically detects emergent properties using supervised machine learning techniques and learns about the dynamics of the subsystems to detect emergent properties.

In summary, researchers propose several methods to detect emergent behaviour in systems. However, some approaches fail to determine if the detected behaviour is emergent. A topic other researchers have studied in detail.

2.5.2 How to confirm Emergent Behaviour

Several academic publications look into validating if emergence exists in natural and artificial systems [CDF+20, LSHL06, MG15, ZPK00] by considering the different viewpoints on emergence (cf. Section 2.3.4). The following paragraphs extend the more general view on emergence.

Szabo and Teo [ST13] divide emergence verification into grammar-based, variable-based, and event-based methods. The authors take the standpoint that system designers must know what type of behaviour they can expect and which are emergent. There is a lack of knowledge among designers or users unfamiliar with the principles of agent interactions on how to differentiate emergence from other behaviour. The authors used a flock of birds model to evaluate the various methods.

TU WIEN Bibliothek Your knowledge hub
Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.

An example of a grammar-based method is done by Kubí [Kub03]. The method compares different languages, namely the language of the system and the sum of the component languages. If there is a difference, emergence is validated. In a variable-based method, a variable describes the emergence, and any change validates the presence of emergent features. Set [Set08] presents the changes at the centre of a group of birds as an indicator of emergent behaviour. Event-based methods analyse a series of events that change the state of a system or subsystem [CNC07]. Chen et al. [CNC07] argue that those sequences are the emergent behaviour of the system if there is no evident connection between the macro-level and interactions of the subsystem. The authors use simulation to create the event sequences and the later validation of emergence.

Szabo and Teo [ST13] provided an objective-based method in their paper. The method uses semantic validation to prove emergent behaviours. The authors describe a system in terms of goals and objectives, and if behaviours occur that are not part of the goals and objectives, those have the potential to be emergent behaviours. Similarly, Gore et al. [GRTB07, GR08] an explanation-exploration method. Here the authors use a semi-automatic model adaptation and a random inference procedure that reveal interactions that lead to emergent behaviour. They also point out the method's suitability to detect unwanted emergent behaviour in the early design phase.

In robotics, validation plays an essential role in creating swarm behaviour. Rouff et al. [RVH+04b] present a formal method to verify and validate group missions. The authors highlight the benefit of formal methods to accurately and adequately prove whether group behaviour is emergent. Other research applies model checking to prove the emergent behaviours of robot swarms [JV11, DWF11]. Model checking is a formal alternative for testing [CJGK+18]. De Angelis and Serugendo [DADMS15] utilise a logic language to verify graph-based topology-dependent emergent properties at runtime. Their approach allows the decomposition of global behaviours into several local properties. The interface operators verify the emergent behaviour.

As mentioned in Section 2.5.1, formal and mathematical methods strongly focus on identifying and validating emergent behaviour. However, representing agents, their behaviour and interactions must be exact. Therefore it is challenging to model complex and dynamic systems accurately. Primarily representing the behavioural interactions of complex interactive systems in equations and verifying the existence of emergence is sometimes challenging when not impossible.

2.5.3 How to Simulate and Model Emergent Behaviour

After looking into how to confirm emergent behaviour, another interesting question is how to create or simulate emergent behaviour in systems. Epstein [Eps99] and Desalles [DP06] promote a clear distinction between natural systems and philosophical discussions about emergence. Furthermore, the authors point out that to simulate emergence; there must be a clear relationship between the phenomenon and a model. The model needs to consist of elements and relations that allow emergence to appear; therefore, emergence is epiphenomenal relative to that model. Epiphenomenal describes a phenomenon that is not dependent on the underlying interactions.

Tolk [Tol19] promotes using computer simulations to study complex adaptive systems. Such computer simulations can produce, discover and describe emergent properties. However, Tolk also points out the limitations of simulation; namely, it is impossible to create ontological emergence. Ontological emergence is knowledge independent and cannot be described by components and their interactions and relations. In the literature, several simulation approaches follow a method to create simulation models, for example, distributed systems [DWHS06]. The models include an environment, the entities and the possibility of observing both. Each entity performs its tasks over time, and the simulation provides a history of measurements related to the interactions [VDPSR98]. The following paragraphs present three types of modelling approaches found in the literature.

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

- **Agent-based Modelling and Simulation**: The first approach is agent-based modelling (ABM). As with any other type of simulation, the aim is to understand why and how a system shows emergent behaviour [MFT05]. The simulation environment continuously evaluates the simulated systems [RB95] and allows getting detailed information on phenomena and dynamic relationships that are not explicitly present in the code. This capability leads most authors to suggest that simulation is valuable for studying emergence [RB95].

  In detail, a simulation model replaces the system in ABM. Each model consists of agents that imitate the system's entities' behaviours. The entities either interact directly or indirectly through the environment [VDPSR98]. Unsurprisingly, authors in self-organising systems research favour ABM as it allows a deeper understanding of the phenomena [MFT05]. Chan [Cha11] points out the flexibility in simulating the interaction of autonomous agents and that ABM is an essential tool in finding emergent behaviours of complex systems. Moreover, De Wolf, Holvoet, and Samaey [DWHS06] state the simplicity of simulation compared to other approaches. A one-to-one mapping between the system and the model allows high degrees of localisation and distribution. Therefore the models can imitate self-organising emergent systems.

  Simulation also contributes to studies in complex systems [Bak10] as the fields are interconnected [Bed03]. In the paper of Baker [Bak10], the authors use simulation to create emergent features in a system. Similarly, in Mittal [Mit19], the authors focus on synthetic emergence and use modelling and simulation to study emergence. The authors call synthetic emergence emergent behaviour created in a controlled artificial environment. Moreover, a controlled artificial environment helps control complex systems' emergence. Other authors, such as Madey [MFT05], consider agent-based modelling suitable for understanding self-organising systems' temporal dynamics and emergent features. Boschetti [BPMG05] emphasises simulation's usefulness in identifying and modelling emergence in communication networks.

  Other authors such as Zeigler and Muzy [ZM16] utilise modelling methods such as Discrete Event System Specification to find emergent behaviours. At the same time, Weyns and Holvoet [WH02] present an approach based on coloured Petri nets to represent multi-agent systems. Their focus is on the evolution of the agents and the environment, which allows the modelling of social skills and coordinated social behaviours. Moncion et al. [MAH10] show that particular circumstances allow agent-oriented simulations to show emergence. The authors show that simulated actions and reactions among lower-level entities can create emergent behaviour and be identified by simulation. Moreover, the authors use dynamic graphs to store and analyse the interactions.

  Chen, Nagl, and Clack [CNC07] present another method to simulate emergent behaviour in agent systems. In their approach, the authors form complex events based on interrelated events. The complex events surface at different spatial and temporal levels. The authors use directed graphs with coloured vertices and edges to describe the complex events. The edges represent the relationship types, and the coloured vertices the types of events. According to the authors, it is possible to identify the type of complex events by analysing the subgraphs in the main graph. Each event type that creates emergent behaviour is classified and stands for one specific emergent behaviour. One interesting part of this approach is that higher-level behaviours can be decomposed into lower-level events, which allows for predicting the effect an agent change creates on other levels.

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

- **Equation-based and Equation-free Modelling and Simulation**: Another type of modelling is equation-based modelling (EBM). The equations represent the relations between entities or variables in EBM and, observed over time, show emergent behaviour [VDPSR98]. EBM is, therefore, a standard tool in variable-based emergence verification (cf. Section 2.5.2).

  Nevertheless, the creation of equation-based models is quite tricky and with the increasing complexity of a system nearly impossible to formulate the mathematical models. The typical approach for solving this issue is simplification. However, even small changes to the system entities can strongly affect the outcomes, e.g., if the system shows emergence or not [DWHS06]. Chen [CNC07] points out that EBM is beneficial in terms of provability but comes at the cost of high effort and the danger of oversimplification.

  De Wolf et al. [DWSHR05] propose another type of modelling, namely equation-free macroscopic analysis techniques, to study system-level behaviours. In the eyes of the authors, such techniques reduce the effort required for EBM. In addition, it is impossible to formulate the macro equation from the model in some complex systems.

In summary, the modelling and simulation of natural and artificial systems for identifying self-organisation and emergence phenomena have created a more comprehensive range of research output. Nevertheless, creating models of dynamic and complex systems suitable for simulation remains tedious. A lack of information on how the system's entities interact can result in inaccurate models and incorrect simulation results. Therefore, finding an optimal between details and effort when modelling complex systems is crucial.

2.5.4 Influencing Emergent Behaviour

A question yet left open is if there are any methods to influence emergent behaviour. The answer to this question is disputed; some authors support the claim that it is possible, others oppose it, and another group sees it as problematic.

Li, Sim, and Low [LSHL06] argue for the latter. The main reason for their standpoint is that users or designers usually do not know why emergent behaviours occur; therefore, influencing it is impossible. However, if the designers know about the cause or at least can recognise emergent behaviour, the possibility for control increases. Nevertheless, the authors acknowledge the importance of controlling emergent behaviour, as unexpected behaviour can have catastrophic outcomes.

Similarly, De Wolf et al. [DWHS06] and Müller [MS04] highlight the contradiction between the typical top-down designing process when developing artificial systems and uncontrolled emergence. Typically, development projects follow a top-down approach, such as the waterfall model. Such a hierarchical and sequential model supports understanding each system component and their states. Each step starts with a high-level description and is continually refined to models or executable code. However, designing a self-organising emergent system would require an opposite method—the macro systemic behaviour requires local autonomous interactions and activities of the entities. In this case, the designers must design low-level interactions to create macro behaviour, which is more challenging. When designing micro-level behaviour that creates and maintains the desired macro behaviour, the involved uncertainty requires certain guarantees [DWHS06]. Those guarantees are essential for industrial applications.

Unfortunately, there is a lack of research that presents solutions for controlling emergence. Parunak and VanderBok et al. [VDPV97] provide one example. The authors studied emergent behaviour management in distributed control systems. They found that a population of asynchronous running processes, without a top-down or centralised control, can create emergent behaviours at the system level. In detail, their study on welding robots in an automotive body shop identified

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

the interactions of control elements as the cause and not random events or improper functioning elements. Therefore, they proposed several methods to control undesirable emergent behaviours: nonlinear systems and agent-based control theory.

In summary, research concerning the control of emergent behaviour in self-organising systems received some academic attention but has not resulted in deep insights. The topic is closely related to the emergence prediction as the higher the prediction success, the higher the chances to control emergence.

2.5.5 Predicting Emergent Behaviour

The prediction of emergent behaviour is another issue without a consensus among researchers. Some researchers argue that predicting emergent phenomena is possible, particularly in nature.

As an example of a predictable emergent phenomenon in nature, Bajec and Heppner [BH09] see the organised flight of birds. Birds usually fly either in a line or in some cluster formation. Line formations describe birds flying in single lines joined together like pearls. Such formations are more typical with larger birds. Cluster formations, on the other hand, are more usual for smaller birds like pigeons or blackbirds. Each bird flies irregularly, but they form a cluster as a group. Academic research focuses primarily on how those clusters form or why certain birds fly in specific formations (e.g., the V-shaped formation of geese). According to Bajec and Heppner, it is simple for an external observer to predict emergent formations based on past observations. However, the birds cannot understand and predict their group's collective behaviour.

Now some researchers find the prediction of emergence as not reasonable, as the prediction contradicts the needed novelty of emergence. In Ashby [Ash56], the author presents an example. If several black boxes are connected based on a specific pattern through certain connections, it is possible to predict all behaviours. Such a group of black boxes will not display any emergent features because the components can predict all features. Therefore, if the knowledge about the components is complete, the system is complete and cannot show any new emergent behaviour. However, if the prediction is based on incomplete knowledge, there is a chance for new emergent features [Ash56]. Li, Sim, and Low [LSHL06] also do not consider predictable behaviour as emergent behaviour, as it would allow the design of management tools to control emergence.

Another group of researchers thinks the prediction of emergence is possible but only with high effort and complexity. Madey [MFT05] argues that describing local interaction rules of components is simple, but predicting macro-level behaviours is challenging, at least. Chen et al. [CNC07] support the same viewpoint as emergent behaviour is not simply and accurately predictable from the behaviour of the single elements. Another reason is given by Berezhnoy et al. [Ber03]. The prediction gets complicated if the agents carry out several activities and coordinate flexibly with others. However, without the interactions, no emergence can occur. The behaviour of each agent and its interactions with others and the environment create complex feedback loops, which makes predicting high levels of emergent behaviour challenging.

The last group of authors state that prediction is impossible. As complex systems have such a complex, dynamic, and uncertain nature, predicting or even proving emergent behaviour is impossible [JTSP13]. In the following, we list a few methods that aim to predict emergent behaviour similar to the ones that simulate emergence.

For example, Dogaru et al. [Dog08] present a method for forecasting emergent behaviour. The authors use cellular automata to predict the global behaviour of the entire system. The method assumes a state probability for the cell and a relationship between the emergent features and predicts the evolution of neighbouring cells' probabilities. In Zeigler and Muzy [ZM16], Markov models predict emergence through behavioural observations. The authors extract a state transition

TU WIEN Bibliothek Your knowledge hub
Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.

matrix and a continuous time Markov model. By analysing the Markov model for differences in the dynamic and variable structure, the authors can create probability statistics for emergent behaviour.

Other authors, such as Lancaster and Gustafson [LG13], propose to predict emergent behaviour by observing population density in different parts of the environment. The authors use probability graphs to estimate population density, the possibility of an agent's movements, and their tendencies to move in different directions. The authors examine Autonomous Nano Technology Swarms in Rouff et al. [RVH+04a] with the help of formal methods such as Communicating Sequential Process, Unity Logic, WSCCS, and X-Machine. All methods are compared and evaluated to describe the activities and predict emergent behaviours. The authors state that predicting emergent behaviour is challenging due to constant changes, and the chosen method needs to be flexible enough to continuously re-predict the emergent behaviour. Another type of prediction is found in pattern recognition [HW08]. While a traditional top-down approach, the methods show quite a high accuracy in predicting emergent features.

In summary, prediction emergence depends strongly on the complexity of the system observed. The prediction is more plausible for natural systems with repeated patterns and behaviours over time and artificial systems with specific agent interactions. The situation changes when the system's components (natural and artificial) show structures and behaviour without any time relation.

2.6 Summary

The previous sections and paragraphs provided a short but broad overview of the context of emergence. Emergence, self-organisation and complex systems are closely related, and researchers have offered several facts, facets, tools, and methods to define, measure, predict, and simulate those phenomena. Despite the variety of research, grasping the true nature of emergence remains difficult. One apparent issue is that most authors do not present examples of emergent systems. The high complexity of the systems in question might cause that.

On the other hand, instances found in nature are well-explored and documented. However, most cannot be reproduced or only with the help of simulations and simplified models. Within this dissertation, we narrow the focus to industrial systems and question whether such systems can create emergent behaviour. Therefore, the next chapter presents how the research was conducted. Chapter 4 presents further findings, a formal approach, and a potential tool for identifying emergence in industrial systems.

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Chapter 3

Methodology

The methodology chapter provides an excursion into finding truth in computer science. It provides the reason why the scientific method design science was chosen in this dissertation. Moreover, it introduces the chosen methods and gives insight into how the literature review was conducted, which was essential for the context description and the narrowing down the research focus.

3.1 A Short Excursion in Testing vs. Proving Programs Correct

The historical process of forming computer science as an independent scientific discipline included several severe debates among scientists and practitioners. One particular intense discussion, held fiercely between the participants, was the "*verification debate*". Despite its early start during the software crisis in the 1950s, this argument between the followers of software engineering and those preferring logical verification lasted more than three decades [CFR12]. One side supported the application of formal approaches as the only way to enhance software quality, whereas the opposing party, firmly argued against it.

By taking a closer look at the whole debate retrospectively, some of the arguments in this discussion resemble arguments already brought up during an earlier debate in the history of science. Namely, the dispute between logical positivists and falsificationists in their quest for truth, shows remarkable similarities, and raises the question: *"Was it the same discussion?"*.

3.1.1 A Quest for Final Truths

One of the most profound endeavours within the philosophical discourse is how to obtain knowledge in science (empirically or rationally), and how to prove the underlying facts to be true [Cru06]. During history, several schools of thought have emerged, aiming at answering this question within a widely accepted manner. Two very prominent proponents were the schools of logical positivism and falsificationism.

Positivism is a paradigm that first appeared in linguistics, semiology and epistemology, at the end of the nineteenth century. The paradigm relies on the stance that there is a one-to-one relationship between a word and a thing/idea, i.e., a positive term. The positive term excludes opinion, critic, and any trace of subjective thought [CN34]. It is, therefore, by definition, an entity that is absolutely true, independently of the various aspects of human perception. As an example, Chalmers [Cha99] mentions the case of colours. For instance, the word red, referring to the absolute perfect red, denotes something that is above everyone's approximation of the word. Positivists believe that by

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

following this paradigm, it is possible to acquire an objective model for representing information, which is applicable to all sciences. Such a model should be used as a scientific method, in order to explain phenomena in a causal manner, rather than according to intentions and goals [VW71].

Contrarily to positivism which supports that there should be strict criteria for characterising scientific phenomena, falsificationism supports that scientific claims can be made based on the lack of evidence to refute a hypothesis. Falsificationism was ignited by the observation that in science, there may not be a way to prove a universal truth [Pop05]. For instance, considering the claim that one's novel algorithm produces correct results in all cases. The number of potential cases needed to prove the algorithm correct, may be prohibitively large to allow verifying the claim based on observations. For this reason, as an alternative, falsification proposes trying to find the one observation that is enough to refute the claim, i.e., the one case for which the result of the algorithm is wrong. Thus, in falsificationism, observations are used for proving that refuting the claim can be extremely difficult.

Criticism of Positivism

Positivism's strong perspective towards finding universal truths has been the spawn for various types of criticism by the philosophy of science community. Since the views of positivists, propose using a unified model of truths in all sciences, it soon became evident that empirical claims cannot be universally true [SP05]. Thus, the universal truths were deemed as meaningless, and the original ideas of positivism became the subject of modification. Karl Popper, among the critics of positivism, noted that no number of successful empirical observations could prove a scientific hypothesis. This insight led to a new scientific epistemology called the falsificationism. However, Popper's firm negative attitude against positivism was also criticised, quite extensively, by the community. Bartley [Bar76] accused Popper of ruining the philosophical discussion and pointed out that if Popper's views on this subject, are on the right path, then the majority of the philosophers have been wasting their careers. Others have described Popper's ideas as not convincing [Cha15].

3.1.2 Formal Verification, the Only Solution

At an early stage of software development, specifically during the 1950s, the created software was prone to be faulty, which resulted in severe accidents and high monetary cost in the industry. For this reason, there was an urgent need for new methods for improving the code quality.

At the same time, the field of computing underwent significant scientific breakthroughs such as automata and complexity theory, formal languages, and semantics [Cho56, Mah11]. These developments empowered scientists to propose a rigorous mathematical approach to program construction for solving the problems in software development. Pioneers of computing such as Dijkstra, Floyd, Mc-Carthy, Naur, and Wirth believed that the only way to improve software quality is to apply strict formal approaches, sometimes referred to as formal verificationism or mathematical reductionism [Dij01]. C.A.R. Hoare promoted an even more extreme position. In his opinion, computing should be reduced into a mathematical discipline, since it would allow proving the functioning of a computer system correct [Hoa69].

Formal verification begins with the formal description of a program specification in some symbolic logic. The actual proof follows then a proof system and confirms that the program meets the foregoing formal specification. A sound proof implies that the program meets for all inputs the specification and therefore confirms that a computer system/program works correctly.

TU Bibliothek Your knowledge hub WIEN
Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.

On the contrary, the followers of software engineering believed that software testing is the most suitable process for revealing program defects. Software testing, as described by Vyatkin [Vya13, p.1243] is *"the process of revealing software defects and evaluate software quality by executing the software"*. Or as Myers. et al. [MSB11] expresses it, checking if the computer code conforms to what it is designed to do and, conversely, what is it not intended to do. Besides significant advancements in testing methods [GM16], the field suffered limitations. One is proofing a program correctly, might require exhaustive testing what is not always possible nor desirable. Further, testing bears the potential that the testing process itself introduces faults.

Notably, this debate focused on code artefacts and how to ensure their correct functioning. The spectrum of the discussion however, also involved other softer issues, such as practicability for example. Nevertheless, everything boiled down to which approach, software testing or formal verification is better suited to improve code quality. Although formal verification proved to be a cumbersome endeavour, and was refused to a certain extent by software professionals in the industry, it took more than two decades for the opponents to develop credible counter-arguments.

Criticism of Formal Verification

The sincere dedication of the verification movement making programming a rigorous mathematics-like activity to overcome the lack of program quality made the debate for the opponents quite challenging [Mac01]. However, precisely the obsession on the rigorous application of mathematics turned out to be a significant point of critique exploited by the opponents.

During the 1970s, the first critiques pointed towards the complexity of proving programs correct, as well as the relatively large effort required even for small algorithms. There is a high vulnerability to errors within the verification process itself due to its complexity [Mac01, Lon70]. Following arguments targeted the difference between the actual construction of proofs in mathematics and program verification. Mathematical proofs are undergoing an extensive social process, reviewed by peers of the same field, adjusted and republished. A very different process as the sometimes several pages long formal proofs no one dared to read again [DMLP79].

Another issue with formal verification was that there are gaps between programs, specifications, and their execution on computers in the physical world. One critique focused on the question *What a proof proves "correct"?*. In practice, a proof verifies that the program text corresponds to the formal specification and in the best case shows that the constructed program is "correct". Nevertheless, the opponents argued that program verification has no connection to the real world and does not say anything if one constructed the right program. Therefore, if an application is valid and useful is outside the scope of formal proofs and makes the whole idea of proofing programs correct difficultly.

The second primary argument was that the physical world is uncertain compared to theoretical constructions. It might be possible to prove in some cases in an abstract level that the program code conforms to the specification: Though, specifications and programs are models and abstractions of reality and therefore never 100% accurate [Fet99]. This particular argument was further extended by Fetzer [Fet88] by pointing out the fundamental difference between algorithms and executed program code on a computer. An algorithm is nothing more than a mathematical formula, whereby a program running on physical machines moves very real electrons in circuits around.

At the end of 1980, the formal verification movement relinquished their firm standpoint and started to adjust to the software engineering practices. C.A.R. Hoare wrote that it was a mistake to see formal verification and testing as opposites, as they both contribute to the quality of programs [Hoa09].

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

3.1.3 The Two Debates Compared

At first glance, the two debates show several similarities. Both positivism and formal verification hold an idealistic view of the world where proving scientific facts is possible by applying a unified model. Similarly, testing and falsificationism follow a more realistic approach towards truth in science and acknowledge the possibility that there are no universal truths. Nevertheless, comparing the finer elements of each debate reveals distinct differences that refute the hypothesis that the discussions are the same.

Non-Exclusiveness

Besides that, the followers of formal verification saw their approach as superior; no one ruled out the usefulness of testing altogether. In computing, the situation exists that both methods would work for a task, but the community considers one better suited. Sometimes testing and formal verification are used complementary.

The field of digital design, for example, is concerned with converting human logic in electrical logic based on flip-flops, arithmetical logical units, and other similar electronic circuits. In digital design, hardware description languages (HDL's) are used for describing the hardware itself, and not logic directly as in C-like programming. For instance, a traditional addition (Var C=A+B), is in HDL represented as registers, connected wires and other elements. The output of HDL is a static electronic circuit that executes the intended logic.

Based on the digital design, the output is transformed into hardware, and the final product must represent the exact intended logic. Testing is in that case, not sufficient to ensure the functionality, and therefore, the community prefers a formal approach. The verification involves additional programming, and strict evaluation of all input and output values, nonetheless, the additional effort is considered necessary.

A counterexample is distributed systems that span over larger geographical areas. Such systems are used mainly for providing unified services to users of various locations (with similar quality of service). Instead of hosting one service in one location (e.g., in a cloud data centre), multiple instances of this service are hosted in various locations around the world. Presumably, these instances communicate with each other (and with the users) through the Internet. Since the geographical distance (over the Internet) among the instances is an integrated part of the system, verifying that the system functions correctly is difficult. Modelling the Internet is not credible since it is a vast network with many unknown influencing factors. Therefore, even though it is possible to follow a verification approach, testing the system under real-world conditions is considered more trustworthy.

Both examples show that the community decided what method is better suited for the task even besides both being applicable. Such a co-existence was no alternative in the debate between positivism and falsificationism. The two schools of thought saw their views as absolute, especially in how to define facts to be true and valuable for science. With such a narrowed viewpoint, it was simply not possible that both could exist in parallel.

Pseudo Science

Hidden in the more profound nature of the debate between positivism and falsificationism there is another difference to the formal verification discussion. Both paradigms show a harsh judgement if knowledge or facts are scientifically useful or not. In the case of falsificationism, theories can only be of scientific value when falsifiable. For illustration, a statement such as *"Strong typing reduces run-time errors"* [Sne98] are not clearly falsifiable. Although plausible, in a falsification manner,

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

such remarks would be classified pseudo-science without value for computer science research. The same applies to positivism that considers only by the senses observable facts as worthy.

The proponents of formal verification partly followed such a rigorous mindset, yet their main focus was on improving code quality, not the usefulness of the code. For example, a program that produces faulty results in certain circumstances might be still useful for all the other inputs. Testing and formal verification were always about improvement, and not about absolute terms.

3.1.4 Conclusion

The formal verification debate and the discourse between positivism and falsificationism show significant similarities considering the used arguments. Driven by the ambition to provide universally applicable methods in their respective field, all approaches suffered from weaknesses. Positivism, as well as formal verification, saw universal truths and logic as the only way to achieve scientific relevance and in turn, lost the connection to the real world by ignoring the relations between facts/-code and reality. Similarly, falsificationism and testing suffered from the fact that sufficient testing sometimes requires an unreasonable (or infinite) amount of observations. Additionally, the actual testing process bears the possibility to introduce new flaws that, in the end, falsify the theory or identify bugs.

Notwithstanding all the similarities, the discussions are not entirely the same. This claim rests on the earlier described inconsistencies within debates that in summary are: (*i*) The impossible co-existence of positivism and falsificationism compared to the common practise in testing and formal verification to choose the best fit. (*ii*) Furthermore, that positivism and falsificationism treat not "correct" facts as non-valuable knowledge for science, whereby testing and formal verification aims for improvement. Moreover, one might argue that the narrowed focus on code in the formal verification debate, is not on the same level as the strive for an all-encompassing method applicable in science. The pointed out differences are valid arguments to support the claim that the two debates are indeed not the same.

Another insight is that there were many cases where positivism and falsificationism worked well, but due to their strictness were not flexible enough to cover other situations. This inability to adapt might have been one of the reasons that both paradigms disappeared in time besides having a large number of followers. Testing and formal verification, however, moved on towards combined approaches that further improved software quality and ensured each other's continuation.

3.2 Choosing Desing and Creation as Research Strategy

While the above excursion is not directly applicable to the topic of this dissertation, it still points towards an important question: "How to create knowledge in computer science?". Based on the above example, there is no universal truth, and only a variety of tools and methods create valuable knowledge. Considering this finding, this dissertation follows a design and creation research strategy.

The design and creation research concept builds upon academic literature and relevant specifications [Oat05]. This strategy applied to computing research focuses on developing new products called artefacts [MS95]. Such artefacts encompass constructs such as models, methods and instantiations [Che00a]. However, most research in the field is a combination of several artefacts that contribute to actual knowledge gain. As the artefacts are mostly related to computing, the design and creation research strategy emphasises analysis, explanation, argument, justification, and critical evaluation of the results to distinguish itself from traditional product development.

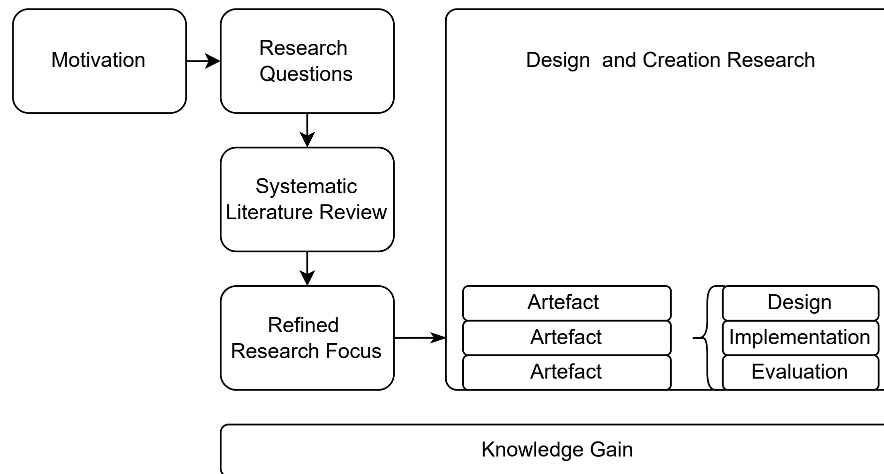


Figure 3.1: Outline of the research design and the relations between the single parts.

For this reason, the design and creation research approach focuses either on the artefact itself (e.g., the computing application incorporates a new theory), the artefact as a vehicle to create new knowledge (e.g. the IT application in use), or on the process to create an artefact to create knowledge [HMPR04]. Depending on the focus point, different methods and tools are more suitable.

Figure 3.1 outlines the research design and how the parts are connected. The starting point is the motivation and the resulting research questions. A systematic literature review (cf. Section 3.3) created a solid foundation. The outcome provided the input for Chapter 2 but also refined the research focus and allowed the specification of the artefacts in Chapter 4.

The first artefact is the interacting agents in an industrial setting that create emergent behaviour. By analysing the interactions between the agents, new insights are created. However, for doing the analysis, a suitable tool/method needs to be available. In our case, we identified a suitable formal method (algebra) but had to extend it with new capabilities. Therefore, the method is also an artefact; its extensions represent additional knowledge gain.

Design and creation research follows a problem-solving approach similar to the principles of system development [HWH99]. For the artefacts, we adopted the typical five steps of the system development process: awareness, suggestion, development, evaluation and conclusion. Nonetheless, the steps are not executed in full depth, i.e., no finished product/system is developed, instead a formal description that reaches the level of a prototype, which is tested.

3.3 Systematic Literature Review

The performed systematic literature follows the suggestions described in Pearl Brereton et al. [BKB+07]. The first step was defining guiding research questions that would provide fundamental and more specific answers. The following task was choosing data sources and the search strings, followed by acceptance/rejection criteria that support selecting relevant publications and how to extract data.

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

3.3.1 The Guiding Research Questions

The intention of the guiding research questions is, on the one hand, to provide basic information about the research area. In this dissertation, the research is about the emergence or emergent behaviour, which includes self-organisation and complex systems. On the other hand, the questions shall provide answers specifically supporting the main research questions of this dissertation. The following list contains all guiding questions with some additional comments.

- G_RQ1: *"What is/are emergence, self-organisation and complex systems?"*
  A fundamental question to provide connections between the research fields. Moreover, it provides the possibility to give generally accepted definitions.
- G_RQ2: *"What are the differences between emergence and self-organisation?"*
  This question points towards to common misinterpretation that emergence and self-organisation are the same. In addition, the differences are essential for detecting emergence.
- G_RQ3: *"What are the characteristics of emergence? "*
  A question that targets the different characteristics of emergence described in the literature.
- G_RQ4: *"What methods are used to identify emergence?"*
  This question aims to collect possible methods to identify emergence.
- G_RQ5: *"How to confirm emergence?"*
  Similar to the previous question, a focus is given if an emergent behaviour is truly emergent.
- G_RQ6: *"How to control emergence?"*
  A question to find tools to influence the occurring emergent behaviour.
- G_RQ7: *"How to model and simulate emergence?"*
  Mainly collect available tools and methods.
- G_RQ8: *"How to predict emergence?"*
  Or, in other words, is it possible to foresee emergent behaviour within a system based on simple interactions or structural constellations?
- G_RQ9: *"What science disciplines emergence has been studied/found?"*
  Another general question that supports the next question.
- G_RQ10: *"Has emergence been found in industrial automation or multi-agent systems?"*
  A question specifically designed to narrow down the research focus.

Following the definition of the guiding research questions was creating search strings for the databases.

3.3.2 Search Strings and Databases

Each of the guiding research questions provided the keywords for the search strings. For example, for G_RQ1 the string contained the words (emergence $\vee$ emergent $\vee$ spontaneous order) $\wedge$ (behaviour $\vee$ pattern) $\wedge$ (complex systems) $\wedge$ (self-organisation $\vee$ self-assembly $\vee$ self-organising $\vee$ self-generating $\vee$ self-created $\vee$ self-ordering $\vee$ self-arranging). In Table 3.2, we present all used search strings in the search. Note: We used both versions in cases where English terms like "behaviour" have American and British spelling. In addition and we used a synonym finder for each keyword.

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

We used the databases IEEE Explore, SpringerLink and Science Direct for our search. At first, we used Google Scholar as well, but the search yielded too many results and filter options are limited on this platform. On the other platforms, we applied the following filters (if available):

- Publication date: 1950-2018
- Peer Reviewed
- Language English
- Search Fields: Title, Keywords, Abstract.

After the initial search, we examined the found documents according to the following criteria.

3.3.3 Acceptance/Rejection Criteria

The screening process was mainly conducted manually and had three stages. In the first stage, the abstracts were read, and the following criteria decided if an article stayed in the set:

- Yes if: The presented research clearly relates to emergence (emergent behaviour), self-organisation or complex systems (e.g., multi-agent systems).
- Yes if: The presented research should consider proper sciences (e.g., physics, biology, engineering, chemical, natural or artificial).
- Yes if: The article type is either a conference publication, journal paper or a PhD dissertation.
- No if: There is no evidence of the published venue or less than three pages.
- No if: The paper is obviously out of scope (e.g., philosophical nature or pseudo-science).

In addition, some more practical exclusion criteria were: duplicates and articles not available as full text (i.e., some databases provide an abstract, but the article is behind a paywall or is not online accessible). This first stage resulted in 537 papers. The next stage focused on the main text and included a first allocation to the relevant G_RQ's. Highly cited papers were given priority.

In the last stage, we used the papers with the highest citations counts and survey papers as a source for further relevant papers. In systematic literature research, this step is called forward and backward search. This step yielded some additional papers we had not found in the main search. The result of this final stage is presented in Table 3.1 split up by type of publication.

3.3.4 Data Extraction

The final set of publications was read, and each paper was sorted according to its contribution to the guiding research question. An article can either only contribute to one G_RQ1 or several. The examination dept varied as in some papers, only parts were relevant.

3.3.5 Results

Each guiding research question provided input for some of the subsections in the previous chapter. Some outcomes were used in the following chapter to refine the research focus and formulate a formal proposal on identifying emergent behaviour in industrial agent systems.

TU Bibliothek WIEN Your knowledge hub
Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.

3.3.6 A Comment on the Conducted Literature Review

When we conducted the literature review end of 2018 till the middle of 2020, a review paper by Kalantari et al. [KNM20] was published that covered the same topic. As the overlap was significant, we decided not to publish our findings as initially planned but to keep them inside this dissertation. Nevertheless, we compared the findings, and the differences were the following. The authors in Kalantari et al. [KNM20] included a smaller set of articles but provided an additional classification on emergence [KNM20, p.13]. For better comparability, we introduced the same classification. While their article set was smaller than ours, we identified four papers we had not found in our search. In addition, the authors aimed for a broader overview, while ours was on the connection between emergence, complex systems and self-organisation. Moreover, a specific interest of our study, if emergence has been found in industrial agent systems, is also not covered by Kalantari et al. [KNM20].

Table 3.1: Overview of the number of final documents of the final stage split by type.

Type of document	Number of used documents	Year of publication
Journal	134	1965-2020
Conference	62	1995-2018
Dissertation	6	1990-2016
Book	53	1920-2019
Technical reports and others	6	1875-2004
Total number:	**261**	

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Table 3.2: Search strings used in the literature review and their relation to the guiding research questions.

Number	Search String
G_RQ1	(emergence ∨ emergent ∨ spontaneous order) ∧ (behaviour ∨ pattern) ∧ (complex systems) ∧ (self-organisation ∨ self-assembly ∨ self-organising ∨ self-generating ∨ self-created ∨ self-ordering ∨ self-arranging)
G_RQ2	(emergence ∨ emergent ∨ spontaneous order) ∧ (behaviour ∨ pattern) ∧ (complex systems) ∧ (self-organisation) ∧ (differences)
G_RQ3	(emergence ∨ emergent ∨ spontaneous order) ∧ (behaviour ∨ pattern) ∧ (complex systems) ∧ (self-organisation) ∧ (characteristics ∨ properties ∨ features)
G_RQ4	(emergence ∨ emergent ∨ spontaneous order) ∧ (behaviour ∨ pattern) ∧ (complex systems) ∧ (self-organisation) ∧ (identify ∨ detect)
G_RQ5	(emergence ∨ emergent ∨ spontaneous order) ∧ (behaviour ∨ pattern) ∧ (complex systems) ∧ (self-organisation) ∧ (validation ∨ verification)
G_RQ6	(emergence ∨ emergent ∨ spontaneous order) ∧ (behaviour ∨ pattern) ∧ (complex systems) ∧ (self-organisation) ∧ (control)
G_RQ7	(emergence ∨ emergent ∨ spontaneous order) ∧ (behaviour ∨ pattern) ∧ (complex systems) ∧ (self-organisation) ∧ (simulation ∨ tools)
G_RQ8	(emergence ∨ emergent ∨ spontaneous order) ∧ (behaviour ∨ pattern) ∧ (complex systems) ∧ (self-organisation) ∧ (prediction ∨ forecast)
G_RQ9	(emergence ∨ emergent ∨ spontaneous order) ∧ (behaviour ∨ pattern) ∧ (complex systems) ∧ (self-organisation) ∧ (Computer Science ∨ Social Science ∨ Nature ∨ Biology ∨ Economics ∨ Physics ∨ Chemistry)
G_RQ10	(emergence ∨ emergent ∨ spontaneous order) ∧ (behaviour ∨ pattern) ∧ (complex systems) ∧ (self-organisation) ∧ (IIoT ∨ IoT ∨ Multi-Agent Systems ∨ Internet of Things ∨ Cyber Physical Systems ∨ CPS)

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

Chapter 4

Emergent Behaviour in Industrial Systems

This chapter brakes down and aligns the emergence and self-organisation phenomena to the industrial context. The focus, especially on multi-agent systems, allows for setting boundaries to the phenomena and places some critique on past research. Based on a definition of a multi-agent system (MAS) and the introduced boundaries, the chapter presents a formal language-based identification method for emergence. Moreover, using a formal language approach, linking Kronecker Algebra as a possible tool to detect emergent pattern formation is possible.

4.1 Seamless Communication in Industry

In the industrial context, emergent pattern formation might appear in the shadow of Industry 4.0 (I4.0) and Industrial Internet of Things (IIoT). Factories are complex technical environments built upon software and hardware agents from information technology (IT) and operational technology (OT) domains. Since the 1970s, IT and OT in industrial automation have formed a hierarchical automation pyramid [Wil94] with several layers, as shown in the left part of Figure 4.1. The lower levels of the automation pyramid, close to the factory floor, represent OT devices such as programmable logic controllers (PLCs) with industrial communication systems, e.g., EtherCAT or Profibus. OT representing complex control loops must fulfil strict real-time requirements to guarantee a timely processing of sensor values and a safe operation of actuators, valves, and electrical motors [SKJ18]. On a higher level, the control loops are being monitored by employing Supervisory Control and Data Acquisition (SCADA) systems and other industrial applications [WSJ17]. Applications in the third OT layer do not have to meet strict real-time requirements; instead, they focus on high data throughput paired with computational power and internal connectivity. The two top layers contain Manufacturing Execution System (MES), plant management, business, and Enterprise Resource Planning (ERP). They utilise commercial off-the-shelf (COTS) IT, such as servers and desktop PCs that interconnect via standard IT communication systems (Ethernet) [HGB15].

In this hierarchical architecture, IT/OT are separated. As the communication systems in OT are optimised for deterministic low latency, tight synchronisation, and low jitter [SKJ18], they are difficult to bridge into a standard IT network commonly deployed as Ethernet infrastructures. Similarly, the IT communication networks and systems cannot cope with the deterministic OT requirements [HGB15]. Those differences result in device isolation on the factory floor from the computational resources and the connectivity available in the IT levels of a factory. They, therefore, hinder seamless vertical communication between all devices.

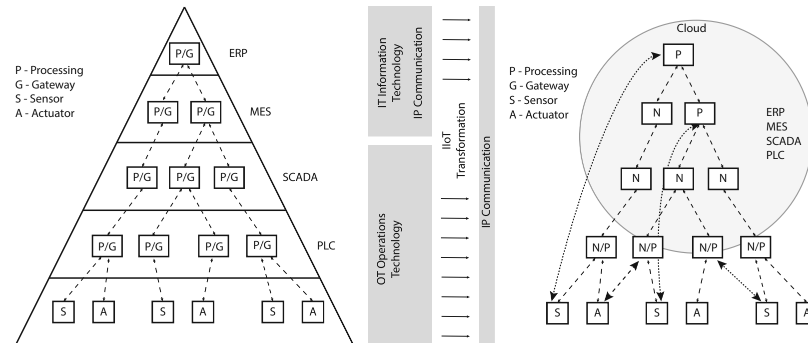


Figure 4.1: Automation pyramid transformation towards a flat IIoT architecture. Adjusted from [SKJ18].

Closing this IT/OT gap is one of the aims of I4.0 and IIoT. The information can flow without any hindrance by seamlessly connecting all factory elements and flattening the architecture to an only IP-based environment, as visualised in the right part of Figure 4.1. The idea is not new and is based on the IoT, a term coined by Kevin Ashton in 1999. In IoT, the seamless connection spans the entire Internet and foresees intelligent devices interacting with each other [IA12]. While ubiquitous computing has existed since the 1980s, the past few years have seen accelerated progress in various domains, e.g., home and building automation, smart grids, and e-health applications [IA12].

While these changes might benefit data collection and process optimisation, they also introduce more complexity to the systems. Before, the IT/OT gap created a barrier between agents; now, they can interact directly. Most components will behave like agents interacting with each other and their environment. Recalling the main characteristics of emergence in Section 2.3.4, such as micro-macro effect, novelty, flexibility, decentralised control, interacting parts or coherence, the reader might agree that in IIoT, emergent behaviour can appear.

4.2 Emergence in Industry 4.0, IIoT and Multi-Agent Systems

Based on the assumption that the emergent pattern formation exists in I4.0 and IIoT, the first question was whether research covers that aspect. The guiding research questions 9 and 10 were specifically formulated to answer this question.

A the time being, publications focusing on I4.0, IoT and IIoT and emergence were somewhat limited. Some researchers focused on hierarchical emergent behaviours influencing an IoT system to show a desired "emergent behaviour". The idea is to apply lightweight local rules that define the interactions between "things" and the environment and create a higher level of behaviour [RNN+16, RMNV18]. A similar direction is proposed by Mihailescu et al. [MSHD18], where the users' intention is included in configuring a role-based approach to influence the IoT system. While the keywords I4.0, IoT and IIoT did not produce relevant hits, research in MAS is more common.

MASs are closely related to I4.0, IoT and IIoT. However, including MAS brings up the problem of differentiating between complex adaptive and multi-agent systems. As stated in Section 2.1, in a complex system, complexity is created by the agent interactions and the creation of information in the process [Ger12]. The components create the information by transforming old information into new, therefore considered emergent. Those transformations are dynamic, static, active or stigmergic [FMG14].

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

In addition, the interactions which are not centrally controlled lead to a higher macro behaviour of the entire system. The macro behaviour is absent on the agent levels and has an increased structure or order. Another requirement is that the macro behaviour is created in a self-organising manner [MMS11]. This self-organised creation of emergence in complex systems is emphasised by several authors [Gol99, DMSGK11]. Dessalles and Phan [DP06] mention that the macro behaviour is visible to an external observer (weak emergence) or in strong emergence by the agents themselves.

Research in MAS defines emergence slightly differently. Similarly, according to Li, Sim, and Low [LSHL06], the interacting agents create the behaviour collaboratively. The Santa Fe Approach to Complexity (SFAC) is often mentioned within the context of multi-agent systems and emergence. SFAC defines emergence as patterns, structures, and features as part of the entire system but not as part of its components (agents). The agents do not have the knowledge or the capacity to predict the macro behaviour of the system. Nevertheless, knowing about the nature of the interactions between the agents increases the knowledge about emergent behaviour [DP06].

Multi-agent and complex adaptive systems overlap in the field of robotics. Robotic agents can self-organise and interact among themselves (and humans), creating emergent behaviour [TNN+16]. Sturdivant and Chong [SC18] propose identifying robot emergence based on symbols.

Lastly, there is a relation to swarm intelligence. This multi-agent type is more common in biological systems such as fish or ant colonies [Har18, LBT12]. In such systems, the intelligence created by the entities is considered an emergent property [DSA93]. Collective intelligence is created by several simple agents [BDTT99, LBT12].

In summary, whether academic research in I4.0, IoT and IIoT covers emergent pattern behaviour needs to be answered with no. However, the work done in the context of MAS is more extensive and is closely related.

4.3 A Little bit Critique and Some Clarifications

Before it is possible to go on, there is a need for some criticism of the previously described work. The following list summarises a few points that are reoccurring issues:

- Some found articles treat emergence relatively as informal and intuitive. This issue goes so far that emergence is explained without any theoretical foundation [BDG95a, BDG95b, Bro95] or framework [Baa94, Bed03, RSC99].
- Similarly, some articles use too broad theories and therefore include un emergent phenomena.
- In some cases, the definitions are too narrow [Baa94]. It is impossible to differentiate different types of emergence or identify the source of the emergent behaviour [Bed03, Car89, RSC99].
- Some authors introduce an aspect of “surprise” to emergence [RSC99, RS00].
- Little is written about modelling techniques that would allow the design and study of emergent behaviour in multi-agent or complex systems. Note: There is work based on simulations but with limited results (cf. Section 2.5.3).

It is essential to define boundaries for a formal definition to avoid similar issues. Therefore in the following, the listed limitations shall apply.

- Firstly, despite a wide spectrum of related work and various definitions presented in the previous sections, the focus is on a formal method. The aim is to build upon work from authors such as Kubí, Aleš [Kub03], which propose formal definitions to find emergence.

- Secondly, the formal method should allow the construction of examples formally and using a tool. Most authors remain purely formal without evaluation.
- Thirdly, there is no surprise involved. Surprise is subjective and covers the fundamental explanations for new system behaviour. An observer might be surprised by behaviour at first and later gain knowledge about the system that removes the surprise. In addition, the system's creator might not be surprised as the behaviour was expected.

In summary, in the following formal proposal shall provide a comprehensible approach for identifying emergence in industrial systems.

4.4 What Could Cause Emergent Behaviour in a Multi-Agent System?

Towards a formalised definition of emergence suitable for industrial and multi-agent systems, let us have a closer look at the possible root causes. As we have learned, the primary indicator of emergence is the macro-behaviour of the entire system. The interacting agents cause this macro-behaviour with themselves and the environment.

Therefore we can assume that the agent's properties, the environment's influence on the agents, the interactions between the agents or evolutionary changes could be the source of emergent behaviour. The possible combinations of causes could lead to different levels of emergence, as other authors have already indicated. Therefore, we propose the following cases:

1. In the first case, the agents are in a static environment; there is no feedback or input from the environment towards the agents. Moreover, the properties of the agents remain the same.
2. The second case involves the environment, as it provides input to the agents and influences their actions. However, both the agents and the environment keep their properties.
3. The third case is similar to the first one; only the agent properties can change over time, i.e., the behavioural rules can change or evolve.
4. Case four includes the environment, but only the agents can change their behaviour.
5. The last case allows the agents and the environment to evolve. That is the most complex case.

The observant reader might have realised that those cases mirror different types of MASs. From very simple to very complex systems, in some sense, a similar concept as the in Section 2.3.5 presented classification of emergence. Nevertheless, what is still missing is an understanding of what an agent is.

An agent can be a program, a process, an organism or any entity that can perceive its environment with some sensors (senders) and produces changes or events with its actuators (receivers). Moreover, such agents can act autonomously to fulfil their internal goals or programming. That includes elementary agents that react (memory-free) or react based on a memory, up to agents that apply reasoning and planning. The latter agent type would represent a social agent that can negotiate with other agents, e.g., making a plan or solving a conflict. Some robots can already show such agent behaviour.

A particular type of agent is the environment. The environment is a joint base or structure all other agents act upon or interact. It can be an active participant or a passive one. In its active

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

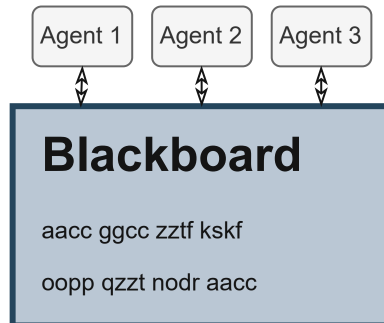


Figure 4.2: A simple model to visualise three agents and a common blackboard (environment).

form, it influences all agents, for example, by imposing rules (e.g., laws in social agents). Based on this understanding of a MAS, it is possible to represent cellular automata, neural networks, software applications, organisms, ecosystems, or other social systems.

4.4.1 Informal Definition of Basic Emergence and the Connection to Formal Languages

After getting an idea of categorising emergent behaviour in multi-agent systems, we can formulate a first informal definition for emergence. This informal definition will provide the baseline for our formal definition.

First, we limit our approach to an elementary form of emergence without self-organisation. A type often called "Basic Emergence" where the agent interactions in a shared environment clearly cause the phenomena. However, the emergent phenomenon is not just a summation of the behaviours of individual agents in the environment. The agents and the environment are not evolving during their interactions (i.e., an agent's behavioural set stays the same during the time). The environment is purely passive and does not influence the agents; however, it is changed by them. Therefore, in our definition, only emergent phenomena that MAS of the first case can represent are possible. Compared to the definition of DeWolf and Holvoet [DWH05] in Section 2.3.2, we exclude dynamical aspects, but remain closely in line.

Focusing on basic emergence first makes it possible to formulate an approach based on formal languages and grammar systems. In recent decades, research in distributed computing and decentralised systems has brought up several grammar systems that allow a formal description and modelling of multi-agent systems [CVDKP18]. The main advantage of a formal language approach is that it provides a joint modelling base with a defined alphabet and rewriting rules for all the system parts. The micro-structures (the agents' alphabet and the rewriting rules) create directly observable macro-patterns (the language of the system). There is no need for different languages to describe the agents and the system, as the rewriting rules specify the agents' actions that from the system's behaviour [Hol98].

Before diving into formal language definitions, let us establish a simple model. Figure 4.2 shows three agents that communicate with each other via the environment (blackboard/tape). The agents each have a language created by grammar and symbols. Those symbols are written on the tape and form words. In the following, we introduce the basics of formal languages and how grammar systems can help to create a formal language proposal to identify emergence in MAS.

TU Bibliothek Your knowledge hub WIEN
Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.

4.5 A Formal Language Proposal

Research about formal language theory examines formal language properties and their relation to model computational devices, such as FSMs or Turing machines. The Chomsky hierarchy of formal languages [Cho56] connects the various languages with different computational devices.

A central element of formal languages are grammars which define how to write symbols on a tape guided by rewriting rules. As grammar creates a language, a significant research interest lies in creating simple grammars to generate complex languages. In most cases, formal grammars define nonterminal and terminal symbols. The nonterminal symbols can be rewritten, while the terminal symbols are unchangeable. Strings that consist of terminal symbols produced by the grammar are called words. Therefore, a language defined by grammar is a set of words and strings.

If more than one grammar is operating on one tape (or several tapes), then the system has evolved into a grammar system. The different grammars interact via the tapes they write upon, i.e., each grammar is an individual computational device with a set of rules and symbols. It is possible to model MAS with specific properties, with grammar systems. The single grammar represents an agent that operates on the tape (environment) and creates events based on their behaviour (rewriting rules) [CVDKP18]. Different grammars are available, each varying in communication type, amount of tapes or component grammars. For further information, we guide the reader to [Sal73, ST99].

4.5.1 Formal Languages

As mentioned before, most formal languages and grammar systems are based on Chomsky Grammars [Cho56]. Within this dissertation, let us define $\mathcal{A}$ as a finite alphabet containing symbols or letters. $\mathcal{A}^*$ shall represent all strings ℓ that can be created of the symbols in $\mathcal{A}$ and the length of a string $\ell \in \mathcal{A}^*$ (the number of symbols in string ℓ) we define as $|\ell|$. An empty string is denoted by $\varkappa$. In a string, the number of times a symbol can occur $H \subseteq \mathcal{A}$ in $\ell \in \mathcal{A}^*$, shall be $|\ell|_H$. To concatenate strings, we define $u \diamond v$, where $u, v, \in \mathcal{A}^*$, yet we often omit the symbol and write uv instead. A Chomsky Grammar [Cho56] is defined as a quadruple:

$$G = (N, T, P, S), \tag{4.1}$$

where N stands for a nonterminal and T for a terminal alphabet. P is a finite set of rewriting rules (productions) which define operations in the form of $y \rightarrow x$, with $y, x \in \mathcal{A}^*$. $S \in N$ is a starting symbol. A direct derivation in G is $\Longrightarrow$ or $\Longrightarrow_G$. Transitive and reflexive closures of the relation $\Longrightarrow$ are $\Longrightarrow^+$ and $\Longrightarrow^*$. A language $\mathcal{L}$ in $\mathcal{A}$ is a subset of $\mathcal{A}^*$. Union, intersection, and complementation are defined over languages, as usual. The language generated by G is a construct:

$$\mathcal{L}(G) = \{w \mid S \Longrightarrow_G^* w, w \in T^*\} \tag{4.2}$$

The formal language elements are called *strings* or *words*. Returning to Figure 4.2, each agent has its specific language.

4.5.2 Cooperating Grammar Systems

Grammar systems are a combination of grammars (agents) that rewrite their symbols on a joint tape [CVDKP18]. Each grammar rewrites a portion of the tape, as an agent would do in its environment. Different cooperation strategies define how grammars can rewrite the symbols on the tape and whether they communicate directly. Moreover, it is defined when the process of derivation

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

ends. In some cases, the environment (tape) can have its own rewriting rules and therefore interfere with the agents [CVKKP97]. The grammar system creates a new language, different from the languages generated by individual grammars.

Formally a cooperating grammar system can be defined as:

$$\Gamma = (N, T, S, P_1, P_2, \ldots, P_n) \tag{4.3}$$

N is again a nonterminal and T a terminal alphabet, $S \in N$ is the starting symbol. The $P_1, P_2, \ldots, P_n$ stand for the finite sets of rewriting rules over $N \cup T$. In a cooperating grammar system, a basic derivation is defined as $\Longrightarrow_{p_i}$:

$$x \Longrightarrow_{p_i} y \quad \text{if holds that,} \quad x = x_1 u x_2, \quad y = x_1 v x_2, \quad x_1, x_2 \in (N \cup T)^*, \quad u \to v \in P_i. \tag{4.4}$$

It is also possible to define derivations of arbitrary length $\Longrightarrow_{p_i}^{*}$ and for a specific amount of steps h for $h \geq 1$ we write $\Longrightarrow_{p_i}^{h}$. For the maximal derivation, we write $\Longrightarrow_{p_i}^{t}$. However, more interesting is how a derivation of strings on the tape is coordinated when more than one agent operates on it. Let us define: $F = \{*, t\} \cup \leq h, = h, \geq h \mid h \geq 1$. Therefore a cooperating grammar system Γ with n elements and $f \in F$, the language Γ creates in the mode f is:

$$\begin{aligned} \mathcal{L}_f(\Gamma) = \{x \in T^* | S \Longrightarrow_{p_{i_1}}^{f} x_1 \Longrightarrow_{p_{i_2}}^{f} x_2 \ldots \Longrightarrow_{p_{im}}^{f} x_m = x, \\ m \geq 1, \quad 1 \leq i_j \leq n, \quad 1 \leq j \leq m\}. \end{aligned} \tag{4.5}$$

Another question is how the derivation in a grammar system proceeds. The answer is either parallel or sequential. If the agents write on the tape in parallel, synchronisation is required as the tape is a shared resource. On a tape, two or more agents cannot rewrite the same symbol simultaneously. The synchronisation prevents the agents from writing on the same symbol. In case the agents write in sequence, no synchronisation is required as they get their turns to write without disturbance. In Figure 4.2, one line on the blackboard represents the output of the cooperating agents.

The next issue open to be solved is that so far we only considered a one-dimensional tape of symbols. For our proposal, we need to be able to capture all symbols on the blackboard.

4.5.3 Cooperating Array Grammar Systems

One possibility for grammars to operate on a canvas or blackboard is introducing array grammars that extend the one-dimensional tapes with a second dimension. This approach is successfully used in pattern recognition or analysis of agent behaviour. The book of by P.S.P. Wang [Wan89], provides an overview of possible applications of array grammars and cooperating array grammar systems.

Array grammars are just regular two-dimensional arrays that allow the identification of each symbol by its i, j index. Nevertheless, in array grammars, the extent of the board is limitless, i.e., there is no limit to the number of symbols that can be written. # symbols fill empty spaces. As the one-dimensional grammars, array grammars are defined by their rewriting rules, just in this case, the input and outputs can be arrays. We note we are using only a small portion of the array grammars capabilities, and therefore we guide the reader for more details to [DFP95, FFH99, Fre00]]. Especially, Dassow et al. [DFP95] provides interesting insights into the topic.[1]

[1] The following definitions are based on Dassow et al. [DFP95]. For better readability, we refrain from adding to each line a reference.

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

However, let us first define an array of symbols as a set of point coordinates i, j and a corresponding symbol placed on the tape. Moreover, let $\mathbb{Z}$ be a set of integers and V again an alphabet. An array A over V is $A : \mathbb{Z}^2 \rightarrow V \cup \{\#\}$, where,

$$\text{support}(A) = \{v \in \mathbb{Z} | A(z) \neq \#\} \tag{4.6}$$

represents a set of coordinates without the blank symbol ($\# \notin V$). We can write

$$A = \{(v, A(v) | v \in \text{support}(A)\}. \tag{4.7}$$

Next, we denote all two-dimensional nonempty arrays over V by V^{+2}. Each subset of V^{+2} represents an array language, while the arrays are the languages' words. Therefore we can define that an array $A \in V^{+2}$ and a finite pattern α of symbols over $V \cup \#$, α is a subpattern of A if the following holds:

- The superimposition of α on A needs to be possible so that all vectors of α with symbols from V coincide with the corresponding symbols in A.
- In addition each $\#$ in α corresponds to a $\#$ in A.

According to Dassow et al. [DFP95] an array grammar is a construct

$$G_{Ar} = (N, T, \#, P, \{(v_0, s)\}), \tag{4.8}$$

with the alphabets N, T containing nonterminal and terminal symbols. The symbol $\#$ is, as above, a blank symbol. P is a finite set of rewriting rules $\alpha \rightarrow \beta$. The array pattern α, β span over $N \cup T \cup \{\#\}$. $\{(v_0, s)\}$ defines a starting array, with an starting vector $v_0 \in Z^2$ and starting symbol $S \in N$.

Applying the definitions above, we can now say that for an array grammar $G_{Ar} = (N, T, \#, P, \{(v_0, s)\})$ and two words $A, B \in (N \cup T)^{+^2}$ the relation $A \Longrightarrow B$ holds if:

- a rule exists $\alpha \rightarrow \beta \in P$ such that α is a subpattern of A,
- and that B is created by replacing α in A by β.

In addition, $\Longrightarrow^*$ denotes the reflexive and transitive closure of $\Longrightarrow$. Following again Dassow et al. [DFP95], an array grammar G_{Ar} generates a language with the following structure:

$$L(G_{Ar}) = \{A \in T^{+^2} | \{(v_0, S)\} \Longrightarrow^* A\}. \tag{4.9}$$

Therefore, based on the previous definitions, a cooperating array grammar system can be formally defined as an $(n+4)$-tuple:

$$GS_{Ar} = (N, T, \#, \{(v_0, S)\}, P1, P2, ..., Pn), \tag{4.10}$$

In summary, a cooperating array grammar system describes parallel rewriting operations on a tape by various array grammars. Nevertheless, more than one production rule cannot rewrite a symbol. Using the above definitions makes it possible to define subsets of array grammars. For example, in a subset $Sub1 = \{Ai, 1 \leq i \leq n\}$, and two arrays $D1, D2 \in V^{*^2} \cup \{\#\}$, the direct derivation can be expressed as $D1 \Longrightarrow_{S1} D2$. This derivation requires array productions in the form of $p_{i_j} \in P_i, 1 \leq i \leq n, 1 \leq j \leq l_i$ (jth rule of ith array grammar) $p_{i_j} = \alpha_{i_j} \rightarrow \beta_{i_j}$. Those productions need to apply for:

- Any array symbol ω_1 with index $v_k, 0 \leq k \leq r \times s - 1, r, s \in Z$

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

- that is a subpattern of $\beta_i j_1$, and not a subpattern of $\alpha_{i_{j_1}}$.
- Any array symbol ω_2 with index $v_m, 0 \leq m \leq r \times s - 1, r, s \in Z$
- that is a subpattern of $\beta_{i_{j_2}}$ and is not a subpattern of $\alpha_{i_{j_2}}$.
- Plus all symbols are disjoint ($k \neq m$).

With the above definitions, the agents in Figure 4.2 can rewrite a portion of an array tape in one step. In each step, the array grammars (agents) apply their rewriting rules on the tape.

4.5.4 Modifications on Cooperating Grammar System

Now there is the possibility to capture the agents behaviours on the blackboard. However, there is one limitation in Chomsky grammars. This type of grammar assumes that the derivations end at some point, e.g., terminal symbols. As in agent systems, the interactions might go on endlessly, we propose to soften this requirement, and use so called "pure" grammars. Another simplification, is that all agent symbols shall be part of the alphabet V_A and all environmental symbols in V_E.

4.5.5 Summing up the Agents Behaviour

Returning to the informal definition of emergence in Section 4.4.1, there is a need to define the sum of the agents' behaviours in a MAS. One possibility is to sum up the languages of the agents if they do not communicate. By doing so, we get a set of words that the agents can generate if here is no interaction. In other words, the agents write on the tape as they are the only ones doing so. By summing up these words, we get a candidate for all individual agent behaviours. Let us define L_{sum} for n languages as $L_{sum} = \{L_1 + L_2 + \ldots + L_n\}$. Each language contains words of symbols $\in V_A$.

4.5.6 A Formal Definition of a Basic Emergence

After the short overview of formal languages and their application in cooperating grammar systems, it is possible to define a proposal for a formal definition of emergence in MAS. First, let us represent a MAS as a cooperating grammar system in the following way:

$$\mathcal{MAS} = (V_A, V_E, A_1, A_2, A_3, \ldots, A_n, S). \tag{4.11}$$

Where V_A, as introduced before, is the alphabet of agent symbols and V_E stands for the environment. The alphabet describing the entire MAS is $V = V_A \cup V_E$. S is the initial starting point of the environment, in most cases, an empty set. Each of the n agents is defined by $A_1, A_2, A_3, \ldots, A_n$; they are a finite set. The agents itself we define as:

$$A_i = (V_i, P_i, S_i) \tag{4.12}$$

Similarly, as before, $V_i \subseteq V$ is the alphabet of agent A_i. S_i is the initial starting point, and P_i is a set of rewriting rules. Utilising the definitions of the cooperating grammar system, an agent can derive from $u \Longrightarrow v$ in case $u, v \in V^+$ there is a rule $\alpha \rightarrow \beta \in P_i$ such that α is a subpattern of A, and B is obtained by replacing α in A by β. We denote the language of an A_i by

$$\mathcal{L}(A_i) = \{w \in V^+ | S_i \Longrightarrow^* w\}. \tag{4.13}$$

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

The derivations are done either parallel or sequentially so that there is no interference between the agents when writing on the environment. Note that it still applies that more than one agent cannot rewrite a part of the environment/symbol at the time. As not all agents will write at the same time and produce a w, we define a subset of agents:

$$Sub1_{\mathcal{MAS}} = \{A_i \,|\, 1 \leq i \leq n\}. \tag{4.14}$$

Now the $\mathcal{L}(\mathcal{MAS}) = \{w \in V^+ | S \Longrightarrow Sub1_{\mathcal{MAS}} w_1 \Longrightarrow Sub2_{\mathcal{MAS}} w_2 \ldots \Longrightarrow Subn_{\mathcal{MAS}} w_n\}$ is the language generated by the MAS. If $\mathcal{L}(\mathcal{MAS})$, compared with the agent languages $L_{sum} = \{L_1 + L_2 + \ldots + L_n \,|\, 1 \leq i \leq n\}$ creates a situation where a w $\exists$ and is $\in \mathcal{L}(\mathcal{MAS})$ but is $\notin L_{sum}$, then the defined property of basic emergence is fulfilled. Simplified, the definition above states that in case a MAS as a whole creates a language (behaviour) that cannot be found in the sum of the agents' languages (agent behaviours), the ground that the MAS shows basic emergence is fulfilled.

4.5.7 Some Comments

The above-presented definition is a simple approach to identifying basic emergence in MAS, focusing on agents' interactions. While this focus is minimal, considering the variety of possible causes of emergence, it represents a reasonable starting point. The approach can be extended in future work, as there are possibilities to include different forms of communication [Kub01], adjust the internal structure of the agents (stateful, reasoning), or incorporate the influence of the environment [CVDKP18, CVKKP97]. Those extensions would cover all four MAS cases presented in Section 4.4.

The next section, briefly elaborates on, how the tool was chosen to apply the above definition. We note already, that the process was not linear rather evolved over time after several tryouts.

4.6 Finding a Suitable Tool

Let us have another look at Figure 4.2 and try to adjust it to visualise the previously presented formal definition. Each agent writes its symbols on the tape and changes parts of the tape based on the cooperating grammar rules. If we add several stacked "tapes", we already get a good idea of how it would look when the agents write the symbols over time. Figure 4.3 shows the three agents writing their symbols on the tapes. If we see the different tapes as a sequence, the different written symbols could also be state changes.

As introduced before, formal languages model computational devices. In our case, we use a grammar (Type 3, regular grammar in the Chomsky Hierarchy [Cho56]) that can be executed by finite state machines (FSMs). FSMs are finite automatons represented by states and directed edges.[2] The strings created by the state machine represent the language. Therefore, we can represent the agent's behaviours completely.

If we assume that FSMs can represent our agents, we need a tool or method that can operate with FSMs. Additionally, to fulfil the idea of the entire system representation, it must be possible to determine all possible states of an MAS in operation. In other words, the tool must provide all recorded steps on the blackboard the agents issued. Let us summarise all basic requirements:

[2] A formal introduction will be given in Chapter 5.

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

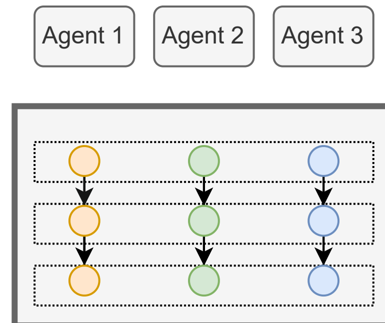


Figure 4.3: The adjusted simple model of three agents cooperating on a common blackboard (environment). The tapes are stacked to visualise the activities over time.

- The tool needs to individually represent the agents as FSMs.
- The FSMs need to be able to interact (synchronisation)
- All interactions and issued symbols must be available (the entire state space).

One possibility would be to apply a simulation environment, where the state machines execute the steps and the communication symbols are recorded for later analysis. However, in simulation, there is always the possibility that specific interactions never get recorded. Moreover, simulation tools only represent the state machines but do not manipulate them. Another option is mathematical tools such as Matlab or specific programming languages like LinguaFranka and Rebeca or model checker like UPPAAL+, capable of representing MAS. Those tools would make a good choice, especially LinguaFranka or Rebeca; however, their focus lies timing analysis to discover issues in safety critical applications such as airplanes or cars.[3]

In our search, we identified a particular type of algebra, namely Kronecker Algebra, that fulfils all the requirements and additionally inlines very well with our formal approach. Nevertheless, we had to extend Kronecker Algebra with capabilities such as:

- Combining state machines to a large system representation.
- Be able to create state machines from message sequence charts (MSCs).
- Create the possibility to prioritise agents over others.
- Allow timing analysis of execution paths.

We elaborate on the reasons for the extensions in the following sections. Nevertheless, the findings of the literature review support some extensions. So far, no research has been conducted on the influence of priorities and timing of agent interactions. As mentioned in the methodology chapter, we also created knowledge while extending Kronecker Algebra. Therefore, each chapter is a single contribution. Each contribution is applicable to other problems, not only to our formal approach.

[3] We plan to use LinguaFranka and Rebeca in future work.

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

4.7 Summary and Limitations

Within this chapter, we elaborated on our definition of basic emergence. The definition is based on the literature review results and aims to provide a formal definition. The use of different cases of MAS as a framework and formal languages as guidance, the definition proposes a cooperating array grammar system to identify basic emergence. In essence, the symbols issued by the cooperating agents will give rise to a higher level of systems behaviour. If there is a difference between the total system behaviour and the sum of the languages of the agents, emergence might be present. In this way, it is possible to observe the micro-macro behaviour of the agents formally.

While only basic emergence is the target, the definition can be extended to more complex types of emergence. Limitations are that no evolutionary processes of the agents nor the environment are considered, as well as hierarchical interactions. Moreover, we consider the environment passive and not influencing the agents' behaviours. The presented definition follows ideas from Kubí [Kub03], and Dassow et al. [DFP95]. Nevertheless, the differences are that Kubí [Kub03] focuses on the superimposition of the cooperating agents' languages while we use the summation of the agents' languages. Moreover, we can provide a total system representation based on our chosen tool. The work of Dassow et al. [DFP95] is different in the perspective that the authors used their cooperating array grammars to identify patterns in images.

Based on the presented formal definition, we defined requirements a possible tool must fulfil for the evaluation. Kronecker Algebra was chosen as an appropriate solution as it allows the continuation of a formal treatment of basic emergence. Moreover, it represents a significant difference to other research that mainly remained in the formal context or resorted to simulation to identify basic emergence. The next chapter introduces the basics of Kronecker Algebra.

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

Chapter 5

Kronecker Algebra — A Matrix Calculus

This chapter will mainly be concerned with introducing the basic concepts of Kronecker Algebra and its applicability to verify software properties. In particular, we introduce methods to check if a single-threaded software implementation complies with usage scenarios specified for parts of the software. More accurately, if the software's control-flow graphs (CFGs) comply with a usage scenario in the form of an FSM. Thus we further introduce FSMs, their matrix representations and how to manipulate them with Kronecker product operation—followed by Kronecker Sum and synchronisation primitives such as semaphores. The chapter also explains how to implement Kronecker operations efficiently and presents an overview of related work.

5.1 A little History of Kronecker Algebra

The history of Kronecker Algebra, has many facets and fascinated several researchers over centuries that came up with ever new ideas how to apply it. Therefore, the following background is only a mere scratch on the surface.

Johann Georg Zehfuss introduced the primary operand defined by the symbol $\otimes$ in 1858 [Zeh58]. Throughout history, this operand received various names, such as the Zehfuss product, the Hurwitz product, the Producttransformation, the conjunction, the tensor product, the direct product, and Kronecker product. Nowadays, the name Kronecker product is mainly used for the symbol and operation, $\otimes$. Research conducted by Brigitte Plateau [Pla85] in the context of stochastic automata, provided the operation with wide attention from the community. In the following, Gerhard Küster [Küs91] proved that Kronecker Sum generates all interleavings of concurrently executing automata. Over the last decade, Kronecker Algebra was extended with other operations. For a general overview, the interested reader might consider consulting the work of Graham [Gra18]. Applying Kronecker Algebra (i.e., Kronecker product and Kronecker Sum) to automata within the Kuich-Salomaa notation was presented by Mittermayr and Blieberger [MB11].

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

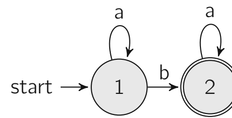


Figure 5.1: A simple example of an FSM (M)

5.2 Finite State Machines and their Matrix Representation

The possibilities and tools for modelling software systems are close to endless. However, so-called finite state machines are a basic recurring pattern differing only in naming in most modelling languages. Similar concepts are, for example, finite automata or flow graphs. In the course of this dissertation, FSMs play an essential role.

Each FSM consists of a finite number of states, and there is generally only one initial state. The FSM (M) depicted in Figure 5.1 has state 1 as its initial state, indicated by an arrow named "start". Each state can also be a final state, and there is no limitation on the number of final states. FSM (M), has only one final state, state 2, marked with a double circle. The states can be named arbitrarily; however, within this dissertation, the states will be numbered from 1 to n if an FSM consists of n states.[1]

The connections between the states in an FSM are "directed edges". Each edge has a direction indicated by an arrow and indicates a state transition. Those transitions happen while a system evolves, i.e., it changes its state via an edge.[2] Usually, each edge has a label, and our FSM (M) in Figure 5.1 has three:

- edge ($1 \rightarrow 1$) labelled with "a",
- edge ($1 \rightarrow 2$) labelled with "b", and
- edge ($2 \rightarrow 2$) again labelled with "a".

The edge labels can serve different purposes; however, for the moment, we define them as a set of labels $\mathcal{L}$. Later on, labels will play an essential role. While most readers might be familiar with how a state machine evolves, the following explanation helps to understand the next section. The system represented by the FSM (M) will start at state 1. From there, two options exist. Either via the state transition $1 \rightarrow 1$, i.e., (M) stays in the same state but still issues label "a". Alternatively, it transits to state 2 via the edge ($1 \rightarrow 2$) and produces the output "b". In state 2, (M) has the possibility again via the edge ($2 \rightarrow 2$) to produce output "a", or the system terminates as state 2 is a final state.

While the above example implies an activity, an FSM does not "issue" an edge label; it is a passive construct that reacts to the input string created by a program and does not create an output. However, later chapters often suggest that an FSM issues a label/edge. The first reason is that we sometimes use CFGs of programs as FSMs (cf. Section 5.4.1). As a program executes, the activity is indicated by issuing a label. Another reason is the previous chapter, where we used FSMs to represent MAS agents with "active" behaviour.

[1] In some cases, the node numbering will be different due to the applied operations.

[2] This is a deterministic process. Other state machines use different approaches, such as probabilities, when changing a state. [HMU06]

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

A formal definition of a deterministic FSM [HMU06], is a quintuple: $(Q, \iota, \mathcal{L}_A, T, F)$, with a specific name M and a given alphabet $\sum_A$, where:

- Q is a finite non-empty set of states,
- ι is an initial state ($\iota \in Q$),
- F is the set of final states, a subset of Q,
- $\mathcal{L}_A \subseteq \sum_A$ is the input alphabet, and
- T is the state transition function: $T : Q \times A \rightarrow Q$.

The reader agrees that representing an FSM in a graphical form has advantages. The visualisation of states and edges provides a better understanding of what is happening in which case. However, that benefit only applies to human readers. For computers, it is unsuitable as it does not allow further operations or manipulations.

5.2.1 Matrix Representation

An alternative representation for FSMs utilises matrices and vectors. But let us first introduce basic matrix terminology that applies in the remainder of this dissertation.
A p-by-q matrix

$$M = (m_{i,j}) = \begin{pmatrix} m_{1,1} & m_{1,2} & \cdots & m_{1,q} \\ m_{2,1} & m_{2,2} & \ddots & \vdots \\ \vdots & \ddots & \ddots & \vdots \\ m_{p,1} & m_{p,2} & \cdots & m_{p,q} \end{pmatrix} \tag{5.1}$$

has p rows and q columns. It has therefore, p times q entries $(m_{i,j})$. The i and j in $m_{i,j}$ denote the row and column numbers within the matrix M. That allows us to define the set of matrices $\mathcal{M} = \{M = (m_{i,j}) \mid m_{i,j} \in \mathcal{L}\}$ and state that only matrices $M \in \mathcal{M}$ will be used in the remaining parts of this work. Often we use a k-by-k matrix known as a square matrix of order k. Furthermore, we introduce zero matrices $Z_n = (z_{i,j})$, where $\forall i, j : z_{i,j} = 0$. For the zero matrices, we require neutral elements, for which applies $m_{i,j} \cdot 1 = 1 \cdot m_{i,j} = m_{i,j}$ and $m_{i,j} + 0 = 0 + m_{i,j} = m_{i,j}$ and we note that $m_{i,j} \cdot 0 = 0 \cdot m_{i,j} = 0$. The operation, written in infix notation "$\cdot$", simply denotes the juxtaposition of its operands. For example let a and b denote two entries (edge labels), then $a \cdot b$, denotes their juxtaposition. In addition, we define it as a repetitive juxtaposition of identical operands in the power notation, e.g., $a \cdot a \cdot a = a^3$. The operation with, written in infix notation "+", denotes an operation of choice. For example let a and b denote two entries (edge labels), then $a + b$, denotes that we are free to choose either a or b. In a program context, that is the same as an if-statement. We further assume that $a + b = b + a$, as the order of the then- and else-branches of if statements, is not crucial for our case. The reason is we do not know the conditions of the if statements and the exact order of the then- and else branches and their evaluation order as, for example, in elif arms. Further, we assume that "+" is an idempotent operation, i.e., $a + a = a$. And finally, we note that the operations obey distributive laws, i.e., $a \cdot (b + c) = a \cdot b + a \cdot c = ab + ac$ and $(a + b) \cdot c = a \cdot c + b \cdot c = ac + bc$.

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

As identity matrix I_n of order n, we introduce a matrix with ones at the main diagonal and zeros elsewhere, i.e.,

$$I_n = (m_{i,j}), \text{ where } m_{i,j} = \begin{cases} 1 & \text{if } i = j, \\ 0 & \text{otherwise.} \end{cases}$$

Another matrix that we often refer to is a so-called sparse matrix. Let $M = (m_{i,j}) \in \mathcal{M}$ and the number of entries unequal to zero defined by $||M|| = |\{m_{i,j} | m_{i,j} \neq 0\}|$. Thus a sparse matrix is an n-by-n matrix that fulfils $||M|| \leq c * n$, where c is a constant independent from n.

After the above definitions, it is possible to create a correspondence between matrices and FSMs. Let us define that any arbitrary FSM is also a directed labelled graph $G\langle V, \mathcal{L}, n_e \rangle$, with a set of labelled nodes V, a set of labelled directed edges $\mathcal{L}$ and an entry node n_e. Therefore, the correspondence between FSMs and matrices (referred to as adjacency matrices) is as follows. Each graph node represents a positive integer, reflecting the row and column in the adjacency matrix. If an entry $m_{i,j} = a$ in an adjacency matrix exists, a directed edge exists from node i to node j with label a in the directed graph. If there is no entry, $(m_{i,j} = 0)$, there is no edge from node i to node j.

Let us return to our example FSM (M). M has two states; therefore, a square matrix M is created with the order of k = 2, i.e., a matrix with $p = k$ rows and $q = k$ columns. Each of the edges of FSM (M) is now filled into M as described above, which leads to:

$$M = \begin{pmatrix} a & b \\ 0 & a \end{pmatrix}$$

Note: places without an edge are filled with a "0". However, as the matrices tend to grow fast, we replace the 0 entries with "·", to enhance readability. Therefore, the matrix M reads now:

$$M = \begin{pmatrix} a & b \\ \cdot & a \end{pmatrix} \tag{5.2}$$

The initial and final states are the last FSM (M) elements that are still missing. We can represent those states with the previously introduced neutral elements (e.g., $a \cdot 1 = 1 \cdot a = a$, $a + 0 = 0 + a = a$ and $a \cdot 0 = 0 \cdot a = 0$). With the neutral elements, we model:

1. The initial states by a line vector solely consisting of 0s and precisely one 1. In particular, if i is the initial state, then the ith entry equals 1.

2. The final states by a column vector consisting of 0s and 1s. More precisely, if f is a final state, then the fth entry equals 1. Thus, the final state vector has precisely as many entries equal to 1 as there are final states in the FSM.

In our example FSM in Figure 5.1, the initial state vector is $S = (1, 0)$ and the final state vector is $F = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$.

While it is now possible to represent an FSM as a matrix, what benefit is gained? Especially as the graphical representation (cf. Figure 5.1) is more intuitive and the matrix representation more complex at first sight. Therefore, the following section explains how matrices are valuable tools for modelling state transitions.

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

5.2.2 State Transitions (Finding Successors)

To recapitulate, the matrix M of an FSM is a compact representation of all possible state transitions. In case there is an interest in knowing in which state a system might evolve, based on current state i, the natural way might be to turn towards the graphical representation to find all possible successors of node i, i.e., all connected nodes that are reachable via a directed edge starting from node i. However, that can be done as well by looking at row i of matrix M, because all non-zero entries are successors of node i. Alternatively, if the (i,j) (row i, column j) entry in matrix M is non-zero, node j is a possible successor of i. If the system evolves along that edge, it will issue the label at the matrix's entry $m_{i,j}$.
In our example FSM in Figure 5.1, let us assume the system is in state 1. The state machine can therefore evolve either:

- via edge $(1 \rightarrow 1)$ and issue the label a,
- or via edge $(1 \rightarrow 2)$ and issue the label b.

The reader will agree that the same information is immediately visible when looking at the matrix M in Eq. (5.2) at row $i = 1$. At first sight, that might not be very impressive; however, if the transition count is higher, it becomes more challenging to gain information from the graphical representation of the FSM. The matrix representation is more suitable for automation purposes.

For example, it is possible to compute M^k,i.e., the output of the FSM after k transitions. Let us use matrix M again and calculate,

$$M^2 = M \cdot M = \begin{pmatrix} a & b \\ 0 & a \end{pmatrix} \cdot \begin{pmatrix} a & b \\ 0 & a \end{pmatrix}.^3$$

We assume the reader is familiar with matrix multiplication; however, we apply the operations introduced in Section 5.2.1 for multiplication "·" and summation "+". For example, to calculate entry $(1, 1)$ of the above matrix product, we take row 1 from the left matrix (a, b) and the column 1 of the right matrix $\begin{pmatrix} a \\ 0 \end{pmatrix}$. Thus we get

$$a \cdot a + b \cdot 0 = a^2 + 0 = a^2,$$

for our first entry. We calculate the remaining entries in the same way.

$$M^2 = M \cdot M = \begin{pmatrix} a^2 & ab + ba \\ 0 & a^2 \end{pmatrix}$$

Now, the result can be interpreted again via the rows of the matrix. If the FSM was in state one, we could see that it either stayed in state 1 (by performing two transitions via the edges $1 \rightarrow 1 \rightarrow 1$, with the outputs $a \cdot a = aa = a^2$) or it evolved to state 2. The latter can transition in two ways:

$1 \rightarrow 1 \rightarrow 2$ with the outputs $a \cdot b = ab$ or
$1 \rightarrow 2 \rightarrow 2$ with the outputs $b \cdot a = ba$.

Which transitions will be performed at the end depends on the input. With some further mathematical effort, we can define "all possible outputs of a state transition system". By introducing a Kleene star and an algorithm introduced by Ésik and Kuich, we can calculate M^* [ÉK12, p.36]. M^* represents the $\sum_{k \geqslant 0} M^k$ (the sum of all M^k), and applied to our example would result in

[3] We use 0 instead of "·" for the mathematical representation.

$$M^* = \begin{pmatrix} a^* & a^*ba^* \\ 0 & a^* \end{pmatrix}.$$

By taking the initial S and final F state vectors into account, we obtain

$$S \cdot M^* \cdot F = (1\ 0) \cdot \begin{pmatrix} a^* & a^*ba^* \\ 0 & a^* \end{pmatrix} \cdot \begin{pmatrix} 0 \\ 1 \end{pmatrix} = a^*ba^*.$$

This result can be understood that our FSM creates as an output an arbitrary number of "a" followed by one single "b", again followed by an arbitrary number of "a". One would expect this output when looking at the FSM in Figure 5.1. However, the same finding could be found straightforwardly and algebraically, which can be easily automated. In the following sections, we introduce other well-known matrix operations allowing other interesting analyses of transition systems.

As a side note, the above definitions assume that there is always an end state; however, later in this dissertation, state machines without end nodes, such as semaphores, will be used. Such state machines are defined as Büchi Automata and are not explicitly introduced (the interested reader is guided to [ÉK12] for a definition). The reason is that the following matrix operations allow handling both types of state machines. Moreover, the two types will not be differentiated in the following chapters for simplicity.

5.3 Definition of Kronecker Product

Kronecker product denoted with operand $\otimes$ is the product of two matrices $A_{m_A \times n_A} \in \mathbb{R}^{m_A \times n_A}$ and $B_{m_B \times n_B} \in \mathbb{R}^{m_B \times n_B}$, written as $A \otimes B$. This algebraic tensor operation is defined as:

$$A \otimes B = \begin{pmatrix} a_{1,1}B & a_{1,2}B & \dots & a_{1,n}B \\ a_{2,1}B & a_{2,2}B & \dots & a_{2,n}B \\ \vdots & \vdots & \ddots & \vdots \\ a_{m,1}B & a_{m,2}B & \dots & a_{m,n}B. \end{pmatrix} \tag{5.3}$$

Whereby, each $a_{i,j}B$ is a block of size $m_B \times n_B$. E.g., if

$$A = \begin{pmatrix} a_{1,1} & a_{1,2} \\ a_{2,1} & a_{2,2} \end{pmatrix}, \quad B = \begin{pmatrix} b_{1,1} & b_{1,2} \\ b_{2,1} & b_{2,2} \\ b_{3,1} & b_{3,2} \\ b_{4,1} & b_{4,2} \end{pmatrix} \tag{5.4}$$

$$\text{then } A \otimes B = \left(\begin{array}{cc|cc} a_{1,1}b_{1,1} & a_{1,1}b_{1,2} & a_{1,2}b_{1,1} & a_{1,2}b_{1,2} \\ a_{1,1}b_{2,1} & a_{1,1}b_{2,2} & a_{1,2}b_{2,1} & a_{1,2}b_{2,2} \\ a_{1,1}b_{3,1} & a_{1,1}b_{3,2} & a_{1,2}b_{3,1} & a_{1,2}b_{3,2} \\ a_{1,1}b_{4,1} & a_{1,1}b_{4,2} & a_{1,2}b_{4,1} & a_{1,2}b_{4,2} \\ \hline a_{2,1}b_{1,1} & a_{2,1}b_{1,2} & a_{2,2}b_{1,1} & a_{2,2}b_{1,2} \\ a_{2,1}b_{2,1} & a_{2,1}b_{2,2} & a_{2,2}b_{2,1} & a_{2,2}b_{2,2} \\ a_{2,1}b_{3,1} & a_{2,1}b_{3,2} & a_{2,2}b_{3,1} & a_{2,2}b_{3,2} \\ a_{2,1}b_{4,1} & a_{2,1}b_{4,2} & a_{2,2}b_{4,1} & a_{2,2}b_{4,2} \end{array}\right). \tag{5.5}$$

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

5.3.1 Further Properties

Kronecker product has some further properties. Let A, B, C, and D be matrices. Kronecker product is non-commutative because, in general, it applies

$$A \otimes B \neq B \otimes A.$$

However, it is permutation equivalent because there exist permutation matrices P and Q such that $A \otimes B = P(B \otimes A)Q$ (cf. [Wei62, Gra18]). Moreover, in the case that A and B are square matrices, then $A \otimes B$ and $B \otimes A$ are permutation similar (cf. [Gra18, Pla85]). It is associative as

$$A \otimes (B \otimes C) = (A \otimes B) \otimes C. \tag{5.6}$$

In addition, Kronecker product distributes over $+$, i.e.,

$$A \otimes (B + C) = A \otimes B + A \otimes C, \tag{5.7}$$

$$(A + B) \otimes C = A \otimes C + B \otimes C. \tag{5.8}$$

Other properties are for example, the connectedness of the corresponding undirected [Wei62] and directed graphs [McA63, HT66]. Similarly, about the correctness of resulting graphs in [HIKer]. We guide the reader to [Bel97, Hur94] for additional properties and proofs.

5.3.2 Our Semiring

As indicated the Kronecker product found various applications suitable for natural and real numbers ($\mathbb{R}$) [Gra18]. Nevertheless, in the course of this dissertation, $\mathbb{R}$ is generalised to R, which denotes a semiring $(R, +, \cdot, 0, 1)$ where $(R, +)$ is an idempotent monoid with identity element 0 (i.e., $(a + b) + c = a + (b + c)$, $a + 0 = 0 + a = a$, $a + b = b + a$, and idempotent means $a + a = a$ for $a, b, c \in R$). In addition, $(R, \cdot)$ is a monoid with identity element 1 (i.e., $(a \cdot b) \cdot c = a \cdot (b \cdot c)$, $a \cdot 1 = 1 \cdot a = a$, but in general $a \cdot b \neq b \cdot a$ for $a, b, c \in R$), multiplication left and right distributes over addition (i.e., $a \cdot (b + c) = a \cdot b + a \cdot c$ and $(a + b) \cdot c = a \cdot c + b \cdot c$), and multiplication by 0 nullifies R (i.e., $a \cdot 0 = 0 \cdot a = 0$ for $a \in R$).[4] Furthermore, the semiring is equipped with the unary star operation $*$. For each $l \in \mathcal{L}$, l^* is defined by: $l^* = \sum_{j \geq 0} l^j$, where $l^0 = 1$ and $l^{j+1} = l^j \cdot l = l \cdot l^j$ for $j \geq 0$.

The main application of Kronecker product within this dissertation is related to FSMs, as it calculates the simultaneous executions of the input matrices. For this reason, we can use the operation $\otimes$ to synchronise automata.

5.3.3 Applying Kronecker Product

As introduced in Section 5.2, state machines consist of states, edges, and labels. For using Kronecker product on FSMs, they have to be represented as matrices in the form introduced in Section 5.2.1. For demonstration purposes, let us assume we have two FSMs, A and B (cf. Figure 5.2), multiplied by Kronecker product as shown in Eq. (5.9) and considering the definitions given in the semiring.

[4] The semiring follows up on the definitions presented in Section 5.2.1.

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Figure 5.2: The two FSMs A and B

$$A \otimes B = \begin{pmatrix} a & b \\ \cdot & \cdot \end{pmatrix} \otimes \begin{pmatrix} \cdot & a \\ \cdot & b \end{pmatrix} = \begin{pmatrix} \cdot & a \cdot a & \cdot & b \cdot a \\ \cdot & a \cdot b & \cdot & b \cdot b \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \end{pmatrix} \quad (5.9)$$

Additionally we compute with the initial and final states $S_A = (1, 0)$, $S_B = (1, 0)$, $F_A = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$ and $F_A = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$:

$$S_{A \otimes B} = (1, 0, 0, 0), \quad F_{A \otimes B} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix} \quad (5.10)$$

this leads to the FSM depicted in Figure 5.3. The reader might find this at first sight not very impressive. However, after a short reflection, it becomes evident that Kronecker product of A and B issues all edge labels as A and B would perform in lockstep. In lockstep both FSMs issue their labels at the same time, i.e., $\mathcal{L}_A \cup \mathcal{L}_B$. Another intriguing effect, while the matrix in Eq. (5.9) has size four, there are only three reachable states in Figure 5.3. State number three cannot be reached from the initial state.

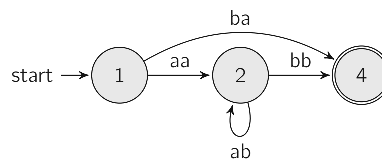


Figure 5.3: The resulting FSM of $A \otimes B$

The described capability of Kronecker product can be used to verify computer programs.

5.4 Verifying Programs with Kronecker Product

This section will show how to use Kronecker product to check modular program blocks for their conformity with specified usage scenarios. However, there is a need to introduce first a change to the operation "$\cdot$", which says that $a \cdot a = a$ and $a \cdot b = 0$. Therefore, the new result of $A \otimes B$ of the previous example is shown in Eq. (5.11).

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

$$A \otimes B = \begin{pmatrix} \cdot & a & \cdot & 0 \\ \cdot & 0 & \cdot & b \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \end{pmatrix} \tag{5.11}$$

Now, let us transform Eq. (5.11) back into an FSM. Looking at the FSM depicted in Figure 5.4, the new FSM issues the same labels as the two state machines A and B would in synchronised lockstep, i.e., the two FSM issue their intersections ($\mathcal{L}_A \cap \mathcal{L}_B$). Nevertheless, this behaviour was also present in Eq. (5.9) without focusing on synchronisation. The issued labels are the synchronisation edges between the two initial FSMs [MB11].

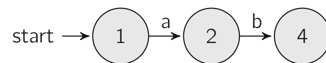


Figure 5.4: The resulting FSM of $A \otimes B$ with a focus on the synchronisation edges, while executing in lockstep.

5.4.1 Control Flow Graphs

As a next step, let us glimpse into compiler construction, where looking at programs not in the machine or source code form, but rather in the form of a control-flow graph (CFG) is pretty standard. A control-flow graph (CFG) is a directed labelled graph $G\langle V, \mathcal{L}, n_e, V_f \rangle$, with a set of labelled nodes V, a set of labelled directed edges $\mathcal{L} \subseteq V \times V$, an entry point n_e and a set of final nodes $V_f \subseteq V$ [ASU86]. A CFG has exactly one root node n_e without any incoming edge. As with FSMs, the final nodes have a double circle, and a CFG can have one or more final nodes. Those nodes stand for the termination of the program. Moreover, each $n \in V$ is reachable via a path starting from n_e.

Typically CFGs consist of basic blocks and relations to model the control flow [ASU86]. Because Kronecker Algebra operates on the edges, the basic blocks need to be moved to the missing edges.[5] The edges represent the transfer of control between the basic blocks, and each edge is assigned a basic block. Figure 5.5 depicts two CFGs, based on the same program code. Figure 5.5 (a) shows the CFG with the basic blocks on the nodes, while Figure 5.5 (b) shows the basic blocks at the edges. The operations on the basic blocks are $\cdot$ and $+$, as defined in Section 5.3.2. Those operations model the consecutive program parts, conditionals, and loops. As a final note to control-flow graphs (CFGs), there are several possibilities to create CFGs from the source code.

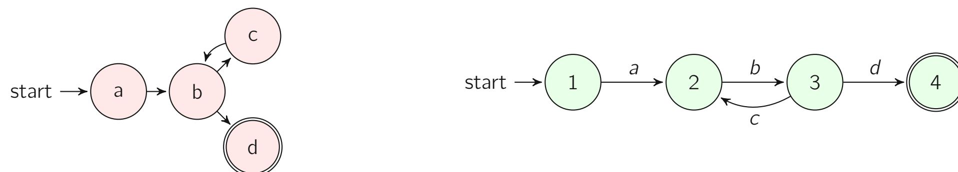


Figure 5.5: The figure depicts on the left a CFG with the basic blocks at the nodes (a), while the CFG on the right, the basic blocks are on the edges (b).

[5] We decided to use the incoming edges.

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

5.4.2 Usage Scenarios

Not only programs can be represented as a CFG, the same applies to so called usage scenarios. A usage scenario defines a task, pattern or steps a program is required to correctly execute. If this is not the case the program is faulty. Modelling languages such as Unified Modeling Language (UML), use state machines to express and specify usage scenarios of classifiers, interfaces, and ports. The UML state machines can be represented in the same fashion as the previously introduced FSMs. However, they need to be deterministic, i.e., all outgoing edges of a node have different edge labels.

Let us use an example for better understanding. While handling files on a system they can be opened and closed. Open files can be used to read data. We use the shorthands "o" for open, "c" for close, and "r" for read. Figure 5.6 shows an example for such file usage scenario (F).

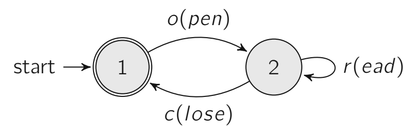


Figure 5.6: Graphical Representation of File Usage Scenario F

A piece of software or system that want to use a file, has therefore to comply with the usage scenario F. Figure 5.7, depicts such a system A, that performs two read operations, and then closes the file.

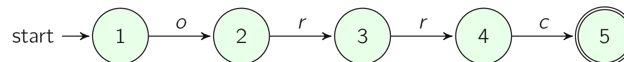


Figure 5.7: Graphical Representation of File Usage System A

5.4.3 Applying Kronecker Product

As introduced before both program A and usage scenario F are CFGs which can be represented by FSMs. Executing both in lockstep should terminate the program if it complies with the usage scenario. Therefore let us apply Kronecker product to the two CFGs, as introduced in Section 5.3.3. Figure 5.8 shows the resulting graph of $A \otimes F$.

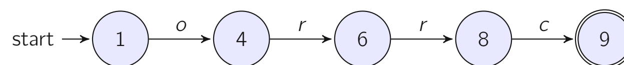


Figure 5.8: Graphical Representation of $A \otimes F$

The reader agrees that Figure 5.8 and Figure 5.7 are identical, except the node IDs differ. In graph theory, such a similarity is called an isomorphism of graphs.

5.4.4 Isomorphism

Kronecker product of the program and the usage scenario will result in a new FSM. In case the resulting FSM and the program CFG are isomorphic, the implementation is correct (at least in this

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

concern). Recalling, graph theory, an isomorphism of graphs G and H is a bijection f between the node sets of G and H such that any two nodes u and v of G are adjacent in G if and only if $f(u)$ and $f(v)$ are adjacent in H. The graphs are called *isomorphic*, written $G \simeq H$, if an isomorphism exists between two graphs. Moreover, to compare CFGs, there is a need to check if the root node r of G is the root node $f(r)$ in H. All final nodes s of G have to be final nodes $f(s)$ in H and all final nodes t in H need to be final nodes $f^{-1}(s)$ in G. If this comparison fails there is no guarantee that program terminates. For further details on the isomorphism problem and how to automate it, the reader is directed to [MB11].

In summary, Kronecker product can be used to verify programs if they comply with a usage scenario. However, so far, that only works for programs consisting entirely of operations defined in the usage scenario.

5.5 Kronecker Skip

For dealing with additional statements in a program, a Kronecker operation is required that allows for "skipping" statements not of interest to the analysis. An additional statement or an operation in a program are actions that are not part of the usage scenario.

Using the same example as above, the usage scenario F remains the same (cf. Figure 5.6), but Figure 5.9 shows a new file usage system K with additional statements. K consists of the file operations plus some additional statements a and b. If we would apply Kronecker product to the program K and the usage scenario F would result in a graph not isomorphic to K. The analysis would indicate the program is incorrect, despite the file usage is correctly implemented. One reason lies in the fact that both FSMs need to execute in lockstep, if an operation needs to be executed not existing in the usage scenario the system would stop. In the example, the system would not get over the statement a. Therefore, it is necessary to be able to skip operations not existing in the usage scenario.

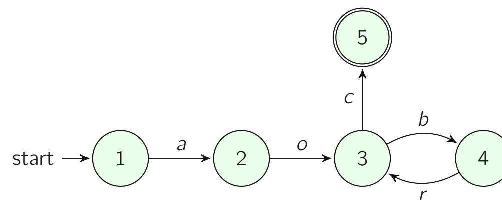


Figure 5.9: Graphical Representation of File Usage System K with additional statements.

The solution for skipping is relatively simple, adding self-loops to all nodes of the usage scenario, where each loop edge label consists of the additional statements. More formally, let $\mathcal{S}$ be a set of synchronising operations of a usage scenario U, plus $\mathcal{V}$ represents a set of additional statements used in program A such that $\mathcal{V} \cap \mathcal{S} = \emptyset$ and let $\mathcal{L} = \mathcal{V} \cup \mathcal{S}$ be the set of all operations in A. Moreover, let $\mathcal{M}_n(\mathcal{L})$ be the set of all matrices of size n with entries $\in \mathcal{L}$. The identity matrix of size m is denoted by $I_m \in \mathcal{M}_m$ and the zero matrix of size n by $Z_n \in \mathcal{M}_n$. Let $A \in \mathcal{M}_n(\mathcal{L})$ and $U \in \mathcal{M}_m(\mathcal{S})$. Furthermore $A_\mathcal{V} + A_\mathcal{S} = A$, $A_\mathcal{V} \in \mathcal{M}_n(\mathcal{V})$ and $A_\mathcal{S} \in \mathcal{M}_n(\mathcal{S})$. The "modified" Kronecker product (Skip operation) shall be introduced as $\odot$, with the conditions $a \cdot a = a$ and $a \cdot b = 0$ shall apply for all $a, b \in \mathcal{S}$, as only the synchronisation edges are of relevance. From the matrix point of view, self-loops can be added by multiplying an identity matrix with the scalar $\sum_{x \in \mathcal{V}} x$. Eq. (5.12) depicts the skip operation.

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

$$A \odot U = A \otimes \left(U + \left(\sum_{x \in \mathcal{V}} x \right) \cdot I_m \right) \tag{5.12}$$

Eq. (5.13) can easily be derived from Eq. (5.12).

$$A \odot U = A_{\mathcal{V}} \otimes I_m + A_{\mathcal{S}} \otimes U \tag{5.13}$$

With the skip operation, a program A can be checked whether it complies with usage scenario U. For the problem at hand in this dissertation, the skip operation is an essential part of another Kronecker operation.

5.6 Kronecker Sum

Another required Kronecker operation is called Kronecker Sum, denoted with symbol $\oplus$. This operation allows the identification of all interleavings between programs A and B. One condition is that program A, in whatever state, must accept arbitrary transitions of B, and B must do the same for A. Let $A \in M_n(\mathcal{A})$ and $B \in M_m(\mathcal{B})$ such that $\mathcal{A} \cap \mathcal{B} = \emptyset$. I_n and I_m are the respective identity matrices with the size of the square matrices A and B. Again, the given condition for A and B requires self-loops in A and B as applied in Eq. (5.14).

$$B \oplus A = A \otimes \left(B + \left(\sum_{x \in \mathcal{A}} x \right) I_m \right) + \left(A + \left(\sum_{y \in \mathcal{B}} y \right) I_n \right) \otimes B \tag{5.14}$$

This leads to the ordinary sum of Kronecker products

$$A \oplus B = A \otimes I_m + I_n \otimes B. \tag{5.15}$$

For demonstration purposes, let A and B be represented by the FSMs shown in Figure 5.10.

Figure 5.10: The two FSMs A and B

Calculating $A \oplus B$ results in the FSM depicted in Figure 5.11.

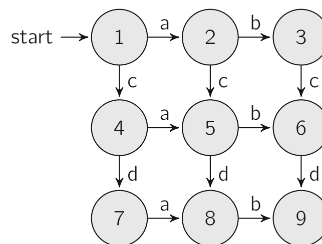


Figure 5.11: The resulting FSM of $A \oplus B$

The reader will agree that $A \oplus B$ represents the concurrent execution of the FSMs corresponding to A and B, i.e., it models all interleavings.

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

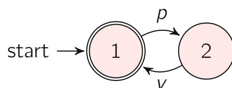


Figure 5.12: A simple Binary Semaphore.

5.7 Concurrent Programs and Semaphores

By identifying all interleaving between two programs, Kronecker Sum is a valuable tool to analyse concurrently executing program threads. Especially with the advent of multi-core processors, verifying multi-threaded applications is challenging. One reason is that the number of thread interleavings grows exponentially, causing a state explosion.

As introduced above, Kronecker Sum provides the means to analyse interleavings. However, a concurrent program with completely independent threads rarely exists in a practical setting. Threads usually communicate or synchronise with each other through accessing shared resources. One way to model synchronisation is to use synchronisation primitives, e.g., semaphores.

Semaphores [Dijna, Sta12, Dij02] are a well-known tool for process synchronisation and, therefore, efficiently implemented in modern operating systems (OSs). In its simplest and most restricted form, a semaphore (a so-called binary semaphore[6]) consists of two operations, p and v; Operation p acquires the resource while v releases it.

For example, if a thread already acquired (p) a semaphore, the following calling thread will be blocked and sent into the semaphore's first-in, first-out (FIFO) queue. After the first thread releases (v) the resource, the subsequent thread in the queue resumes its execution (p). To ensure that there is no way to manipulate the semaphore while locked, the operations p and v need to be atomic operations. Semaphores should not be mixed up with mutexes, as those have an "owner". That implies that a mutex can only be unlocked by the thread that locked the mutex.

Figure 5.12 shows a simple representation of a binary semaphore that will be used during the dissertation to synchronise different threads. As Kronecker Algebra does not involve time, queues and other implementation details are irrelevant.

To utilise semaphores with Kronecker Sum, we assume that $A \in M_n(\mathcal{A})$, $B \in M_m(\mathcal{B})$ and $\mathcal{A} \cap \mathcal{B} \subseteq \mathcal{S}$, with $\mathcal{S}$ representing the synchronisation operations. By using the same idea as in Kronecker Skip, adding loops on both sides, we start from the same Eq. (5.14) above. However, this time only the synchronisation operations are relevant. Therefore, we can model a system with two concurrent threads and one semaphore by adding a semaphore via the skip $\odot$ operation in the following way:

$$(A \oplus B) \odot S = (A \otimes I_m + I_n \otimes B) \odot S \tag{5.16}$$

This result is generalisable for a finite number of threads and semaphores. Let T_i for $i = 1, \ldots, n$ concurrently executing threads and S_j for $j = 1, \ldots, k$ be semaphores. Then the program consisting of T_i and S_j can be modelled by:

$$\bigoplus_{i=1}^{n} T_i \odot \bigoplus_{j=1}^{k} S_j \tag{5.17}$$

[6] There are other types such as counting semaphores or initially un/locked semaphores. Within this dissertation, only binary semaphores are of importance.

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

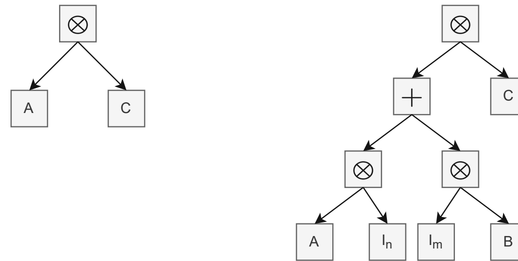


Figure 5.13: Expression tree examples: On the left $A \otimes C$ and on the right $(A \oplus B) \otimes C = (A \otimes I_n + I_m \otimes b) \otimes C$.

The use of semaphores also allows the detection of deadlocks in programs. A famous concurrent programming example is the Dining Philosophers problem, introduced by Dijkstra in 1971 [Dij71].

5.8 Implementing Kronecker Algebra Operations (Lazy Algorithm)

One of the significant problems of Kronecker Algebra is the size of the resulting matrices. Considering two CFGs of m and n nodes, Kronecker $\otimes$ produces an adjacency matrix of m-by-n nodes. Although most matrices are sparse, (i.e., they contain only a few non-zero elements), even standard memory saving methods are insufficient. Therefore, the implementation used in this dissertation utilises *lazy evaluation* [HM76]. Lazy evaluation of Kronecker operations delays all computations until required, i.e., in the end, only reachable nodes are analysed.

5.8.1 Expression Trees

Thus, a Kronecker expression is converted to an *expression tree*, followed by lazy evaluation of the tree's operations. Sparse matrices represent the (input) CFGs and form the leaves of the expression tree, while the inner nodes of the tree represent Kronecker $\otimes$ and standard $+$ matrix operations. There is no computation necessary until the resulting graph is required. Figure 5.13 depicts on the left the expression tree for $A \otimes C$ and on the right a more complex operation $(A \oplus B) \otimes C = (A \otimes I_n + I_m \otimes b) \otimes C$. Note, it is only necessary to handle $\otimes$ and $+$, because $\oplus$ and $\odot$ are defined via $\otimes$ and $+$.

5.8.2 Lazy Evaluation of Kronecker Expressions

For the lazy evaluation it is essential to identifying successors in order to determine the entries of the resulting matrix. For a better understanding, the simplest case is two matrices $A \otimes B$, that result in M. Let the size of matrix A be denoted by m, that of B by n. In this case, the expression tree has only two leaves A and B connected by $\otimes$. Now, the task is to find the non-zero entries of row i, in M. We can achieve this by finding the entries $a_{s,t}$, and $b_{u,v}$ (successors) in A and B that create the entries of row i. The first step requires the relevant rows s and u in A and B. For $a_{s,t}$, this can be done by $s = ((i-1) \div n) + 1$ ($\div =$ integer division), as M is built up by blocks of the size n. In the case of $b_{u,v}$, the operation $u = (i \bmod n)$ (the remainder of the division of i by n), returns the relevant row. If u turns out to be 0 by the above computation, we set $u = n$. We can

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

```
1  list succ(Matrix M, node i) n = M.getBMatSize;
2  s = ((i -1) ÷ n) + 1;
3  u = (i % n);
4  if u=0 then
5  |   u=n
6  su_a = succ(M.A, s);
7  su_b = succ(M.B, u);
8  switch M.getOperation do
9      case " + " do
10         S_i = su_a ∪ su_b;
11     case " ⊗ " do
12         S_i = ∅;
13         foreach l ∈ su_a do
14             t= getEdgeLabel(M.A, s, l);
15             foreach r ∈ su_b do
16                 z= getEdgeLabel(M.B, u, r);
17                 if t=z then
18                     v=(su_a[l] × n) + su_b[r];
19                     S_i := S_i ∪ {v} ;
20             end
21         end
22     end
23 end
24 return S_i
```

Algorithm 1: Lazy Evaluation of Kronecker Expressions (Successor Search)

now identify the successors of node (row) s in A and u in B. Since both matrices are sparse, this can be done in a simple way.

By constructing the semantics of $\otimes$, the successors retrieved from A and B can now be used to construct the successors of node (row) i in $M = A \otimes B$. After initializing $\mathcal{S}_i$ (set of successors of i) to $\emptyset$, two loops take a closer look at the successor lists su_a and su_b. The first loop retrieves the edge label $t = A(l, s)$ for $l \in su_a$. Then another loop retrieves the edge label $l = B(r, u)$ for $r \in su_b$. If $t = l$, the successor index v is calculated as $v = (su_a[l] \times n) + su_b[r]$ and added to $\mathcal{S}_i$. In the end, $\mathcal{S}_i$ contains the set of successors of node i. Similarly, the label entry $M(i, j)$ (row i and column j) can be determined.

Algorithm 1 shows the above-described *lazy evaluation*. The algorithm constructs parent nodes in a bottom-up fashion, and newly found successors are evaluated later. Moreover, the algorithm returns a list of successors and provides a case condition that considers simple "+" plus additions that are not based on " $\otimes$ ". In such a case, the list of successors is the unification of the two successor lists retrieved from the lower operations/matrices in the tree.

As a concluding remark, all algorithms are implemented in Ada. Currently, there is no user interface, and every example is a stand-alone program requiring compilation and execution in the terminal. The matrices of the FSMs and CFGs are stored in text files, and the resulting graphs are saved in the Dot language. In future work, we plan to reimplement part of the code and add a repository with examples for Kronecker operation applications.

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

5.9 Related work in the field of Kronecker Algebra

It is commonly claimed that research can only contribute to knowledge if we are *"standing on the shoulders of giants"*. [7] This metaphor also applies to this dissertation, without the contributions of Blieberger, Mittermayr and Burgstaller, in the areas of deadlock and WCET analysis in concurrent programs [MB21], static analysis of barriers [MB16b] and protected objects [BB14] in Ada programs with Kronecker Algebra, or their work in showing further applications, for example in train scheduling [SSB18], this work would not be possible.

While not entirely aligned with this dissertation, other works are related to creating graph models for concurrent programs. The work of Buchholz and Kemper [BK02] focuses on generating reachability sets in composed automata. They use the plain Kronecker product to describe networks of synchronised automata. The difference to this dissertation is that they use Boolean matrices (one matrix for each label) compared to our matrix entries based on the defined semiring. Another difference is that our approach can use semaphores for synchronisation. In addition, our implementation of Kronecker operation utilises lazy calculation for better performance. The output CFGs of our operation can be analysed with different techniques (e.g. [RP86, RP88, KU76]) and symbolic analysis (e.g. [BSB12])

As mentioned before, though not closely related, is the work of Plateau in the field of stochastic automata networks [Pla85] or the work of Ciardo and Miner about generalised stochastic Petri nets [CM99]. Petri nets [Pet62, Rei12] are a standard tool to model concurrent systems. However, they present a rather monolithic global system view, not on the individual threads. This focus on the global system was a reason not to use Petri nets, as the more interesting interactions between the components would get lost. Another aspect was that Petri nets could not be generated from the source code. Compared to our approach, we can model independent components, such as thread and synchronisation primitives, in the form of CFGs. After Kronecker operations, the resulting CFGs represent a global system view (including all interleavings and synchronisations) and can be automatically generated.

5.10 Concluding Remarks

All Kronecker operations introduced in this chapter are, in one way or another, used in the following chapters. We will extend some operations and add new ones. In some rare cases, we change some definitions, but that will be pointed out explicitly. Moreover, each of the following chapters will contain related work specific to the topic.

[7] The phrase "standing on the shoulders of giants" is a metaphor which means "using the understanding gained by major thinkers who have gone before in order to make intellectual progress". While not clearly attributed to someone, the phrase was used in a letter by Isaac Newton to Robert Hooke. [Tur59]

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

Chapter 6

Finding Implied Scenarios

Building on the insights in Chapter 4, this chapter looks into identifying basic emergence caused by implicit scenarios in agent systems. Therefore, we introduce the basic idea behind implicit scenarios and their relation to emergent behaviour, followed by a definition of message sequence charts (MSCs). The MSCs play an essential role in the presented method since they can be synthesised into CFGs using a new Kronecker operation. After presenting each step of the method, we apply and evaluate it on a simple example and discuss it compared to related work.

6.1 The Idea Behind Implied Scenarios

While the novelty aspect of emergence is widely accepted, its identification remains controversial. One difficulty is the rapidly increasing complexity of systems that exhibit emergence. Researchers introduced categories that describe how a system exhibits emergent behaviour to address this problem [DWH04]. The basic category of emergence is closely related to computational emergence because there is still the possibility of identifying the causes of unexpected behaviour.

One root cause for basic emergence is that unintended agent interactions cause the new behaviour. Such implicit scenarios are the result of the synthesis of several intended agent scenarios. Usually, the unexpected system behaviour only occurs at runtime and only partially satisfies the novelty requirement. Some approaches focus on the component or system level to identify implied scenarios at the design stage [BGMK07]. Component-level synthesis typically misses interactions between components [MMBH10] that are inherent in system-level approaches [HK02]. Both approaches have in common that the scenarios are often captured in MSCs and the system states in state machines.

Figure 6.1 depicts the idea of system-level synthesis by synthesizing different scenarios depicted as MSCs. The result of the synthesis represents all scenarios executable by the system, including implied scenarios. Higher levels of emergence cannot be found in this representation, as the agent's interactions are still deterministic.

6.2 Message Sequence Charts

In general, message sequence charts (MSCs) are diagrams used in software engineering to illustrate the order of messages exchanged between two or more objects [Boo94]. MSCs can document both synchronous and asynchronous messaging systems [KGSB99]. They are commonly used in

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

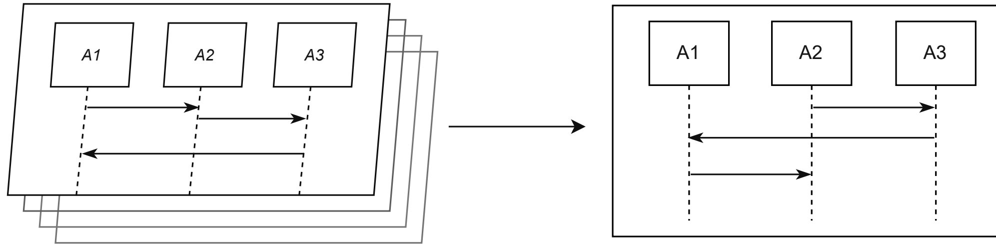


Figure 6.1: The principle of scenario synthesis

conjunction with state diagrams, which show the states that each object can assume, and activity diagrams, which show the flow of control between objects. The main elements of an MSC are lifelines and messages, defined as follows.

A **lifeline** represents an instance of a state machine, distinguished by its name. Different instances of the same state machine may exist in an MSC. Therefore, given a set M of state machines and the alphabet $\sum_L$, a lifeline is a pair $L = (l, M^*)$ where $l \in \sum_L$ is the name of the lifeline and $M^* \in M$ is an instance of a state machine. For later definitions, let $sm(L)$ represent M of a lifeline.

Messages are the communication elements between lifelines. Considering an alphabet $\sum_A$ and a set $\mathcal{L}$ of lifelines, a message can be defined as a triple (σ, a, ρ) where

- $\sigma \in \mathcal{L}$ is the sending lifeline,
- $a \in \sum_A$ is the message symbol, and
- $\rho \in \mathcal{L} \backslash \{\sigma\}$ is the receiving lifeline.

In this dissertation, the sender σ and receiver ρ lifeline are FSMs.

A **message sequence chart (MSC)**, therefore, can be formally defined as a pair $(\mathcal{L}, \mu)$ where

- $\mathcal{L}$ is a set of lifelines over M and $\sum_L$
- with pairwise distinct names and
- $\mu = [m_1, ..., m_n]$ is a message sequence where for each $(\sigma, a, \rho) \in \mu$ must hold $\sigma, \rho \in \mathcal{L}$ and $a \in \sum_A$.

Next we introduce further limited semantic definitions of an MSC. In this chapter, we do not need all of them, but in Chapter 7, they will be relevant. We specify the interactions via messages between the lifelines by a **Global State** $\hat{s}$ with a set of $\mathcal{L} = L_1, ..., L_l$ lifelines, where $sm(L_i) = (S_i, \iota_i, A_i, T_i)$ is the state machine of the lifeline L_i, for $1 \leq i \leq l$. The global state $\hat{s}$ is a tuple $(s_1, ..., s_l) \in S_1 \times \cdots \times S_l$. Each global state contains sets of messages to be sent and received. After sending or receiving an admissible message, the global state $\hat{s}$, changes to a global successor state $\hat{s}' = (s_1, \ldots, s_s', \ldots, s_r' \ldots, s_l) \in S_1 \times \cdots \times S_l$. That brings up how to decide if a message is admissible.

A message $m = (L_s, a, L_r)$ with $L_s \in \mathcal{L}, L_r \in \mathcal{L}, L_s \neq L_r$, and $a \in \sum_A$ is admissible in a global state $\hat{s}$ under the following circumstance. The states of the sender L_s and receiving lifeline L_r need to accept a transition label equal to message a, i.e., $(s_s, ev, a, s_s') \in T_S$, and $(s_r, a, ev, s_r') \in T_r$.

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

Messages in the set of admissible messages in a global state may be independent subsets. Such messages have no sender or receiver in common and can therefore be executed in parallel, in the form of a transaction.

A **transaction** represents a nonempty set $m = \{m_1, \ldots, m_t\}$ of messages, and each message must be pairwise distinct in the following way. Let $i, j \in \{1, \ldots, t\}, m_i = (\sigma_i, a_i, \rho_i)$, and $m_j = (\sigma_j, a_j, \rho_j)$. A message set is pairwise distinct if it holds that all $\sigma_i, \sigma_j, \rho_i$, and ρ_j are pairwise distinct. An admissible transaction applies all of the transaction's messages to the global state. Therefore all messages must be admissible. Moreover, there is no rule that a transaction must contain more than one message, i.e., a transaction can be a singleton. Such a viewpoint represents a sequence of messages, as depicted in an MSC, as a sequence of singleton transactions.

A sequence of transactions is, therefore, a **path** μ that connects a global state $\hat{s}_0$ to a global state $\hat{s}_k$. Such a path can be defined as a sequence $\mu = [m_1, \ldots, m_k]$ of transactions such that there exists a sequence $[\hat{s}_0, \ldots, \hat{s}_k]$ of global states where all $1 \leq i \leq k$, m_i are admissable in the state $\hat{s}_{i-1}$ and leads to $\hat{s}_i$. Moreover, $\hat{s}_i$ needs to be the global successor state of $\hat{s}_{i-1}$ after applying m_i.

6.2.1 Problem Formulation

After we introduced MSCs, we can define the problem of implied scenarios more precisely. The possibility of implied scenarios exists if the combination of several system scenarios (i.e., agent interactions depicted in MSCs) creates sequences of messages (paths μ) that connect some global state $\hat{s}_i$ and the global initial state $\hat{s}_0$ not foreseen in the system scenarios.

Therefore, the problem of implied scenarios can be defined as:

A combination of MSCs can contain implied scenarios if there is a sequence of messages μ present that connect the global state $\hat{s} = (\iota_1, \ldots, \iota_l)$ with any other global states $\hat{s}'$ or $\hat{s}''$, not defined in the single MSCs. Given an MSC $= (\mathcal{L}, \mu)$, with $\mathcal{L}\{L_1, \ldots, L_l\}$ and $sm(L_i) = (S_i, \iota_i, A_i, T_i)$ for $1 \leq i \leq l$ and the alphabets $\sum_A$ and $\sum_L$.

6.2.2 Limitations

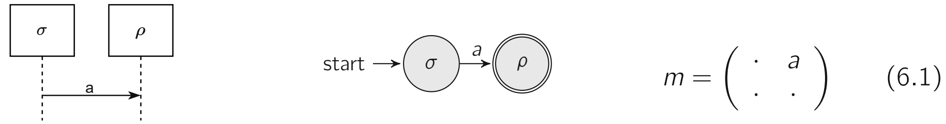
Readers familiar with UML or other modelling languages might recognize that the given definitions for the MSCs are limited. Advanced control flow structures such as hierarchical or history states in state machines or "alt", "loops", fragments in sequence charts, or exception handling are out of scope. The reason is that some structures create indeterministic behaviour. Therefore, the scope of this chapter is limited to control flow structures that simple FSMs can represent.

6.3 Finding Basic 'Emergence'

Based on the problem formulation, the goal is to provide a method to combine different MSCs into a large system representation and to identify new paths that are not present in the individual MSCs. We use the previously described Kronecker operations (cf. Chapter 5) to solve this problem. Below we present the steps included in our proposed method, including:

- Scenario collection
- Synthesize the different MSCs to CFGs

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub



$$m = \begin{pmatrix} \cdot & a \\ \cdot & \cdot \end{pmatrix} \tag{6.1}$$

Figure 6.2: A message "m" with label "a" depicted as a single statement and its matrix representation.

- Combine all CFGs to one system representation FSM ($\mathcal{S}_{\text{Total}}$)
- Analyze $\mathcal{S}_{\text{Total}}$ with Kronecker Algebra
- Evaluate new scenarios

6.3.1 Scenario Collection

The first step focuses on collecting all possible scenarios of the system in question. We assume that some representations of the various scenarios in MSCs are available. As MSCs have gained wide acceptance for scenario-based specifications of component and system behaviour and are present in modern modelling languages such as UML, SysML, or AutomationML [Boo94], this is a reasonable assumption. Currently, the following synthesis process supports only the fundamental sequential interactions between agents. More complex constructs, such as parallel composition, choice, repetition, and hierarchic decomposition [KGSB99], could be handled by Kronecker Algebra but are out of scope. Moreover, if not all scenarios are available, the presented method might not discover some unintended behaviour of the system. However, it is impossible to predict which scenario combinations create a new behaviour until synthesised.

6.3.2 Synthesizing MSCs to CFGs

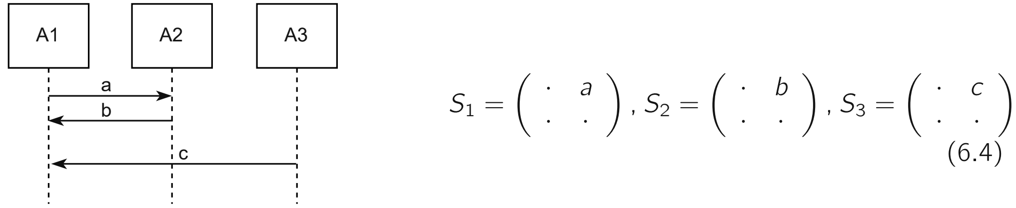
An essential step to identify implied scenarios is that all intended scenarios must be combined into one total system representation. However, the scenarios are contained in various MSCs, unsuitable for such a synthesis. Therefore, the question is how to transform an MSC into a matrix representation applicable to our Kronecker operations while reflecting the underlying semantics of the MSC. For this reason, we introduce Kronecker Synthesize operation.

Let us first look at a single message m. As defined before, a message in an MSC is a triple $m = (\sigma, a, \rho)$, a sender, a label and a destination, as depicted in Figure 6.2. The reader will agree that a message, m, can also be seen as a transaction or a program statement (e.g., set a value to a variable). Such a program statement in a CFG has one initial and final node and one edge with a label. Therefore, we can transform a message m into a CFG as in Figure 6.2, depicted in the middle and the respective matrix representation on the right.

For generalisation, let S_i be a single statement and α_i the respective label, representing a message m with a label "a", as shown in Eq. (6.2).

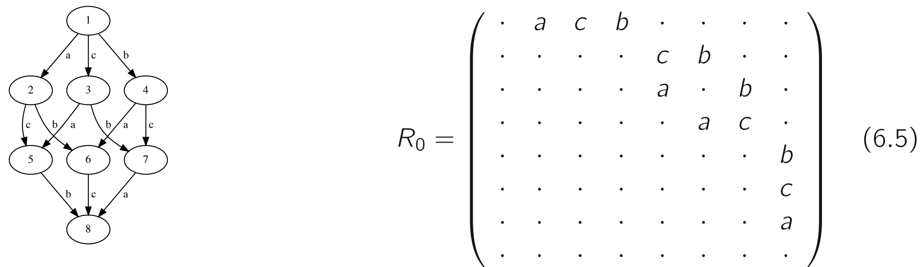
$$S_i = \begin{pmatrix} \cdot & \alpha_i \\ \cdot & \cdot \end{pmatrix} \tag{6.2}$$

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub



$$S_1 = \begin{pmatrix} \cdot & a \\ \cdot & \cdot \end{pmatrix}, S_2 = \begin{pmatrix} \cdot & b \\ \cdot & \cdot \end{pmatrix}, S_3 = \begin{pmatrix} \cdot & c \\ \cdot & \cdot \end{pmatrix} \tag{6.4}$$

Figure 6.3: An example MSC "Z" with three messages "a", "b", "c", and their matrix representations S_1, S_2, S_3.



$$R_0 = \begin{pmatrix} \cdot & a & c & b & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & c & b & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & a & \cdot & b & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & a & c & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & b \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & c \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & a \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \end{pmatrix} \tag{6.5}$$

Figure 6.4: The resulting graph of $R_0 = S_1 \oplus S_2 \oplus S_3$ on the left, and on the right the matrix representation.

Usually, an MSC contains a number of (n) messages that create a scenario together. As defined above, S_i stands for one message; therefore, the combination of all n statements should represent the scenario depicted in an MSC. For creating a combined CFG R_0, we can use Kronecker Sum, as shown in Eq. (6.3). As described in Section 5.6, Kronecker Sum will model all possible interleavings.

$$R_0 = S_1 \oplus S_2 \oplus \ldots \oplus S_n = \left(\bigoplus_{i=1}^{n} S_i \right), \text{for } 1 \leqslant i \leqslant n \tag{6.3}$$

Nevertheless, let us first introduce a simple example. Figure 6.3 depicts an MSC (Z) with three agents interacting with three messages. Based on the definition above, Eq. (6.4) shows the S_i's for the messages a, b and c.

The next step is to apply Kronecker Sum on S_1, S_2 and S_3 as defined in Eq. (6.3). Figure 6.4 shows this calculation's resulting CFG. R_0 represents all possible interleavings or, in our case, message orders, how the messages could appear. However, that is not the correct representation for our example MSC, as the messages have dependencies. Defining those dependencies strongly depends on the applied semantics of the MSC and may vary from case to case.

Let $\mathcal{L}$ be the set of all operations in a program E. $\mathcal{L}$ is a given set of operations, with the condition $\mathcal{L}_{sync} \subseteq \mathcal{L}$ and $\mathcal{L}_{nsync}$ are the remaining operations of E, i.e., $\mathcal{L} \backslash \mathcal{L}_{sync} = \mathcal{L}_{nsync}$. Now we need to introduce a way to specify the dependencies between a message pair $\alpha_k \in \mathcal{L}_{sync}$ and $\alpha_l \in \mathcal{L}_{sync}$, in a flexible way. This problem can be solved again with a CFG. This time we use a CFG with three states. The order of the edge labels in the CFG defines the dependency of the message pair α_k and α_l, (i.e., $\alpha_k \prec \alpha_l$).

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

Therefore, the matrix representation of a dependency between a message pair α_k and α_l is:

$$D_i(\alpha_k, \alpha_l) = \begin{pmatrix} \cdot & \alpha_k & \cdot \\ \cdot & \cdot & \alpha_l \\ \cdot & \cdot & \cdot \end{pmatrix} \tag{6.6}$$

For our example, we only have one dependency, namely $a \prec b$. We only know that message a precedes b; the other dependencies are not given as we do not know in which order the messages will arrive. However, that is one possible interpretation (semantic). Therefore we get the following:

$$D_1 = \begin{pmatrix} \cdot & a & \cdot \\ \cdot & \cdot & b \\ \cdot & \cdot & \cdot \end{pmatrix} \tag{6.7}$$

Those dependencies $(D_1, \ldots, D_s)$ must be applied to R_0. This requires two new matrix operators ($\mathcal{O}$ and $\mathcal{W}$). The first operation $\mathcal{O}$ restricts the matrix entries to those specified in the dependency matrix D_i. These labels are considered synchronised labels, and the $\mathcal{O}$ operation removes the other entries; these are the non-synchronised labels. On the other hand, $\mathcal{W}$ restricts the matrix entries to those not specified in the dependency matrix D_i, thus retaining the non-synchronised labels.

Let $\mathcal{M}_n(\mathcal{L})$ be the set of all matrices of size n with entries $\in \mathcal{L}$, and set $E = e_{ij} \in \mathcal{M}_n(\mathcal{L})$. Let, $\mathcal{Q}, \mathcal{T} \subseteq \mathcal{L}$. Therefore, $\mathcal{O}(\mathcal{Q})[E] = (\mathcal{O}_{ij})$ is a matrix of the same size as E that contains entries $\in \mathcal{Q}$, and $\mathcal{W}(\mathcal{T})[E] = (\mathcal{W}_{ij})$ is a matrix of the same size as E that contains entries $\notin \mathcal{T}$. In more detail,

- if $e_{ij} \in \mathcal{Q}$, then $\mathcal{O}_{ij} = e_{ij}$, otherwise $\mathcal{O}_{ij} = 0$,
- and if $e_{ij} \notin \mathcal{T}$, then $\mathcal{W}_{ij} = e_{ij}$, otherwise $\mathcal{W}_{ij} = 0$.

After having defined $\mathcal{O}$ and $\mathcal{W}$, we can use Kronecker product on R_0, as shown in Eq. (6.8). The equation is applied iteratively, and each step enforces a dependency $D_i(\alpha_k, \alpha_l)$ on R_i for $0 \leqslant i \leqslant s$. Note that the order of applying the dependencies (D_s) is irrelevant, i.e., it does not matter which dependency is first or last.

$$R_i = \mathcal{O}(\{\alpha_k, \alpha_l\})[R_{i-1}] \otimes D_i(\alpha_k, \alpha_l) + \mathcal{W}(\{\alpha_k, \alpha_l\})[R_{i-1}] \otimes I_{|D_i(\alpha_k, \alpha_l)|} \tag{6.8}$$

The resulting matrix R_s then represents a CFG with some behaviour like the original MSC. We only need one iteration for our example as there is only one dependency pair D_s. Therefore R_1 is calculated

$$R_1 = \mathcal{O}(\{a, b\})[R_0] \otimes D_1(a, b) + \mathcal{W}(\{a, b\})[R_0] \otimes I_{|D_1(a,b)|}, \tag{6.9}$$

with the operand,

$$\mathcal{O}(\{\alpha_k, \alpha_l\})[R_{i-1}] = \mathcal{O}(\{a, b\})[R_0] = \begin{pmatrix} \cdot & a & \cdot & b & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & b & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & a & \cdot & b & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & a & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & b \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & a \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \end{pmatrix}, \tag{6.10}$$

TU Bibliothek WIEN Your knowledge hub
Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.

the operand

$$\mathcal{W}(\{\alpha_k, \alpha_l\})[R_{i-1}] = \mathcal{W}(\{a, b\})[R_0] = \begin{pmatrix} \cdot & \cdot & c & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & c & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & c & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & c \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \end{pmatrix} \tag{6.11}$$

and the identity matrix I_n (with $n = 3$, as the size of $D_i(\alpha_k, \alpha_l)$ is 3×3.)

$$I_{|D_i(\alpha_k,\alpha_l)|} = I_3 = \begin{pmatrix} 1 & \cdot & \cdot \\ \cdot & 1 & \cdot \\ \cdot & \cdot & 1 \end{pmatrix}. \tag{6.12}$$

After executing all calculations, we obtain for R_1 the following matrix Eq. (6.13). The matrix has a size of 24×24, yet the reader will realise that the matrix is sparse, as introduced in Section 5.2. This matrix is a perfect example of why the lazy algorithm (cf. Section 5.8) decreases the computation effort. Only the actual entries' successors are computed.

$$R_1 = \left(\begin{array}{ccc|ccc|ccc|ccc|ccc|ccc|ccc|ccc}
\cdot & \cdot & \cdot & \cdot & \color{red}{a} & \cdot & \color{red}{c} & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & c & \cdot & \cdot & \cdot & b & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & c & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\hline
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & c & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \color{red}{c} & \cdot & \cdot & \cdot & \color{red}{b} & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & c & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\hline
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \color{red}{a} & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & b & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\hline
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & a & \cdot & c & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & c & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & c & \cdot & \cdot & \cdot \\
\hline
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \color{red}{b} \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\hline
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & c & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & c & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \color{red}{c} \\
\hline
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & a & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\hline
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot
\end{array}\right) \tag{6.13}$$

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

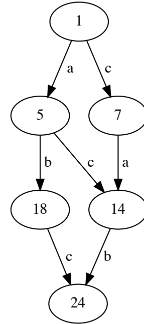


Figure 6.5: The resulting graph of matrix Eq. (6.5) depicting R_1.

The next step is to compute the start and final nodes, resulting in $S = 1$ and $F = 24$. Afterwards, the resulting graph can be constructed by starting from row 1 and finding the successors of each edge. We marked the reachable entries in red; the other entries cannot be reached and are irrelevant to the resulting graph. The reader will agree that Figure 6.5 shows the graph representing R_1.

Let us look closer at R_1. No edge label b is issued in each path before a. Thus, dependency $a \prec b$ is fulfilled. All other possible message orders are intact; no path is lost. Another interesting fact is that the node count is much smaller than the entries in matrix 6.13. As mentioned before, not all edges are reachable, creating no path in the graph. The node numbers do not play a role, as they are a mere "position" in the matrix.

In summary, Kronecker Synthesize operation shown in Eq. (6.8) can convert an MSC into a CFG, with the dependencies reflecting the semantics used in the MSC. Therefore, the semantics are not specified explicitly but only reflected in the dependencies D_i. We will use Kronecker Synthesize later in an example in Section 6.4.

6.3.3 Combine all Scenarios and Create $\mathcal{S}_{\text{Total}}$

As a next step, we combine all previously synthesised CFGs. We again use Kronecker Sum for this task. However, there are some conditions. Kronecker Sum works only with disjoint edge labels, i.e., the state transition labels must be different. In case of the labels are not disjoint, the resulting labels will not be distinguishable. It is self-explaining that all CFGs need to be transformed into their matrix representation, as described in Section 5.2.1. The resulting matrix $\mathcal{S}_{\text{Total}}$ Eq. (6.14) represents the entire system, including all possible interleavings. We introduce m as the number of CFGs.

$$\mathcal{S}_{\text{Total}} = R_1 \oplus R_2 \oplus \ldots \oplus R_m = \left(\bigoplus_{i=1}^{m} R_i \right) \text{ for } 1 \leqslant i \leqslant m \tag{6.14}$$

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

6.3.4 Analyze $\mathcal{S}_{\text{Total}}$ with Kronecker Algebra

This step might not always be executed in the same way as it depends on the system designer's aim. Several of the previously introduced Kronecker operations can be used to analyze $\mathcal{S}_{\text{Total}}$. In the simplest case, the system designer visually analyses $\mathcal{S}_{\text{Total}}$, identifies new paths, and evaluates if they create unintended behaviour. However, in most cases, $\mathcal{S}_{\text{Total}}$ will be rather large, which makes a visual check impossible. In this section, we propose the following Kronecker operations:

- Kronecker product to check if a forbidden path exists.[1]
- Utilise Kronecker Skip to find a sequence of messages (that can be interrupted by other labels) in $\mathcal{S}_{\text{Total}}$,
- and introduce semaphores to simulate shared resources in the system to reduce $\mathcal{S}_{\text{Total}}$ further.

In Section 6.4 each of the operations will be applied to an example and explained further. In summary, this step aims to support the designer in identifying new paths or scenarios that need to be evaluated.

6.3.5 Evaluate New Scenarios

Based on the potentially discovered new scenarios, the system designer must decide whether they represent unintended behaviour. If the new scenario is malicious, the designer can use it to reconstruct an MSC and identify the root cause. While this last step seems straightforward, it requires in-debt knowledge of the intended system behaviour. Therefore, the suggested process is only a support tool for the final decision. Nevertheless, all Kronecker operations can be automated, and the state explosion problem handled.

6.4 Evaluation Example

This section presents an example to illustrate the proposed steps and provide a better understanding. The greatly simplified example represents a centralised car locking mechanism.

Three agents, a controller and two motors steer two locks in the car doors. A driver can lock and unlock the car with a key or a button from the inside. In both cases, the controller will receive either a trigger signal $lck = lock$ or $unlck = unlock$. After receiving the triggers, the control unit will send two signals to the two motors, RM (Right Motor) and LM (Left Motor). Both motors are in the state OFF and change to DD (Drive-Down). As the two motors might not have the same speed to close the respective lock, control waits for an acknowledgement of both motors before changing to the state locked $LCKD$. After sending their acknowledgements, the motors change back to state OFF. To open the car, the control unit sends two unlock signals to the motors, which will again send two acknowledgements back after completing their state DU (Drive-Up). The control unit then changes to state unlocked $UNLD$ and the motors to the state OFF.

Figure 6.6 depicts the MSCs for the scenarios locking/unlocking. Both MSCs follow the definitions given in Section 6.2, with one exception. The rounded boxes represent the states of the agents to indicate the state changes the agents are going through. Table 6.1 summarises the message labels and the triggers.

[1] We acknowledge that a designer might not know all forbidden paths. A typical solution for this issue is to find counterexamples of intended paths, a method used in model-checking. However, this is out of scope and part of future work.

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

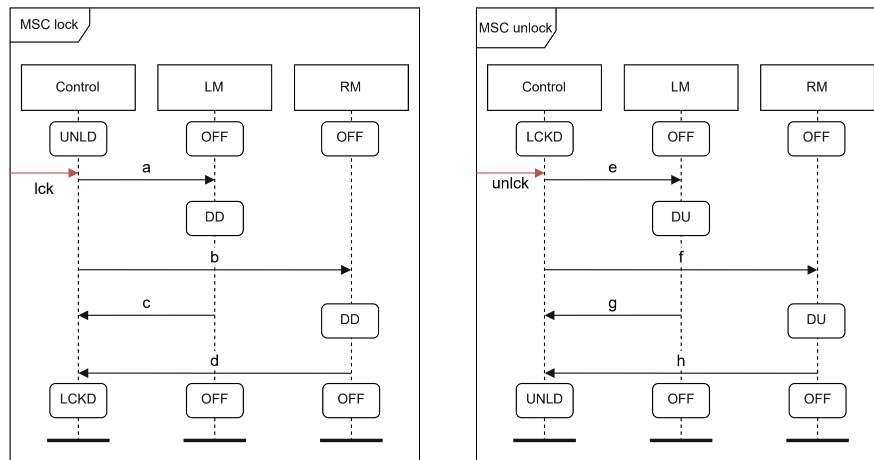


Figure 6.6: The two MSCs for the scenarios locking/unlocking in the car locking mechanism example.

Table 6.1: Triggers and messages (labels)

Trigger	Labels	Trigger	Labels
Lock	lck	Unlock	ulck
Messages		**Messages**	
Left lock (DD)	a	Left unlock (DU)	e
Right lock (DD)	b	Right unlock (DU)	f
Left ready	c	Left ready	g
Right ready	d	Right ready	h

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

6.4.1 Synthesising MSCs to CFGs

As described in the previous sections, each of the MSCs needs to be transformed into CFGs by utilising Kronecker Synthesize operation (cf. Section 6.3.2) before combining them. Synthesising the MSC lock in Figure 6.6 into the CFG $R_{i_{Lock}}$, requires creating $R_0 = S_1 \oplus S_2 \oplus \ldots \oplus S_n$. Eq. (6.15), shows the four matrix message label representations. On a side note, as Kronecker Sum $\oplus$ is an associative operation, the order of the matrices is unimportant.

$$R_0 = \begin{pmatrix} \cdot & a \\ \cdot & \cdot \end{pmatrix} \oplus \begin{pmatrix} \cdot & b \\ \cdot & \cdot \end{pmatrix} \oplus \begin{pmatrix} \cdot & c \\ \cdot & \cdot \end{pmatrix} \oplus \begin{pmatrix} \cdot & d \\ \cdot & \cdot \end{pmatrix} \tag{6.15}$$

The following matrices in Eq. (6.16), are the dependencies between the message labels (i.e., the order of message sending/receiving on a lifeline). As there is no synchronisation between the agents, we assume the messages c and d can arrive in a different order. At the same time, there are explicit dependencies between the message pairs $a \to b$, $a \to c$ and $b \to d$.

$$D_1 = \begin{pmatrix} \cdot & a & \cdot \\ \cdot & \cdot & b \\ \cdot & \cdot & \cdot \end{pmatrix}, \quad D_2 = \begin{pmatrix} \cdot & a & \cdot \\ \cdot & \cdot & c \\ \cdot & \cdot & \cdot \end{pmatrix}, \quad D_3 = \begin{pmatrix} \cdot & b & \cdot \\ \cdot & \cdot & d \\ \cdot & \cdot & \cdot \end{pmatrix} \tag{6.16}$$

To get $R_{i_{Lock}}$, in this case, $i = 3$; since there are three dependencies, Eq. (6.8) must be applied iteratively i times. The operations cannot be displayed since the matrices reach a size of 432×432, despite being mostly empty (sparse). Figure 6.7 depicts the resulting CFG $R_{i_{Lock}}$.

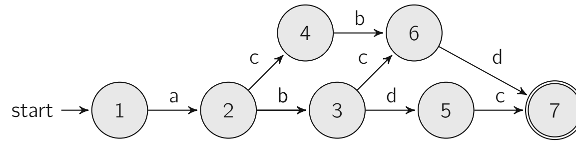


Figure 6.7: The CFG $R_{i_{Lock}}$ of MSC_{Lock}

The second MSC is synthesised in the same way and results in the CFG shown in Figure 6.8. As both MSCs have the same order of messages, just with different labels, the CFG $R_{i_{Unlock}}$ is isomorphic. However, that depends entirely on the chosen dependencies. For the shown example, we made this choice deliberately.

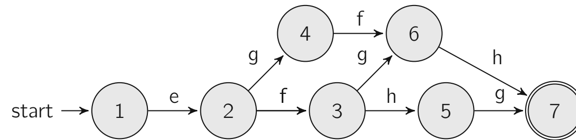


Figure 6.8: The CFG $R_{i_{Unlock}}$ of MSC_{Unlock}

6.4.2 Combine all Scenarios and Create $\mathcal{S}_{\text{Total}}$

As described in Section 6.3.3, it is necessary to combine all synthesised CFGs. For this step, we utilise Kronecker Sum Eq. (6.14) on R_{Lock} and R_{Unlock}, as shown below. Note: we omit the i in the following to improve readability.

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

$$R_{Lock} \oplus R_{Unlock} = R_{Lock} \otimes I_{R_{Unlock}} + I_{R_{Lock}} \otimes R_{Unlock} \tag{6.17}$$

Figure 6.10 depicts the combined CFG $\mathcal{S}_{\text{Total}}$. The CFG shows all interleavings between R_{Lock} and R_{Unlock} when executed concurrently. In other words, $\mathcal{S}_{\text{Total}}$ represents the synthesis of both MSCs considering the dependencies between the messages. As a side note, the beauty of Kronecker Synthesize operation is the possibility to apply further dependencies on $\mathcal{S}_{\text{Total}}$, i.e., dependencies between messages only occurring in separate MSCs. However, this possibility should be used with caution as it might remove the very paths that are most interesting.

6.4.3 Analysing the Graph with Kronecker Operations

After synthesising the MSCs into $\mathcal{S}_{\text{Total}}$, it is, to a certain extent, the developer's job to evaluate the paths. Nevertheless, analysing a large graph by "hand" is not feasible. Therefore, further Kronecker Algebra operations can be used to reduce the graph or search for forbidden paths.

Let us assume the developer wants to search for a path that leaves the two locks of the car in different states, a situation that might be unfavourable for a car-locking system. One possibility for such a situation is when the car owner closes the car and immediately changes his mind. However, a change of mind is not enough; additionally, the system must issue the message labels in a particular order. Essential is that the system issues the messages a and c followed by an e (when the owner opens the car again) and the other labels in the following order f, g, h, b and d. The left motor LM would change its state first into locked and then unlocked, while the right motor RM first into unlocked and locked, leaving one side of the car open and the other closed.

For the above situation, we know the sequence of labels that should not be present in $\mathcal{S}_{\text{Total}}$. Therefore, the labels can be transformed into a CFG $U_{Forbidden}$, as shown in Figure 6.9. As a next step, we can apply Kronecker product $\mathcal{S}_{\text{Total}} \otimes U_{Forbidden}$ to determine if this path exists in $\mathcal{S}_{\text{Total}}$. If the resulting CFG is isomorph to $U_{Forbidden}$, the developer knows that the possibility exists, and preventive measures must be implemented. In the example, the path exists and is also highlighted in red in Figure 6.10.

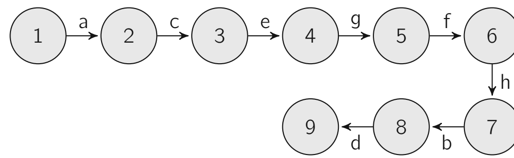


Figure 6.9: The CFG $U_{Forbidden}$

One might argue that usually only a part of a forbidden path is known, e.g., only a sequence of specific labels interrupted by other labels that are not relevant. For that case, we use Kronecker Skip operation introduced in Section 5.5. The skip operation allows comparing CFGs with larger systems. It cuts away all paths that do not conform to the given sequence and reduces $\mathcal{S}_{\text{Total}}$. The resulting CFG is empty if the sequence is not present.

In the example, we know that the message labels a, e, h, and d need to occur in this order to create the problem, while the other labels are irrelevant. Let us call this sequence U_{F2} and apply Kronecker Skip as shown in Eq. (6.18).

$$\mathcal{S}_{\text{Total}} \odot U_{F2} = \mathcal{S}_{\text{Total}\mathcal{V}} \otimes I_m + \mathcal{S}_{\text{Total}\mathcal{S}} \otimes U_{F2} \tag{6.18}$$

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

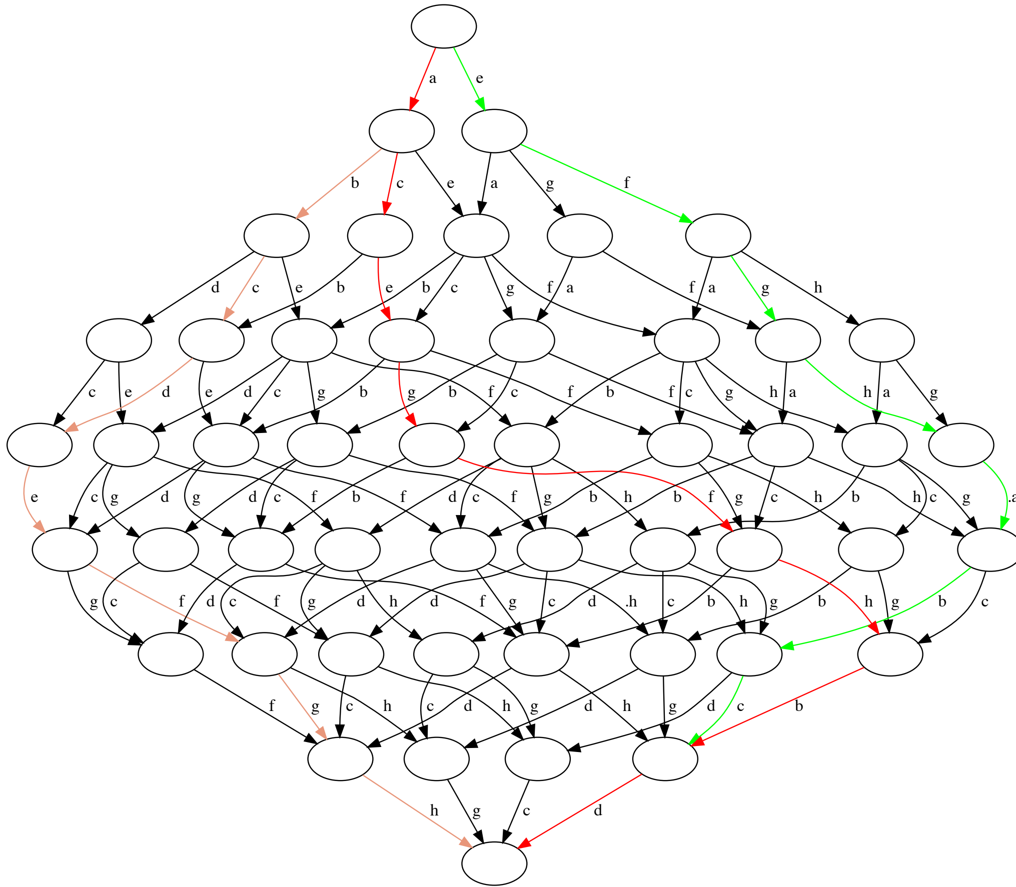


Figure 6.10: The combined CFG $\mathcal{S}_{\text{Total}}$

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Eq. (6.19) depicts U_{F2}, a 5×5 matrix representing the required label order. I_m is the identity matrix of the same size as U_{F2}. $\mathcal{S}_{\text{Total}\mathcal{V}}$ represents the matrix of $\mathcal{S}_{\text{Total}}$ without the labels contained in U_{F2} and $\mathcal{S}_{\text{Total}\mathcal{S}}$, the matrix of $\mathcal{S}_{\text{Total}}$ with the same labels as in U_{F2}.

$$U_{F2} = \begin{pmatrix} \cdot & a & \cdot & \cdot & \cdot \\ \cdot & \cdot & e & \cdot & \cdot \\ \cdot & \cdot & \cdot & h & \cdot \\ \cdot & \cdot & \cdot & \cdot & d \\ \cdot & \cdot & \cdot & \cdot & \cdot \end{pmatrix} \tag{6.19}$$

Figure 6.11 shows the resulting graph of Eq. (6.18). The red path, is again the same, as in Figure 6.9 as it fulfils the requirements. All other visible paths are possibilities where U_{F2} can appear.

Another functional ability of Kronecker Algebra is that it allows the introduction of semaphores. A semaphore is typically used to block a shared resource between concurrently executing programs. In this example, the different MSCs act like concurrent programs, while the controller is a shared resource between the other two agents. Let us assume locking the car issues label a, and the controller does not accept an unlock order until label d has arrived, and the same applies to unlocking with e and h. To ensure this controller behaviour, we need to introduce a semaphore that accepts a or e as an entry and d or h, as an exit edge label. The semaphore S_E shown in its matrix form in Eq. (6.20) fulfils these requirements.

$$S_E = \begin{pmatrix} \cdot & a & e & \cdot \\ \cdot & \cdot & \cdot & d \\ \cdot & \cdot & \cdot & h \\ \cdot & \cdot & \cdot & \cdot \end{pmatrix} \tag{6.20}$$

As previously, we can apply Kronecker Skip operation Eq. (6.21) to remove all paths from $\mathcal{S}_{\text{Total}}$ that do not conform to the semaphore S_E.

$$\mathcal{S}_{\text{Total}} \odot S_E = \mathcal{S}_{\text{Total}\mathcal{V}} \otimes I_m + \mathcal{S}_{\text{Total}\mathcal{S}} \otimes S_E \tag{6.21}$$

Figure 6.12 depicts the resulting graph. There are now only two paths left; one where label a is issued first and another where label e takes precedence. The effect of the semaphore in this example was expectable. However, introducing semaphores can create valuable insights, such as potential deadlocks in examples with several shared resources. To summarise, Kronecker Algebra provides a designer with several possibilities to analyse CFGs, of which we only present a few. The analysis results can be used to evaluate their potential effect on the system.

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

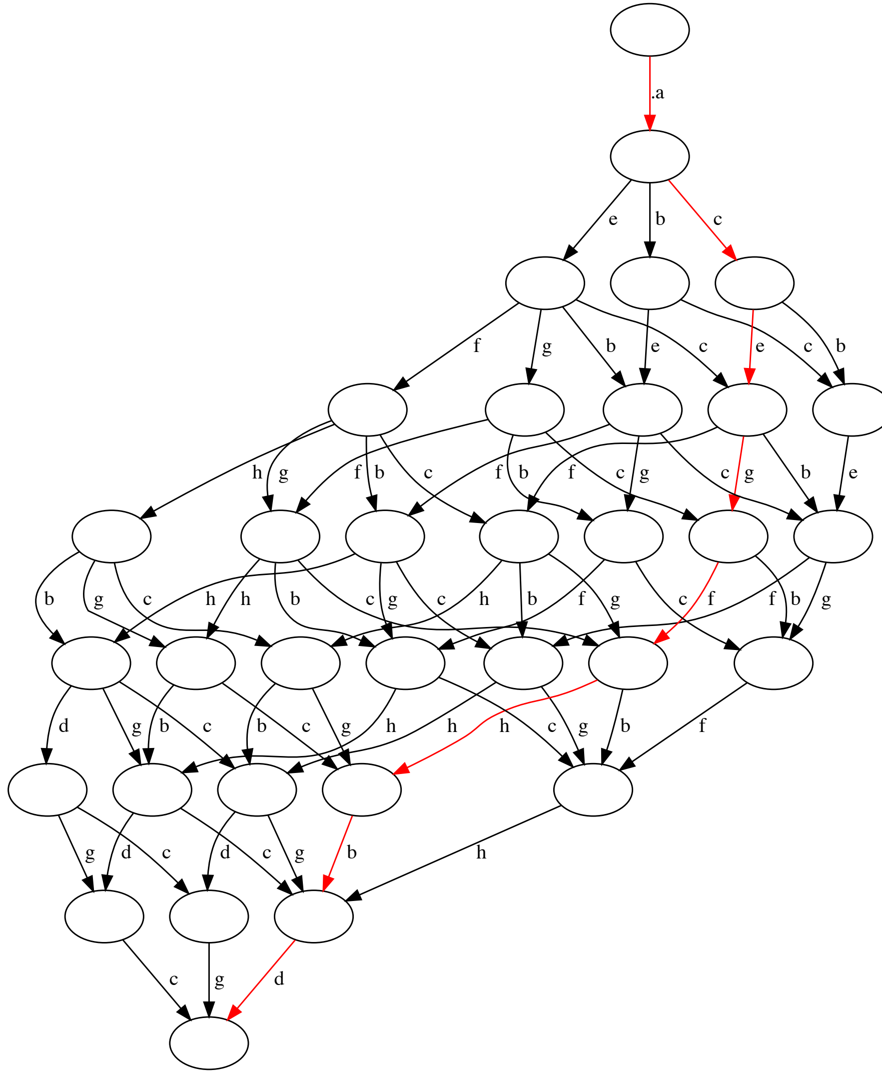


Figure 6.11: The resulting CFG of $\mathcal{S}_{\text{Total}} \odot U_{F2}$

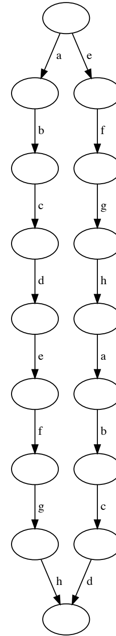


Figure 6.12: The resulting CFG of $\mathcal{S}_{\text{Total}} \odot S_E$

6.4.4 Evaluate New Scenarios

The previous section demonstrated the possibilities of analyzing the system graph with Kronecker Algebra operations. As the last step, the systems designer needs to evaluate the effects of the identified paths. For example, another new path that has been found in $\mathcal{S}_{\text{Total}}$ is depicted in Figure 6.13. Transforming this path into an MSC, as shown in Figure 6.14, allows the designer to assess the possibility of unexpected behaviour. At first sight, it becomes clear that the ready message c of LM arrives before Control sends message b to RM. If the programmer ensured that the standard interrupt service in Control starts before sending message a, that would not cause any issues. If, however, the service is started after sending b, the service will miss c, and the agent control will never reach state $LCKD$. As for now, the assessment of each path remains a manual task performed by the systems designer.

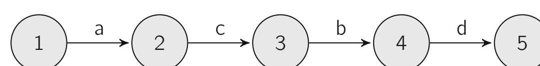


Figure 6.13: The CFG of a new path in $\mathcal{S}_{\text{Total}}$

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

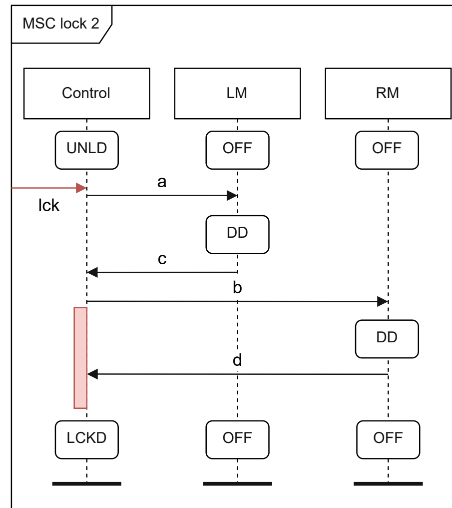


Figure 6.14: The MSC depicting the CFG in Figure 6.13

6.5 Discussion

Within this chapter, we presented a method to identify unintended systems behaviour. Further motivation is that the introduced operations will provide the basis for identifying emergent behaviour in MASs. As for now, the latter could not be confirmed as emergent behaviour strongly depends on the relationship between the micro-and macro-state of the system. Nevertheless, the proposed process and the involved operations allow the identification of implied scenarios which some authors consider as the first step toward identifying emergent systems behaviour [Ash47, DWH04].

The newly introduced Kronecker Algebra Synthesise operation enables transforming MSCs to CFGs while retaining the semantics of the MSC. A particular benefit compared to other approaches is that this model transformation can be automated similarly to all other Kronecker Algebra operations. Therefore, the process might be suitable for system changes or reconfigurations of cyber-physical production systems (CPPSs) at runtime. A limitation, however, is whether a new scenario is desired or harmful, and the resulting root cause search will remain the responsibility of the systems designer. A problem other researchers also point out in their solutions, e.g., Uchitel et al. [UK01, Uch09]. As Moshirpour et al. [MMEF12] reported for their solution, over-generalisation is not present in our process, as we include all intermediate states. A limitation, however, is related to the problem of state explosion.

While we implemented our Kronecker Algebra operation quite efficiently (cf. Section 5.8), there are limits to the number of agents and the involved messages that can be processed. Introducing reduction and filter techniques will be necessary to minimise the state space. Moreover, identifying and removing agents that will not cause new behaviour is a potential solution. Another option is to create subgroups where the boundaries depend on the number of interactions, i.e., agents heavily interacting will be grouped and analysed separately. Each group will be reduced to a new agent interacting with another group. Similar promising research has been conducted by Plateau [Pla85].

A side contribution of this work is that the process would allow a designer to check if an implemented system fulfils the desired scenarios. This type of problem often occurs in model-checking [AY01] and has been approached by researchers such as Kaufmann et al. [KKP+15, KKP+14] and Graaf and Van Deursen [GvD07]. Nevertheless, most of those approaches utilise

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

satisfiability problem of propositional logic (SAT)-solvers, which require previous manual intervention, normalisation, and transformation. This topic we will partly follow up on in the next chapter.

In summary, the proposed process provides the means to identify implied scenarios, which can exist if the combination of several system scenarios (i.e., agent interactions depicted in MSCs) creates sequences of messages (paths μ) that connect some global state $\hat{s}_i$ and the global initial state $\hat{s}_0$ not foreseen in the system scenarios. Therefore, our proposed method solves the problem formulated in Section 6.2.1. Each involved step is exemplified in Section 6.4, based on established tools and algebraic operations.

6.6 Related Work

A large body of knowledge is available for detecting unexpected system behaviour (e.g., faults, implied scenarios, or emergent behaviour). Most relevant to the idea presented in this section are approaches that utilise scenario synthesis methods, including components and interaction messages [Boo94, Mau97] and focus on the early system design phase.

The authors Harel and Kugler [HK02] and Kruger [KGSB99] synthesise state machines from the scenarios representing each agent's behaviour. Moshirpour et al. [MMBH10] show that emergent behaviours can be identified by merging different state machines of a single component. Moreover, they further mention that not all equal states found in the state machines may lead to emergent behaviour. The authors propose first filtering and then selecting only a relevant subset of states for further investigation. Other approaches like the ones from Uchitel et al. [UK01, Uch09] do not consider the message content while synthesising the behaviour model. In the paper of Alur and Yannakakis [AY01], a verification process is presented that connects MSCs using message sequence graphs (MSGs). Nevertheless, it poses the risk of unclear interpretations as the order of connecting the MSCs is undefined.

The authors Puneet et al. [BGMK07, MKS00] propose a technique for detecting implied scenarios using message sequence graphs and FSMs, while the authors' Song et al. [SJHB11, SJB09] use UML and MSGs instead. The results indicate that these methods can effectively find implied scenarios by comparing proposed graphs. However, the designer still must decide if an identified system behaviour is unintended.

Research in model-checking is a large field and is concerned indirectly with finding unintended behaviour [AY01]. By limiting the focus on consistency checking, several proposals are built upon languages such as Promela and the famous model checker Spin [BEG+12a, IMP01, KW07, PIM09, SKM01].

6.7 Concluding Remarks

In the context of the research aim, the presented method is part of identifying basic emergence in distributed systems. Kronecker Synthesise operation can be used in other contexts, as we will demonstrate in the next Chapter. Moreover, while developing the process, the question appeared on verifying if a MSC and a FSM are consistent. Despite a topic naturally located in model checking, solving such a problem is essential.

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Chapter 7

Consistency Checking

This chapter continues the goal of the dissertation and solves a significant problem discovered in the last chapter. Namely, how can it be checked whether an MSC and an FSM are consistent? This conformance is an essential criterion for identifying implied scenarios, since in some cases, after scenarios have been laid out in MSCs, the implemented FSMs may no longer conform to the scenarios, therefore, may produce unexpected behaviour. Based on this insight, the chapter begins with a motivating example that provides the basis for the problem definition. Introducing Kronecker Symmetric Skip operation offers an opportunity to solve the problem. As the last step, we apply our solution to simple examples, evaluate them and discuss them in comparison to related works.

7.1 A Motivating Example

This section provides a motivating example to lay the foundation for the following sections[1]. Moreover, the example will be reused in Section 7.4 to demonstrate the proposed approach.

Figure 7.1 depicts three state machines that describe the behaviours of a PhD student, a coffee machine, and a maintenance unit for the coffee machine. For better readability, we used the UML notation. Later in the paper, an FSM notation will represent the state machines.

[1] The model has been introduced by Kaufmann et al. [KKP+14] and reused in agreement to allow better comparability between the two approaches. We only had to make a few minor changes.

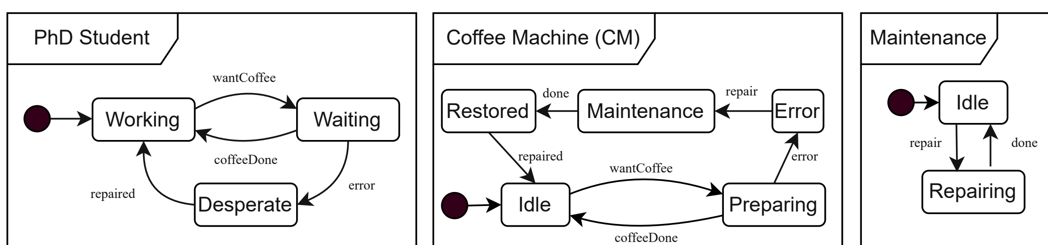


Figure 7.1: The state machines modelling the PhD Student (PhD), a coffee machine (CM), and a maintenance unit (m) of the coffee example.

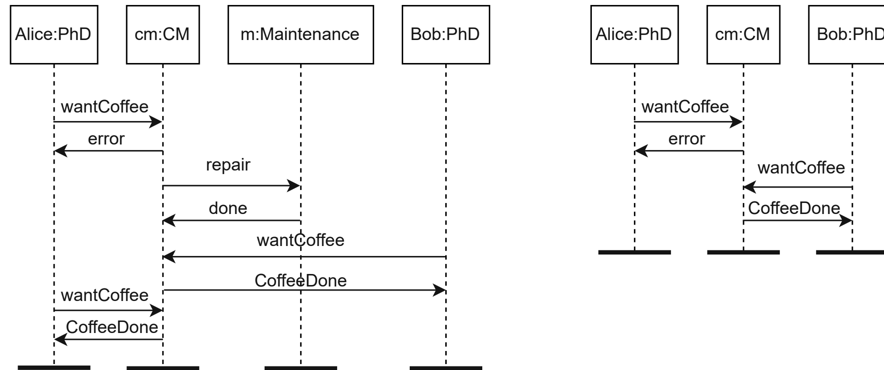


Figure 7.2: Two message sequence charts (MSCs) depicting a desired scenario on the left and a forbidden scenario on the right.

The rounded rectangles present states which are connected by transitions (arrows). Each arrow has a label representing an event that causes a state change. In UML, the initial state is indicated by an incoming arrow/arc from a black dot. The state machines communicate with each other through message passing. A state change occurs according to a received or sent message. It is currently irrelevant whether the event is created externally or by another state. The state machines ignore events if they are irrelevant to the current state, i.e., the transition does not accept the event.

Figure 7.2 shows a set of sequence diagrams, or MSCs, containing communication scenarios between instances of the described state machines. As a reminder, in an MSC, a state machine is instantiated by a lifeline. A lifeline starts with a rectangular box with a dashed vertical line underneath it. Each line has a name inside the box, the name after the ":" indicates the original state machine. In the example, we have shortened these names to save space. Along each lifeline, several message exchanges happen. Each message corresponds to an arrow from the sender lifeline to the receiver lifeline. The labels of the arrows represent the symbol being transferred.

7.1.1 Consistency Between an MSC and State Machines

For an MSC to be consistent with the state machines, the sequence of messages in the diagram must be executable in some global state of the lifelines. A global state is a tuple of the state machine states instantiated by the lifelines. In other words, in the global state, it must be possible for each message to be an event in the sending lifeline and on the receiving lifeline.

Therefore, checking state machines and a sequence diagram for consistency can have two settings: (1) The MSC depicts the desired scenario. If the MSC is consistent with the state machines, then there is certainty that the state machines can execute the scenario. In case there is no consistency, the traces and the global state will indicate where the execution fails and helps, therefore, to identify erroneous or missing transitions. (2) An unwanted scenario is depicted. If the MSC is consistent with the state machines, the state machines are wrongly implemented. Nevertheless, the traces will provide a counter-example, namely a sequence of message labels and states that will support discovering the problem.

TU Bibliothek Your knowledge hub WIEN
Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

7.1.2 A Desired and a Negative Scenario

Returning to Figure 7.2, the two depicted MSCs show each setting. The MSC on the left shows the desired scenario, while the MSC on the right is a negative scenario. Nevertheless, the desired scenario is inconsistent. The PhD student instance "Alice" changes into the state "desperate" after receiving an "error" from the coffee machine. This state can only be changed when "repaired" is received. Looking at the MSC, however, the coffee machine never sends this message. This means the coffee machine never returns to the state "idle" and would never receive the "wantCoffee" from the second PhD student instantiation. Following the message trace of the MSC, it becomes apparent that it can only be executed until and including the fourth message, "done", from "m:Maintenance" to "cm:CM." How the presented approach helps to identify the issue will be shown later in Section 7.4. Fixing this scenario requires removing the state "desperate" from the PhD student and connecting the "state waiting" with the error message directly to the state "working." In the adverse scenario, the coffee machine can prepare coffee while in an error state. This situation is not implemented in the state machine and cannot be executed. The resulting message trace would stop after the first error.

7.2 The Consistency Problem

The problem in the motivating example is a Multiview Sequence Consistency Problem [KKP+14]. The MSCP raises the question of whether the communication sequence modelled in the MSC is executable by the given FSMs. If this is the case, the two diagrams or views are consistent. In other words, a desired positive scenario in a sequence diagram will be consistent with the state machine view, i.e., the state machines can execute the scenario. For the negative scenario in a sequence diagram, the outcome should be inconsistent with the state machine view. That means the state machines cannot execute the desired trace.

The common elements in state machines, sequence diagrams, and their interactions are the symbols in the alphabets $\sum_A$ and $\sum_L$. In $\sum_A$ are all message labels of the sequence diagrams executed by the state machines. The alphabet $\sum_L$ represents the names of the instantiated state machines and the lifelines in the MSCs.

7.2.1 Defining the Problem

Based on the definitions for MSCs (cf. Section 6.2), FSMs (cf. Section 5.2), we can formulate the problem introduced above Multiview Sequence Consistency more precisely. The relevant question is if there is a path between some global state $\hat{s}_i$ and the global initial state $\hat{s}_0$, representing the sequence of messages described in the MSC. Therefore we define the Multiview Sequence Consistency Problem as:

An MSC and a set of state machines are consistent if there is a path, starting from a global state $\hat{s} = (\iota_1, \ldots, \iota_l)$ between two global states $\hat{s}'$ and $\hat{s}''$, by applying the sequence of messages μ. Given a MSC $= (\mathcal{L}, \mu)$, with $\mathcal{L}\{L_1, \ldots, L_l\}$ and $sm(L_i) = (S_i, \iota_i, A_i, T_i)$ for $1 \leq i \leq l$ and the alphabets $\sum_A$ and $\sum_L$.

7.2.2 Limitations

Readers familiar with UML or other modelling languages might recognise that the given definitions are limited. Advanced control flow structures such as hierarchical or history states in state machines

or "alt," "loops," fragments in sequence charts, or exception handling are out of scope. The reason is that some structures create indeterministic behaviour. Therefore, the scope of this section is limited to control flow structures that simple FSMs can represent. In the case of MSCs, only the message sequencing (i.e., the order of message sending/receiving on a lifeline) is of concern but can be adjusted as needed.

7.3 Confirming Consistency

It is obvious that Kronecker operations introduced so far are insufficient to solve the consistency problem. An MSC must be consistent with a *"set of state machines"*. In other words, the MSC must be consistent with the overall behaviour of a system composed of multiple CFGs/FSMs. Kronecker Skip operation is a candidate for modelling the overall behaviour of multiple CFGs/FSMs. However, the operation does not consider additional edge labels of different CFGs/FSMs. We, therefore, introduce Kronecker Symmetric Skip operation in the following.

7.3.1 Kronecker Symmetric Skip Operation

Let $\mathcal{L}_A$ and $\mathcal{L}_B$ be the set of all operations in programs A and B, and $\mathcal{L} = \mathcal{L}_A \cup \mathcal{L}_B$ all operations combined. Moreover, $\mathcal{S}$ denotes a set of common synchronising operations, i.e., $\mathcal{S} = \mathcal{L}_A \cap \mathcal{L}_B$. Similarly, let $\mathcal{V}_A$ and $\mathcal{V}_B$ be additional statements used in the programs A and B, i.e., $\mathcal{V}_A \subseteq \mathcal{L}_A \backslash \mathcal{L}_B$ and $\mathcal{V}_B = \mathcal{L}_B \backslash \mathcal{L}_A$.

Then let $\mathcal{M}_n(\mathcal{L}_A)$ be the set of all matrices of size n with entries $\in \mathcal{L}_A$ and $\mathcal{M}_m(\mathcal{L}_B)$ be the set of all matrices of size m with entries $\in \mathcal{L}_B$. Let $A \in \mathcal{M}_n(\mathcal{L}_A)$ and $B \in \mathcal{M}_m(\mathcal{L}_B)$, followed by $A_S \in \mathcal{M}_n(\mathcal{S})$, $B_S \in \mathcal{M}_m(\mathcal{S})$, $A_{V_A} \in \mathcal{M}_n(\mathcal{V}_A)$ and $B_{V_B} \in \mathcal{M}_m(\mathcal{V}_B)$, such that $A = A_{V_A} + A_S$ and $B = B_{V_B} + B_S$. In addition, the identity matrix of size n is again denoted by $I_n \in \mathcal{M}_n$, and the zero matrix of size n by $Z_n \in \mathcal{M}_n$. Similarly, $I_m \in \mathcal{M}_m$ and $Z_m \in \mathcal{M}_m$. Applying the same idea as in Kronecker Skip by adding self-loops on both sides, results in the following equation:

$$A \mathbin{\bar{\odot}} B = \left(A + \left(\sum_{x \in V_B} x\right) \cdot I_n\right) \otimes \left(B + \left(\sum_{y \in V_A} y\right) \cdot I_m\right) \tag{7.1}$$

This can be simplified as follows

$$= \left(A_{V_A} + A_s + \left(\sum_{x \in V_B} x\right) \cdot I_n\right) \otimes \underbrace{\left(B_{V_B} + B_S + \left(\sum_{y \in V_A} y\right) \cdot I_m\right)}_{\xi} \tag{7.2}$$

$$= A_{V_A} \otimes \xi + \underbrace{A_s \otimes \xi}_{\nu} + \underbrace{\left(\sum_{x \in V_B} x\right) \cdot I_n \otimes \xi}_{\rho} \tag{7.3}$$

$$= \underbrace{A_{V_A} \otimes B_{V_B}}_{Z_{m \cdot n}} + \underbrace{A_{V_A} \otimes B_S}_{Z_{m \cdot n}} + \underbrace{A_{V_A} \otimes \left(\sum_{y \in V_A} y\right) \cdot I_m}_{A_{V_A} \otimes I_m} + \nu + \rho \tag{7.4}$$

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

$$= A_{V_A} \otimes I_m + \underbrace{A_S \otimes B_{V_B}}_{Z_{m \cdot n}} + \underbrace{A_S \otimes B_S + A_S \otimes \left(\sum_{y \in V_A} y\right) \cdot I_m}_{Z_{m \cdot n}} + \rho \quad (7.5)$$

$$= A_{V_A} \otimes I_m + A_S \otimes B_S + \rho \quad (7.6)$$

$$= A_{V_A} \otimes I_m + A_S \otimes B_S + \left(\sum_{x \in V_B} x\right) \cdot I_n \otimes \xi \quad (7.7)$$

$$= A_{V_A} \otimes I_m + A_S \otimes B_S + \left(\sum_{x \in V_B} x\right) \cdot I_n \otimes B_{V_B} + \underbrace{\left(\sum_{x \in V_B} x\right) \cdot I_n \otimes B_S}_{Z_{m \cdot n}} + \underbrace{\left(\sum_{x \in V_B} x\right) \cdot I_n \otimes \left(\sum_{y \in V_A} y\right) \cdot I_m}_{Z_{m \cdot n}} \quad (7.8)$$

hence Kronecker Symmetric Skip operation $\bar{\odot}$ can be given as:

$$A \mathbin{\bar{\odot}} B = A_{V_A} \otimes I_m + A_S \otimes B_S + I_n \otimes B_{V_B}. \quad (7.9)$$

Compared to Kronecker Skip operation, the Symmetric Skip ($\bar{\odot}$) operation combines two FSMs into one, in such a way that all synchronisation operations are respected. Therefore, the symmetric skip operation can be used to build the overall behaviour of a system consisting of several CFGs/FSMs. An example follows in Section 7.4, where running $\bar{\odot}$ iteratively over several FSMs leads to an FSM, capable of representing the global states $\hat{s}$ defined in Section 6.2.

7.3.2 Reusing Kronecker Synthesize Operation

A remaining issue to solve the consistency problem is comparing an MSC with the entire system's behaviour. The problem statement mentions, *"if there is a path, starting from a global state $\hat{s} = (\iota_1, \ldots, \iota_l)$ between two global states $\hat{s}'$ and $\hat{s}''$, by applying the sequence of messages μ."* In other words, a set of instructions μ that lead from global state $\hat{s}$ to a new state $\hat{s}'$. Therefore, we again face the question how to transform an MSC into a CFG, reflecting the underlying semantics of the MSC and the messages in μ. However, this time we already know the answer: our Kronecker Synthesize operation introduced in Section 6.3.2, that creates R_i.

7.3.3 Bringing it All Together

Now all parts are available to solve the Multiview Sequence Consistency Problem defined in Section 7.2.1. A scenario given by an MSC can be transformed with Kronecker Synthesize operation into an FSM (R_i). The resulting FSM represents the path μ in the MSC that creates the desired global state change from $\hat{s}'$ and $\hat{s}''$. For the instantiated state machines, Kronecker Symmetric Skip operation $\bar{\odot}$, creates an FSM ($\mathcal{S}_{\text{Total}}$), representing the entire system's global behaviour. At last Kronecker Skip operation $\odot$, ensures that the MSC complies with $\mathcal{S}_{\text{Total}}$. If the resulting graph R_e is isomorphic to R_i, it is possible to execute the given scenario. Figure 7.3 depicts our approach.

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

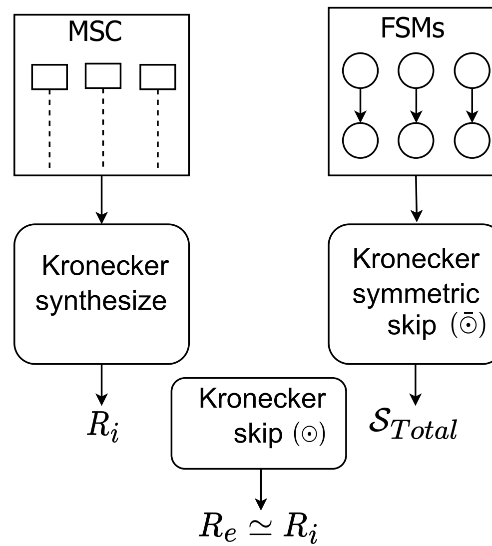


Figure 7.3: Interplay of the different Kronecker operations

7.4 Evaluation

This section demonstrates and evaluates the previously presented Kronecker operations applied to the motivating example in Section 7.1.

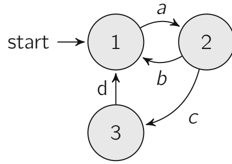
7.4.1 Preparation of the FSM

As a first step, the given state machines are transformed into deterministic FSMs. Plain numbers replace the state names, and for readability, the edge labels are replaced with short letters, as follows:

- a = wantCoffee
- b = coffeeDone
- c = error
- d = repaired
- e = repair
- f = done

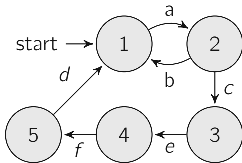
Figures 7.4 – 7.6 show the three transformed state machines with their matrix representation. All state machines do not have a final node because they have to execute continuously.

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub



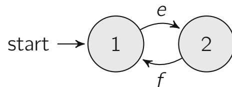
$$PhD = \begin{pmatrix} \cdot & a & \cdot \\ b & \cdot & c \\ d & \cdot & \cdot \end{pmatrix} \tag{7.10}$$

Figure 7.4: The FSM and matrix of the PhD Student (PhD)



$$CM = \begin{pmatrix} \cdot & a & \cdot & \cdot & \cdot \\ b & \cdot & c & \cdot & \cdot \\ \cdot & \cdot & \cdot & e & \cdot \\ \cdot & \cdot & \cdot & \cdot & f \\ d & \cdot & \cdot & \cdot & \cdot \end{pmatrix} \tag{7.11}$$

Figure 7.5: The FSM and matrix of the Coffee Machine (CM)



$$m = \begin{pmatrix} \cdot & e \\ f & \cdot \end{pmatrix} \tag{7.12}$$

Figure 7.6: The FSM and matrix of Maintenance (m)

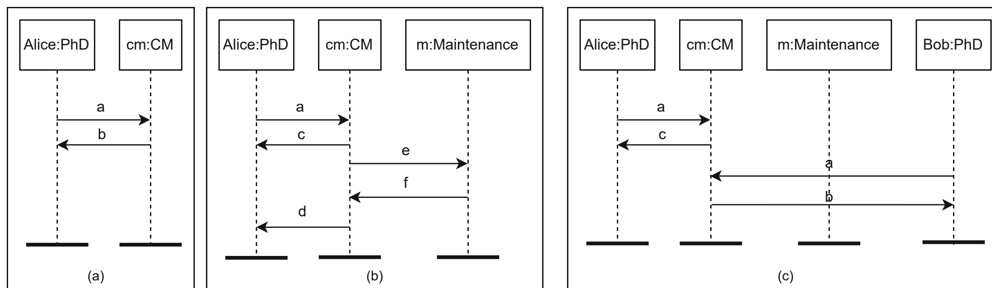


Figure 7.7: Three MSCs for the evaluation

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

7.4.2 A First Simple Example

The first example (a), shown in Figure 7.7, is about a PhD student who wants a coffee and gets it. Therefore the required FSMs are PhD and CM, and as a first step, Kronecker Symmetric Skip $\bar{\odot}$ is applied to get $\mathcal{S}_{\text{Total}}$. Eq. (7.13) shows the elements needed, while Eq. (7.14) and Eq. (7.15) show the matrix representations. I_n and I_m are the identity matrices.

Calculating $PhD_{V_{PhD}} \otimes I_m$ results in Z_n since $PhD_{V_{PhD}}$ has no entries. $PhD_S \otimes CM_S$ is displayed in Eq. (7.16), and $I_n \otimes CM_{V_{CM}}$ in Eq. (7.17). Since the matrices reach a size of 15x15, only the indices of the entries are given. $\mathcal{S}_{\text{Total}}$ is the addition of $(Z_n + T_1) + T_2$.

Creating a FSM from $\mathcal{S}_{\text{Total}}$ shows that it is isomorphic to CM. Therefore, $\mathcal{S}_{\text{Total}} \simeq CM$ applies, and Figure 7.5 equally represents $\mathcal{S}_{\text{Total}}$.

$$PhD \mathbin{\bar{\odot}} CM = PhD_{V_{PhD}} \otimes I_m + PhD_S \otimes CM_S + I_n \otimes CM_{V_{CM}} \tag{7.13}$$

$$PhD_{V_{PhD}} + PhD_S = \begin{pmatrix} \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \end{pmatrix} + \begin{pmatrix} \cdot & a & \cdot \\ b & \cdot & c \\ d & \cdot & \cdot \end{pmatrix} \tag{7.14}$$

$$CM_{V_{CM}} + CM_S = \begin{pmatrix} \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & e & \cdot \\ \cdot & \cdot & \cdot & \cdot & f \\ \cdot & \cdot & \cdot & \cdot & \cdot \end{pmatrix} + \begin{pmatrix} \cdot & a & \cdot & \cdot & \cdot \\ b & \cdot & c & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot \\ d & \cdot & \cdot & \cdot & \cdot \end{pmatrix} \tag{7.15}$$

$$PhD_S \otimes CM_S = \begin{pmatrix} \cdot & \cdots & a_{1,8} & \cdots & \cdot \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ b_{8,1} & \cdots & \cdots & c_{8,13} & \cdot \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ d_{15,1} & \cdots & \cdots & \cdots & \cdot \end{pmatrix} = T_1 \tag{7.16}$$

$$I_n \otimes CM_{V_{CM}} = \begin{pmatrix} \cdot & \cdots & \cdots & \cdots & \cdot \\ \vdots & e_{3,4}; f_{4,5} & \vdots & \vdots & \vdots \\ \cdots & \cdots & e_{8,9}; f_{9,10} & \cdots & \cdot \\ \vdots & \vdots & \vdots & e_{13,14}; f_{14,15} & \vdots \\ \cdot & \cdots & \cdots & \cdots & \cdot \end{pmatrix} = T_2 \tag{7.17}$$

The next step is synthesising the MSC in Figure 7.7 (a) into the usage scenario R_i. However, since there are only two messages a and b, and their dependency is a before b, the resulting CFG R_i can only have three states and two labels in the specified order. Figure 7.8 on the left shows the CFG of usage scenario R_i. The final step is to apply Kronecker skip $\odot$ in a similar way to Kronecker symmetric skip $\bar{\odot}$, i.e. $\mathcal{S}_{\text{Total}} \odot R_i = \mathcal{S}_{\text{Total}_V} \otimes I_m + \mathcal{S}_{\text{Total}_S} \otimes R_i$. The resulting FSM is shown in Figure 7.8 on the right and shows that R_i can be executed in $\mathcal{S}_{\text{Total}}$.

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Figure 7.8: Representation of the CFG R_i (left) and the resulting FMS of $\mathcal{S}_{\text{Total}} \odot R_i$ (right) for MSC (a) in Figure 7.7.

7.4.3 Focusing on the MSC Synthesis

The second example (b) in Figure 7.7 contains the interaction between PhD, CM, and m. As beforehand, the first step is to find $\mathcal{S}_{\text{Total}}$. However, $PhD \mathbin{\bar{\odot}} CM$ is already done, and adding m would mean calculating the $\bar{\odot}$ of the previous $\mathcal{S}_{\text{Total}}$ and m. The new $\mathcal{S}_{\text{Total}}$ is again $\simeq CM$.

Synthesising the MSC in Figure 7.7 (a) into the usage scenario R_i, requires creating $R_0 = S_1 \oplus S_2 \oplus \ldots \oplus S_n$. The first three message label representations are shown in Eq. (7.18) – (7.20) together with the dependencies.

$$R_0 = \begin{pmatrix} \cdot & a \\ \cdot & \cdot \end{pmatrix} \oplus \begin{pmatrix} \cdot & b \\ \cdot & \cdot \end{pmatrix} \oplus \begin{pmatrix} \cdot & c \\ \cdot & \cdot \end{pmatrix} \oplus \ldots \tag{7.18}$$

$$D_1 = \begin{pmatrix} \cdot & a & \cdot \\ \cdot & \cdot & c \\ \cdot & \cdot & \cdot \end{pmatrix}, \quad D_2 = \begin{pmatrix} \cdot & c & \cdot \\ \cdot & \cdot & e \\ \cdot & \cdot & \cdot \end{pmatrix}, \tag{7.19}$$

$$D_3 = \begin{pmatrix} \cdot & e & \cdot \\ \cdot & \cdot & f \\ \cdot & \cdot & \cdot \end{pmatrix}, \quad D_4 = \begin{pmatrix} \cdot & f & \cdot \\ \cdot & \cdot & d \\ \cdot & \cdot & \cdot \end{pmatrix} \tag{7.20}$$

To get R_i, in this case, $i = 4$, since there are four dependencies, Eq. (6.8) must be applied iteratively i times. The operations cannot be displayed since the matrices reach a size of 2592×2592, but the resulting CFG R_i is shown in Figure 7.9. After calculating $\mathcal{S}_{\text{Total}} \odot R_4 = \mathcal{S}_{\text{Total}_\mathcal{V}} \otimes I_m + \mathcal{S}_{\text{Total}_\mathcal{S}} \otimes R_4$, the resulting graph is $\simeq CM$, which shows that R_i can be run in $\mathcal{S}_{\text{Total}}$.

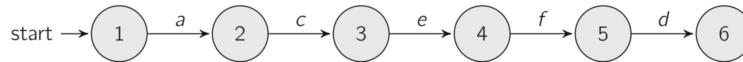


Figure 7.9: The CFG R_i of MSC (b) in Figure 7.7.

7.4.4 The Error Scenario

The last example (c) in Figure 7.7 contains two $PhDs$, one CM and one m. This scenario should not work in $\mathcal{S}_{\text{Total}}$. Compared to the previous examples, $\mathcal{S}_{\text{Total}}$ is not $\simeq CM$ because there is another PhD instance. For visualisation purposes, the labels have indices to indicate their affiliation. These indices are not necessary for the implementation. Figure 7.10 shows $\mathcal{S}_{\text{Total}}$ resulting from Kronecker Symmetric Skip operation. While Figure 7.11 shows the resulting CFG R_i after the synthesis of the MSC in Figure 7.7 (c).

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

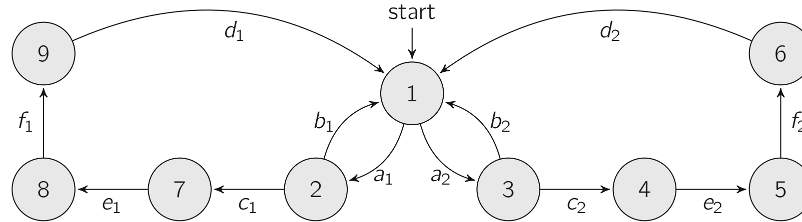


Figure 7.10: $\mathcal{S}_{\text{Total}}$ for MSC (c) in Figure 7.7.

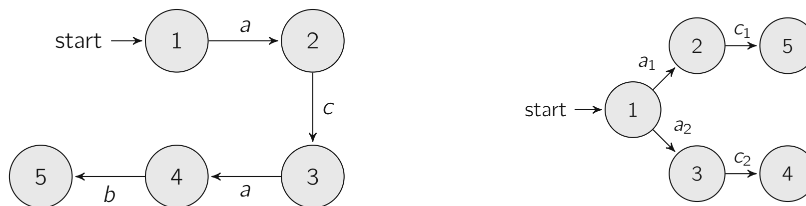


Figure 7.11: Representation of the CFG R_i (left) and the resulting FMS of $\mathcal{S}_{\text{Total}} \odot R_i$ (right) for MSC (c) in Figure 7.7.

To check if R_i is part of $\mathcal{S}_{\text{Total}}$ we calculate, $\mathcal{S}_{\text{Total}} \odot R_i = \mathcal{S}_{\text{Total}_\mathcal{V}} \otimes I_m + \mathcal{S}_{\text{Total}_\mathcal{S}} \otimes R_i$. The result Figure 7.11 shows the resulting FSM on the right. In this case R_i cannot be executed in $\mathcal{S}_{\text{Total}}$. The system either takes the path from PhD_1 and hangs after outputting c or does the same with PhD_2. A programmer can quickly determine whether the system is implemented incorrectly or the scenario is forbidden and should not be run.

7.5 Discussion

This chapter continues the goal of the dissertation and solves a significant problem discovered in the last chapter, namely, ensuring consistency between MSCs and FSMs. This conformance is an essential criterion for identifying implied scenarios since, in some cases, after scenarios have been laid out in MSCs, the implemented FSMs may no longer conform to the scenarios, therefore, may produce unexpected behaviour. The presented approach solves this problem using three Kronecker operations to provide a consistency check.

The novel Kronecker Symmetric Skip operation $\bar{\odot}$ allows combining multiple state machines into one FSM that represents the entire system behaviour. Kronecker Synthesize from the previous chapter creates a CFG based on a given MSC and allows the underlying semantics to be adopted. In the last step, Kronecker Skip $\odot$ compares the previous results for consistency. The resulting graph either confirms consistency or provides valuable insight to a programmer when tracing where the problems are occurring. We have shown the applicability of each of the above steps in the example provided at the beginning of the chapter. Compared to the work of Graaf and Van Deursen [GvD07], Kronecker approach requires no manual intervention and can be automated.

A current limitation is that only simple FSMs and MSCs are supported without advanced modelling capabilities that exist in UML. For example, constructs such as hierarchical or history states in state machines or "alt" and "loop" fragments in MSCs require further research. However, Kronecker Algebra has been used to handle alternatives and loops in worst-case execution time analysis [MB21]. Therefore, we expect these constructs to be integrated smoothly into our approach.

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

7.6 Related Work

As indicated beforehand, the problem approached in this paper is a model-checking issue. Therefore, many proposals are available that build upon languages such as Promela and the famous model checker Spin [BEG+12a, KW07, PIM09]. Nevertheless, due to semantic differences between state machines and Promela, an equivalence-preserving translation of all language elements seems to be the main challenge. The authors in Brosch et al. [BEG+12b, BEG+12a] point out that their solution based on Spin can ensure that given traces do not occur during the execution of a set of state machines but cannot guarantee that a given message sequence is possible. To solve that problem, the same research group exchanged Spin with SAT solvers [KKP+15, KKP+14]. In Matsumoto et al. [MYA+19], the use of the FDR model checker shows similar issues and cannot be applied further to analyse the difference between sequence diagrams. The ideas in Kaufmann et al. [KKP+15] have inspired Küster and Caminati [KC20] to combine the Isabelle/HOL theorem prover with an SMT solver.

Other authors propose formal approaches. Nevertheless, most implementations are either unavailable, have become obsolete, or have not been maintained. A few approaches are listed in the following, but for a more detailed overview, the interested reader is guided to specific surveys like [LMT09]. An example of an algebraic approach based on π-calculus comes from Lam and Vitus [LP05]. Unfortunately, there is no discussion on realisability. The potential use of description logic to formally describe the consistency between sequence diagrams, class diagrams, and state machines is described in Van der Straeten et al. [VDSMSJ03]. Description logic is more expressive than SAT, but their satisfiability checking problem also involves higher complexity classes than NP.

In Bernardi et al. [BDM02], Petri nets are used to check the consistency between diagrams. Engels et al. [EHHS02] decide on consistency based on constraints represented in the form of collaborations and provide an interpreter. Similarly, Egyed [Egy06] uses rules formulated in OCL for instant consistency validation. The author shows that the approach is very efficient on large models. In Feng and Vangheluwe [FV03], a simulation-based approach is presented.

Other research does not solely focus on consistency checking between state machines and sequence diagrams. The focus is on consistency checking between class, collaboration, and activity diagrams. For example, in Stephan and Cordy [SC13], the authors compare two sequence diagrams to detect the longest standard message passings [LMZS06]. In Odamura et al. [OOO20], their method focuses on identifying differences in PlantUML sequence diagrams. In Triandini et al. [TFSR19], vectors are compared that correspond to an MSC. The vectors hold the MSC objects and messages. Other work that looks specifically into UML diagram comparisons can be found in Matsumoto et al. [MYA+19, MYA+20], while some surveys covered that topic more broadly [GBS13, UNKC08].

The approach closest to our approach, next to Kaufmann et al. [KKP+15, KKP+14], is proposed by Graaf and Van Deursen [GvD07]. In their work, the authors synthesise a state machine from the given sequence diagram as previously done in [WS00] and compare the generated state machine to the given state machine. In essence, the approach includes normalisation, transformation, and comparison. However, the comparison requires manual intervention.

7.7 Concluding Remarks

In the context of the research goal, this chapter's procedure extends the previous chapter's method. However, it is not a sole extension as it solves a real-world problem and is applicable as a stand-alone method. In particular, the newly introduced Kronecker Symmetric Skip operation $\bar{\odot}$, contributes to ongoing research based on Kronecker Algebra.

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

Chapter 8

Priority

This chapter examines another possible cause of emergent behaviour, namely priorities. Priorities are a common element in operating systems to schedule processes according to their importance. First, a short introduction is given, followed by a prominent example that can occur in prioritised processes. Second, we show how we adapt our successor search algorithm to incorporate priorities into our Kronecker algebra, followed by more corner cases. Finding priority inversion in the presented example with Kronecker serves as an evaluation case, and we discuss the results compared to related work. In addition, this chapter forms the basis for the next chapter, which deals with execution times.

8.1 Process Prioritisation

Process prioritisation is a concept present in every current operating system. The operating system uses process priority to decide how much time each process is allowed on the CPU. While not going into too much detail here, processes are assigned different categories and priorities, values that enable process scheduling. For example, the priority of a process could depend on time limits and memory requirements, or it could be statically or dynamically set. Based on the priorities, the OS scheduling algorithm takes care of tracking processes and their de- and assignment on the CPU.[1] Naturally, higher-priority execute before lower-priority processes.

Scheduling predominantly follows one of two strategies (algorithms); Non-Preemptive and Preemptive [LS86]. With non-preemptive scheduling, a currently running process is not disturbed even though a process with a higher priority arrives. However, the scheduler schedules the newly arrived process to run versus other processes waiting in the queue. In the case of a high-priority process, once the current process is finished, it is allowed to run on the CPU. Preemptive scheduling, on the other hand, stops and saves the currently running process when a high-priority process arrives. After the high-priority process finishes, the previous one resumes execution. Priority scheduling has disadvantages; For example, lower priority processes may never get access to the CPU and starve, or other more advanced issues like priority inversion.

[1] Scheduling and priorities in network traffic is another large research area, but is at the moment out of scope.

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

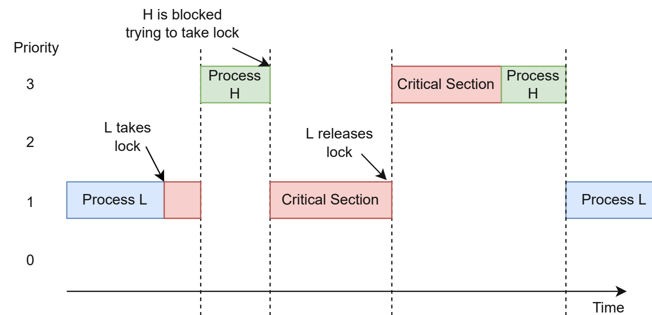


Figure 8.1: An example of a bounded priority inversion with a lower-priority process executing before (or blocking) a high-priority process.

8.2 Bounded/Unbounded Priority Inversion

Priority inversion is a system error that can occur when a lower-priority process indirectly blocks a higher-priority process [LSR+88]. A typical example is that the lower priority process holds a mutex that the higher priority process needs to continue execution and therefore has to wait. Figure 8.1 shows a simple case where, the high-priority process H is blocked while the low-priority process (process L) holds the lock. This particular situation is known as "bounded priority inversion" because the duration of the inversion depends solely on how long the lower priority process is in the critical section or, in this case holding the lock. The critical section in Figure 8.1 represents the time duration process H is blocked until process L releases the lock. Therefore, the priority of the processes is "inverted" since process L now executes before process H [LSR+88].

Another type of priority inversion is called unbounded priority inversion [LSR+88]. In this case, a medium-priority process (process M) interrupts process L, which is holding the lock. Compared to the previous example, process M blocks H for an "unbounded" time by preempting L. The reason is that L needs to wait for M to finish before releasing the lock for H. Another issue is that the scheduler prioritises any other process with a higher priority than L. Therefore, the time until H receives the lock is nearly impossible to determine. In other words, the priority inversion stays for an indefinite time. Figure 8.2 visualises the interaction between the different processes.

A prominent and widely published example is how a priority inversion bug nearly ended the Mars Pathfinder mission in 1997.[2] After deploying the rover, the lander would randomly reset every few days due to an intermittent priority inversion bug that caused the watchdog timer to trigger a complete system restart.[3] The team at NASA could fix the bug via a remote update after a long and challenging search for the cause of the reset. Nevertheless, there are solutions to prevent unbounded priority inversion. Two more frequently cited methods are priority ceiling protocol and priority inheritance [SRL90].

[2]Not to be mixed up with the Mars rover incident, that landed in a crevice due to improper use of Howard Wolowitz.

[3]http://www.cs.cornell.edu/courses/cs614/1999sp/papers/pathfinder.html

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

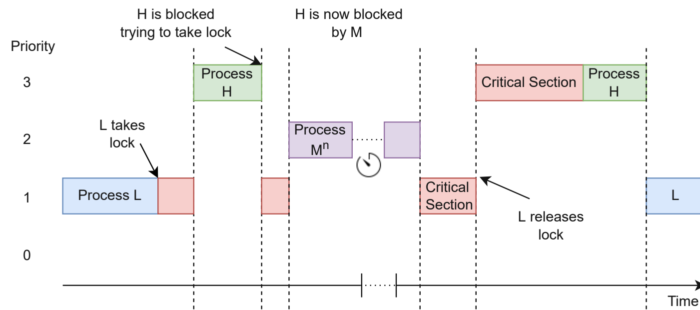


Figure 8.2: An example of an unbounded priority inversion with a mid-priority process blocking a high-priority process for an indefinite time.

8.3 Priority Ceiling and Inheritance Protocol

The priority ceiling protocol introduces a so-called "priority ceiling level" to each lock or shared resource [GS88]. If a process acquires a resource/lock, its priority is automatically increased to the priority ceiling level linked to the specific resource/look. The ceiling priority depends on the highest process priority that accesses the resource or lock. As shown in Figure 8.3, the highest priority is 3 (process H); when process L takes the lock, its priority is increased to 3, equal to process H. The priority boost prevents process M from interrupting L and M. Later in this chapter, we use the priority ceiling protocol in combination with our Kronecker Algebra.

Another method, known as "priority inheritance," follows the idea of giving the process holding the lock the same priority as any other process (of higher priority) that tries to get a hold of the lock. As exemplified in Figure 8.4, process L again takes the lock. If process H attempts to take the lock, L will inherit the priority of H. Therefore, unlike the unbounded example, another process cannot interrupt L until L and H finish their critical sections. It is essential to mention that in both methods, the priority of process L is returned to its original value after it releases the lock. Moreover, those methods can only prevent unbounded priority inversion; it is still possible for bounded priority inversion [SRL90].

In general, bounded priority inversion can only be avoided or mitigated by following certain programming practices:

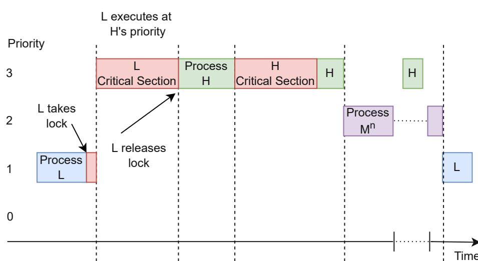


Figure 8.3: Priority ceiling protocol.

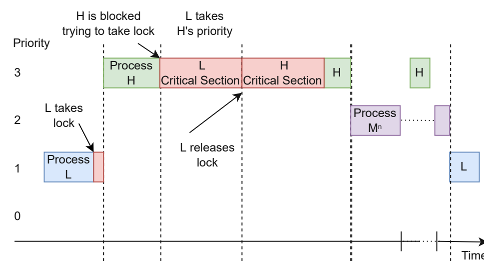


Figure 8.4: Priority inheritance.

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

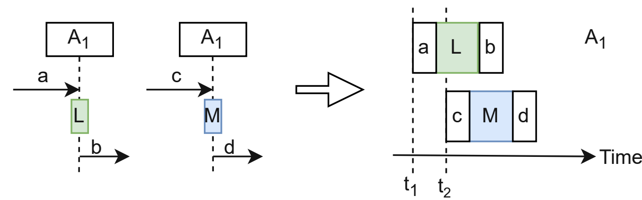


Figure 8.5: The agent A_1 receiving and sending two different message pairs.

- The simplest solution is to avoid critical sections or locks that can block a high-priority task, e.g., by using wait-free (non-blocking) algorithms.
- If required, the sections should be as short as possible to reduce the blocking time.
- Or the introduction of a control task of the shared resource to avoid the need to create locks to protect it

An example of the last option could be a service task that sends and receives messages from a serial port through queues.

8.4 Problem Definition

Now let us return to agent systems and their message interactions from the previous chapters. Figure 8.5 depicts agent A_1 receiving and sending two different message pairs on the left. Usually, an action is triggered when an agent receives a message, e.g., a calculation, relaying a message or determining to ignore a message. The coloured rectangles in the figure represent such actions. As discussed in Chapter 6, messages can arrive at any time and might have different priorities, or the actions they trigger have different importance. This circumstance is best visible on the right side of Figure 8.5. The arrival times t_1 and t_2 are unknown to the agent, as well as the priorities of the activities.[4] Therefore, the agent needs to decide how to process the messages and if there is not only a first in, first out queue, process prioritisation becomes relevant. In other words, an agent is a processing unit facing the same issues as an OS when scheduling processes that require resources.

In the context of basic emergence, the different possible variations of how an agent reacts to incoming messages and process priorities could be the source of unexpected behaviour. Therefore, the problem definition is to extend Kronecker Algebra to handle edges with priorities while considering the required resources. In addition, it should be possible to identify priority inversion that can occur in multi-threaded applications that share resources.

[4]The processing times of the activities also play a role. Those are the topic of the next chapter.

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

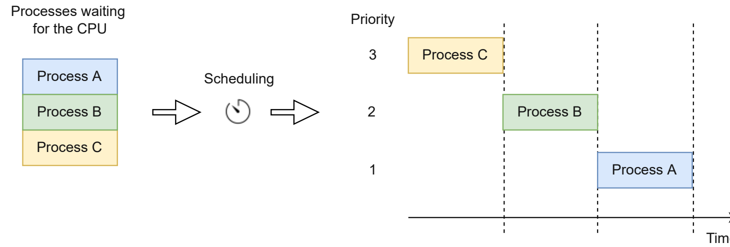


Figure 8.6: Three processes with different priorities scheduled on one CPU

8.5 Kronecker Priority

As a starting point, we assume three processes (A, B, C) running concurrently, that require a CPU, and each has a different priority. If only one CPU is available, the processes execute according to their priority (cf. Figure 8.6); as process C gets the CPU first, the others must wait in line. The relevant parts in this example are the processes P, their priorities p, and the number of shared resources $\#RC$. Currently, processing times are irrelevant, and no external scheduling is involved (i.e., a scheduler that actively changes a process's priority).

Processes like message transactions in Chapter 6 can be represented in Kronecker Algebra as a two by two matrix, i.e., in the form of one directed edge label. For the current example, Eq. (8.1) shows the three matrix representations.

$$P_A = \begin{pmatrix} \cdot & a_{[p:1]} \\ \cdot & \cdot \end{pmatrix}, P_B = \begin{pmatrix} \cdot & b_{[p:2]} \\ \cdot & \cdot \end{pmatrix}, P_C = \begin{pmatrix} \cdot & c_{[p:3]} \\ \cdot & \cdot \end{pmatrix} \tag{8.1}$$

Applying Kronecker Sum to the matrices will result in all possible interleavings between the processes (cf. Eq. (8.2)). However, the resulting P_{Total} does not yet consider any priorities.

$$P_{\text{Total}} = P_A \oplus P_B \oplus P_C \tag{8.2}$$

For this reason, it is necessary to allocate a priority to each label. In the matrices shown in Eq. (8.1), this is indicated by $[p : n]$, for $n \in \mathbb{N}$. The priority will, however, only piggyback while applying Kronecker Sum to the single processes; the solution to our problem lies in identifying a node's successors. As described in Section 5.8, the lazy implementation (cf. Algorithm 1) provides a list S_i of the successors of node i in the resulting matrix. Based on S_i the entries of the resulting matrix are calculated. With the entries of S_i having an allocated priority, it is possible to sort the successors accordingly. Based on this order, it is now possible to allow edges (successors) to continue from node i starting with the highest priority until all resources are utilised.

In a formalised way, we define the following:

- S_i is a list of successor edges of node i ordered by the priorities of $e(i, j)$,
- while $e(i, j)$ is the directed edge from $i \rightarrow j$ and
- $p(i, j)$ the priority of edge $e(i, j)$.
- Moreover $|S_i|$ is the length of the list S_i and

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

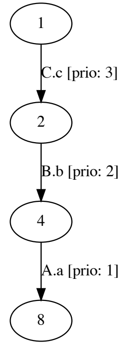


Figure 8.7: P_{Total} with one CPU available.

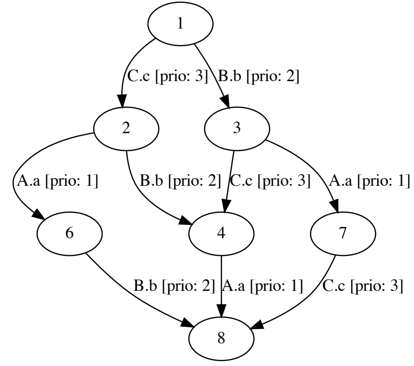


Figure 8.8: P_{Total} with two CPUs available.

- $S_{i,k}$ is the set of the first k edges of S_i.
- $\#RC$ shall represent the number of available resources (e.g., CPU).
- S_N is a set of edges that are the successors of node i.

Nevertheless, it is not straightforward to compute S_N, without considering two border cases. The first one is if more resources are available as required by the processes. In this case, S_N, can be returned directly, as each successor can be executed. The second one comprises that there are not enough resources available, and therefore only a few successors can continue. If there is a clear priority order, only the first $\#RC$ successors continue. However, if there are several equally prioritised successors, those must also be included. The reason is that all the same priority processes have the same probability of getting a resource, and it is impossible to determine which one.

The following equation defines the two cases.

$$S_N = \begin{cases} S_{i,|S_i|}, & \text{if } |S_i| < \#RC, \\ \{e(i,k) \mid e(i,k) \in S_{i,\#RC}\} \cup \{e(i,l) \mid l > \#RC \wedge p(i,l) = p(i,\#RC)\}, & \text{if } |S_i| \geq \#RC \end{cases} \tag{8.3}$$

Now let us return to the three process examples. If only one CPU is available, P_{Total} is reduced to the graph shown in Figure 8.7. The highest priority will take the resource, and releases it for the following priority task to take it.

The graph will change if two CPUs are available, as depicted in Figure 8.8. The successors of node 1 are the edges c and b due to their higher priorities. On the next level (nodes 2 and 3), one might expect only a to execute. However, that is not how Kronecker Sum operates; it shows all possible successors of a node without synchronisation. While this seems counterintuitive, it reflects reality in one path as only the past executions are known. Therefore for each node, all remaining edges are possible successors that can use all available resources.

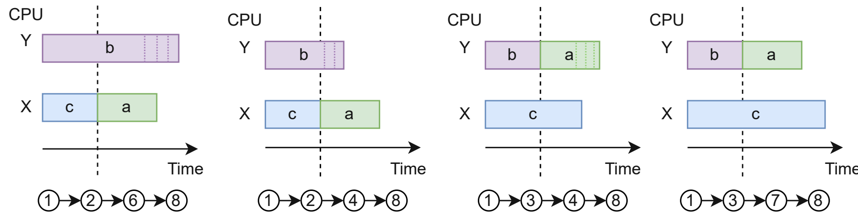


Figure 8.9: Interpretation of the paths in P_{Total} with two CPUs available.

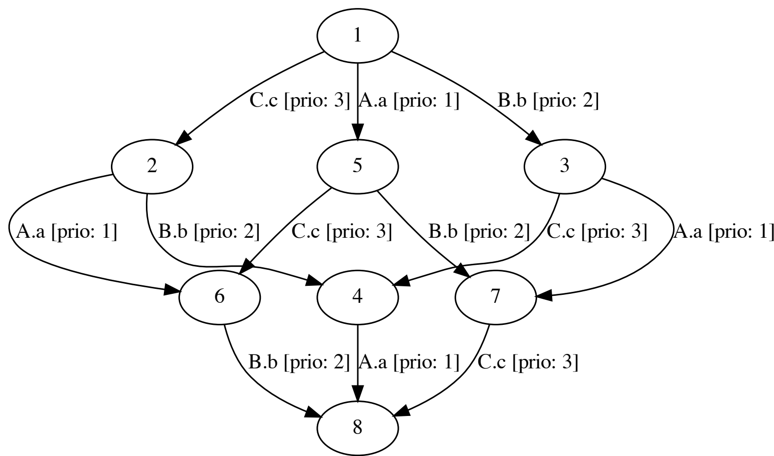


Figure 8.10: P_{Total} with three CPUs available.

Keeping that in mind and that there is no time involved, let us look at one path, e.g., 1→ 2 → 6 → 8. This path yields the labels in order c, a, b. As already mentioned, starting from node 1 label c is on the first CPU and b on the second. Arriving at node 2, the possible successors are a and b. We can interpret this as that b is still running on the second core, and a is now on the first. Node 4 only has one successor (b), again because b is still running on the second core. Figure 8.9 illustrates the above interpretation and the other paths. Some bars have several endings, as it is impossible to determine when those processes end. Only the ones that have a direct successor have finished their execution. Similarly, we can interpret Figure 8.10, which depicts the result with three available CPUs.

8.5.1 Conditionals

After being able to handle processes with different priorities and the number of available resources, other aspects need to be considered. So far, we have only considered a process as a simple path, without any loops or if/else statements. Let us start with an example containing loops. Figure 8.11

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

Figure 8.11: State machines A and B with loops

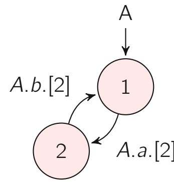


Figure 8.12: Result of $A \oplus B$ without adjustments and one CPU.

depicts two state machines, A and B. State machine A on the left has a loop based on two edges with priority 2, while the last edge has priority 1. The second state machine on the right (B) is identical, but all edges have priority 1. If we compute $A \oplus B$, use the conditions given in Eq. 8.3 and set $\#RC = 1$, the resulting graph looks like the one in Figure 8.12. The reader will agree that state machine B never gets the CPU because the two high-priority labels keep it in the loop. However, that does not reflect reality; at one point, the loop will exit, and either edge c will get the CPU or one of the edges of state machine B.

To incorporate loops, the previous Eq. (8.3) needs some adjustments. The idea is to add to the possible successors all edges with a lower priority that belong to the state machine holding the resource, i.e., the sorted list S_i needs to be searched for other successor edges of the highest priority task belonging to the same process.

For doing so, we have to define additionally:

- $h(i, j)$ which is the name of the process the edge $e(i, j)$ belongs to.

With this addition, we can extend the second option of Eq. (8.4) with all lower-priority edges that belong to the same process that holds the resource $\#RC$.

$$S_N = \begin{cases} S_{i,|S_i|}, & \text{if } |S_i| < \#RC, \\ \{e(i,k) \mid e(i,k) \in S_{i,\#RC}\} \cup \{e(i,l) \mid l > \#RC \wedge p(i,l) = p(i,\#RC)\} & \\ \cup\ \{e(i,q) \mid \exists k (h(i,q) = h(i,k) \in S_{i,\#RC})\}, & \text{if } |S_i| \geq \#RC \end{cases} \tag{8.4}$$

Figure 8.13 shows the result of $A \oplus B$ (with one CPU) after the adjustment. Now, the graph shows an exit of the loop by edge c, which allows B to execute later. However, the result is not complete either. There should be two more edges (d, f) from node 1, as they have the same priority as c. However, we cannot implement this possibility; otherwise, Kronecker Sum would find successors that would not be allowed due to their priorities and order. Therefore, we limit our approach to exclude edges with the same priority but do not belong to the same state machine as successors. Figure 8.14 shows the result of $A \oplus B$ (with two CPUs) after the adjustment.

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

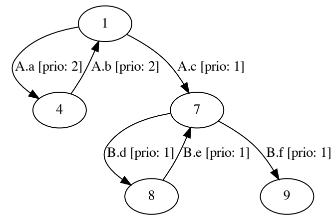


Figure 8.13: Result of $A \oplus B$ with adjustments and one CPU.

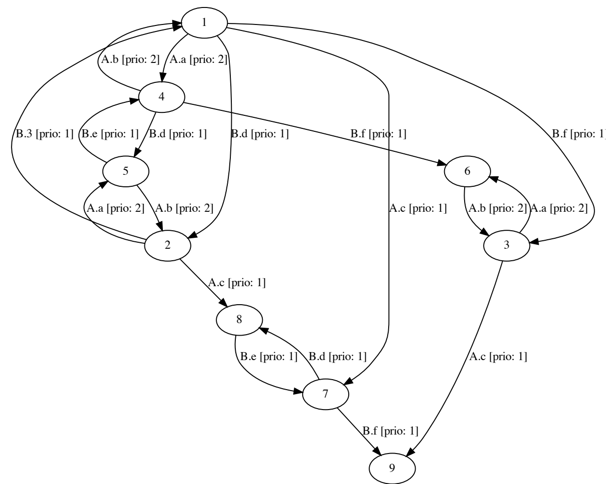


Figure 8.14: Result of $A \oplus B$ with adjustments and two CPUs.

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

8.5.2 Branches (If and Else Statements)

The changes made in Eq. (8.4) include another capability. It is now also possible to introduce a branch where the priorities determine the execution of the edges, a feature that will be useful in the next chapter. Figure 8.15 presents two state machines, A and B, with A containing a branch (if/else condition). By calculating $A \oplus B$, we get the result shown in Figure 8.16. The result clearly shows two possible paths, whereby the execution of edge e depends on its priority compared to the other edges in the path. It is interesting to observe that simple constructs can influence the paths in the desired way. Another requirement for spotting priority inversion is to create dependencies between concurrently executing processes. As introduced in Section 5.7, one possibility that synchronises state machines are semaphores. Within the next section, we use semaphores to create a sync pattern that ensures given dependencies.

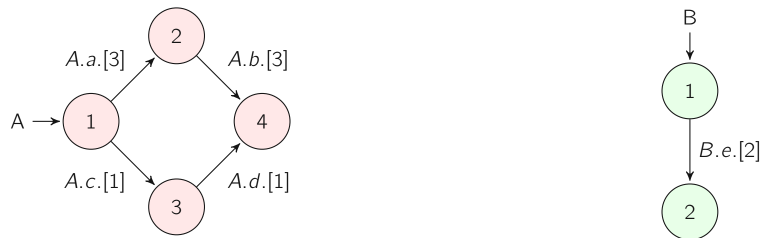


Figure 8.15: State machines A and B, with a A containing a branch (if/else condition).

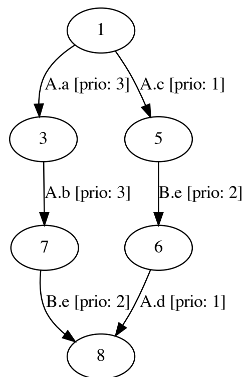


Figure 8.16: The two resulting paths of $A \oplus B$ with one CPU.

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

8.5.3 Sync Pattern

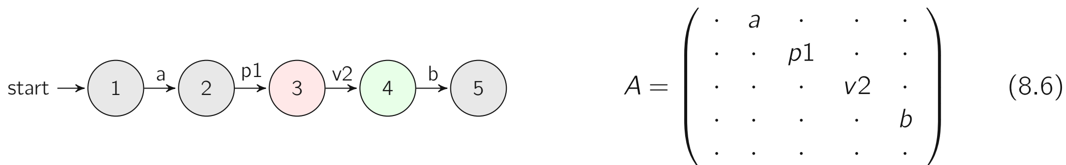
The above-presented approach is insufficient in detecting priority inversion, as there are no explicit dependencies between the processes. Therefore, introducing a mechanism that ensures that edges are executed in a specific order is imperative. Let us assume that we have two processes, A and B, each issuing two labels, a, b, and c, d, that must fulfil the relations given in Eq. (8.5).

$$\begin{aligned} a &\prec b \\ c &\prec d \\ \\ a &\prec d \\ c &\prec b \end{aligned} \tag{8.5}$$

These relations explicitly allow that a either precedes or succeeds c, and the same applies to the pair b and d. For implementing a synchronisation in Kronecker Algebra between the edges, we utilise semaphores as previously introduced in Section 5.7. In this case, two semaphores are enough to ensure the label relations.

Figure 8.17: The two semaphores S_1 and S_2 for synchronising the two processes A and B.

As a next step, processes A and B require adjustment. The order of labels in the processes sets the first two relations in Eq. (8.5). For implementing the last two relations, we use the semaphores S_1 and S_2, as shown in Figure 8.17. The idea is that process A locks S_1 and is blocked until B releases S_1. Letting B lock S_2 and A release S_2 afterwards ensures that the relations hold. Therefore, process A needs to include $p1 \prec v2$ and process B $v1 \prec p2$. Figures 8.18 and 8.19 show the adjusted processes and their matrix representations. We introduced colours to visualise the relations between the parts. The relation patterns $p1 \prec v2$ and $v1 \prec p2$ are interchangeable between A and B.



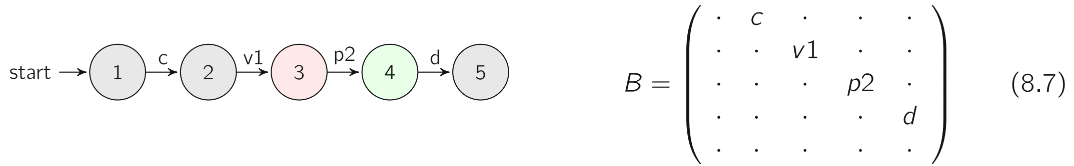
$$A = \begin{pmatrix} \cdot & a & \cdot & \cdot & \cdot \\ \cdot & \cdot & p1 & \cdot & \cdot \\ \cdot & \cdot & \cdot & v2 & \cdot \\ \cdot & \cdot & \cdot & \cdot & b \\ \cdot & \cdot & \cdot & \cdot & \cdot \end{pmatrix} \tag{8.6}$$

Figure 8.18: Process A including the required edges to access the semaphores S_1 and S_2.

After the adjustments, we utilise Kronecker Sum to combine A and B and Kronecker Skip for synchronisation with the semaphores. Eq. 8.8 shows the single elements, while I_5 and I_2 are the identity matrixes with the respective sizes.

$$(A \oplus B) \odot (S_1 \oplus S_2) = (A \otimes I_5 + I_5 \otimes B) \odot (S_1 \otimes I_2 + I_2 \otimes S_2) \tag{8.8}$$

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub



$$B = \begin{pmatrix} \cdot & c & \cdot & \cdot & \cdot \\ \cdot & \cdot & v1 & \cdot & \cdot \\ \cdot & \cdot & \cdot & p2 & \cdot \\ \cdot & \cdot & \cdot & \cdot & d \\ \cdot & \cdot & \cdot & \cdot & \cdot \end{pmatrix} \qquad (8.7)$$

Figure 8.19: Process B including the required edges to access the semaphores S_1 and S_2.

Figure 8.20 shows the result of Eq. (8.8). It is visible that in the beginning, there are different paths possible between the two processes. However, the synchronisation starts when A gets hold of the semaphore S_1. The reader will agree that the result fulfils all relations defined in Eq. (8.5). As a side comment, processes A and B can have further edges on both sides, as the semaphores act like a "tunnel" between the parts before and after. Therefore, it is essential not to create edges that would allow bypassing the semaphores; otherwise, the relations will not hold any longer. Continuative, the sync pattern can be used in different places of a more extended graph; however, with each "tunnel", a new pair of semaphores is added.

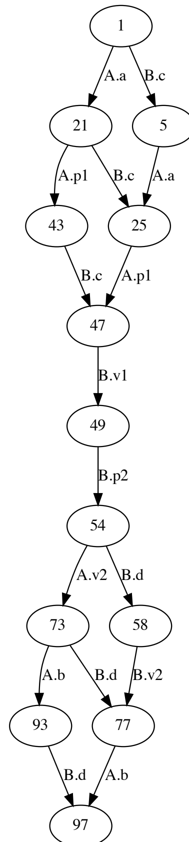


Figure 8.20: Graphical representation of $(A \oplus B) \odot (S_1 \oplus S_2)$

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

8.6 Spotting Priority Inversion

After addressing the synchronisation issues, all elements are available to simulate and detect priority inversion, as introduced in Section 8.2. Let us assume three processes $T1, T2, T3$ with different priorities and edges. $T1$ and $T3$ share the same critical section d, as shown in Figure 8.21.

Figure 8.21: The three processes $T1, T2, T3$, with $T1$ and $T3$ sharing a critical section d.

8.6.1 Introducing the Lock

First, we must introduce the lock around the critical section d with a semaphore $S1$ (cf. Figure 8.22). Figure 8.23 shows the labels $p1$ and $v1$, added into $T1$ and $T3$ to create a lock around d. We added the coloured nodes to show where we added the semaphore.

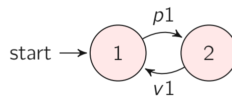


Figure 8.22: The semaphore $S1$ that creates a lock around the shared resource d.

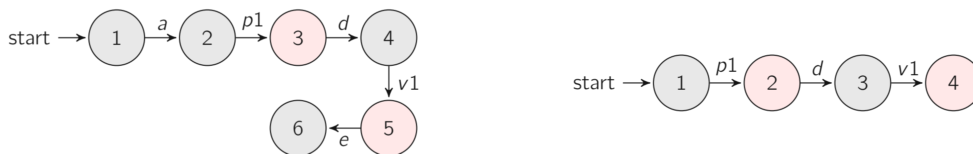


Figure 8.23: The processes $T1$, and $T3$ with $S1$ integrated.

8.6.2 Introducing the Synchronisation

The second step is introducing synchronisation between the processes. In our example, we need to address the following relations between the edges:

$$\begin{aligned} a &\prec d \\ a &\prec e \\ a &\prec c \end{aligned} \tag{8.9}$$

The relation $a \prec c$ requires synchronisation between $T1$ and $T2$. Therefore we introduce our first semaphore pair $(S2, S4)$, as shown in Figure 8.24. Another semaphore pair $(S3, S5)$ is required

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Figure 8.24: The semaphores $S2$ and $S4$ to synchronise $T1$ and $T2$.

Figure 8.25: The semaphores $S3$ and $S5$ to synchronize $T1$ and $T3$.

(cf. Figure 8.25) to ensure $a \prec d$ between $T1$ and $T3$. The last relation $a \prec e$, is part of $T1$ and does not require synchronisation. Note that each semaphore pair requires specific edge labels; otherwise, the synchronisation will not work. In the following, we integrate the semaphores into the processes as described in the previous section. The colours represent the respective elements coded where they are placed in the processes.

8.6.3 Adjusting the Processes

The changes in process $T1$ (cf. Figure 8.26) include two synchronisations. $S1$ needs to remain the first action after edge a in this process, as we want $T3$ to take the lock before any synchronisation happens. The order of the semaphore pairs after the lock is interchangeable and does not affect the synchronisation between the processes. However, they must happen before the critical section d, as we otherwise violate the relation $a \prec d$. For $T2$, we added the semaphore before c (cf. Figure 8.27). Process $T3$ also needs to fulfil $a \prec d$, but that includes the lock. Therefore, in this case, the semaphore pair is placed before $S1$ and d (cf. Figure 8.28). In addition, each label received a priority level. The semaphore priorities, take the priority of the process with the highest priority it synchronises.

Currently, the adjustments of the processes have to be done manually. In future works, if the dependencies are specified, there might be a solution to add the sync pattern automatically.

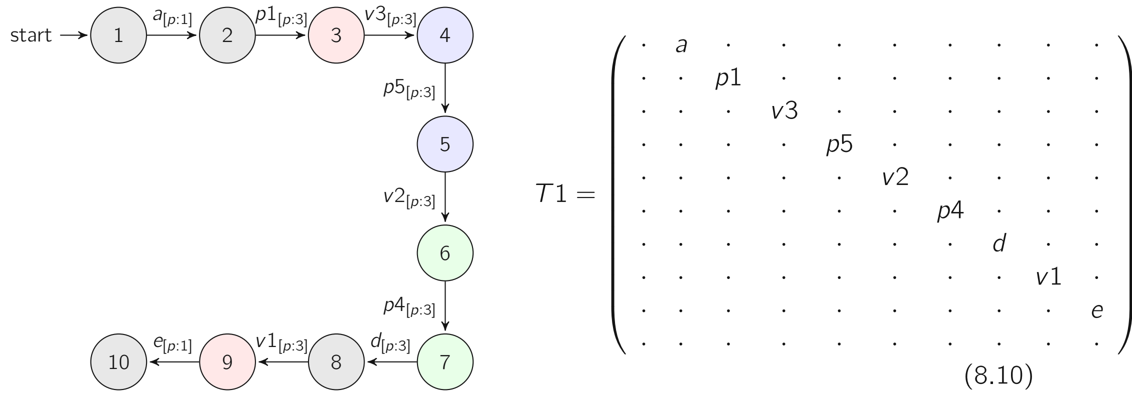
8.6.4 Apply Kronecker Algebra

As the last step, we have to compute all elements with Kronecker Algebra according to Eq. (8.13). For priority inversion to happen, we allocate one CPU.

$$(T1 \oplus T2 \oplus T3) \odot (S_1 \oplus S_2 \oplus S_3 \oplus S_4 \oplus S_5) \tag{8.13}$$

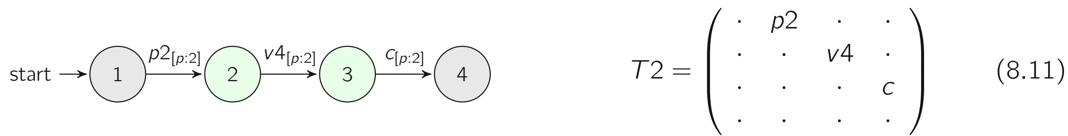
Figure 8.29 shows the graphical representation of the result of Eq. (8.13). The graph contains all synchronisation edges, making interpretation difficult at first sight. However, the reader will agree that the relevant process edges will form the behaviour as shown in Figure 8.32. The sequence of edges represents an unbound priority inversion, as the middle priority process, $T2$, interrupts the execution of $T1$ that holds the lock. Therefore, the high-priority process, $T3$, has to wait. The automatic creation of the timing diagram, is an open task for future work on the topic.

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN



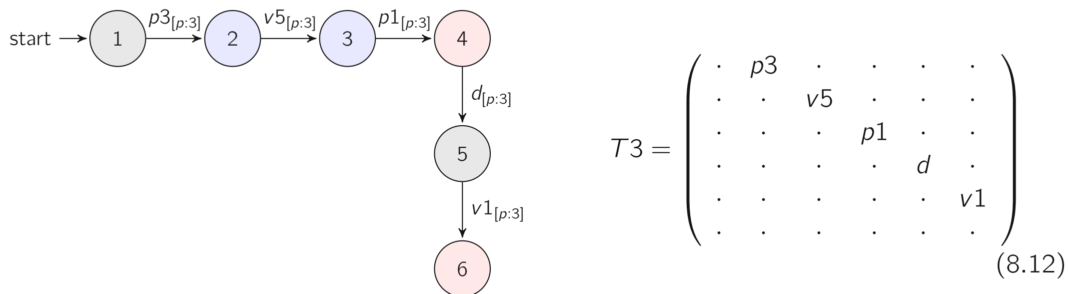
$$T1 = \begin{pmatrix} \cdot & a & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & p1 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & v3 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & p5 & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & v2 & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & p4 & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & d & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & v1 & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & e \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \end{pmatrix} \tag{8.10}$$

Figure 8.26: Process $T1$ adjusted with all semaphores and with priority 1.



$$T2 = \begin{pmatrix} \cdot & p2 & \cdot & \cdot \\ \cdot & \cdot & v4 & \cdot \\ \cdot & \cdot & \cdot & c \\ \cdot & \cdot & \cdot & \cdot \end{pmatrix} \tag{8.11}$$

Figure 8.27: Process $T2$ adjusted with all semaphores and with priority 2.



$$T3 = \begin{pmatrix} \cdot & p3 & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & v5 & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & p1 & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & d & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & v1 \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \end{pmatrix} \tag{8.12}$$

Figure 8.28: Process $T3$ adjusted with all semaphores and with priority 3.

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

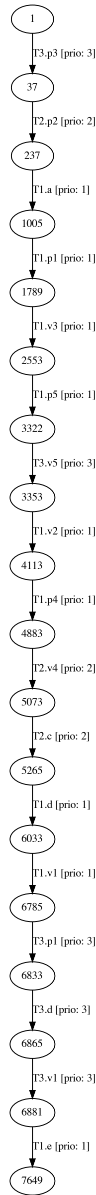


Figure 8.29: Graphical representation of Eq. (8.13) with normal priorities (Priority Inversion)

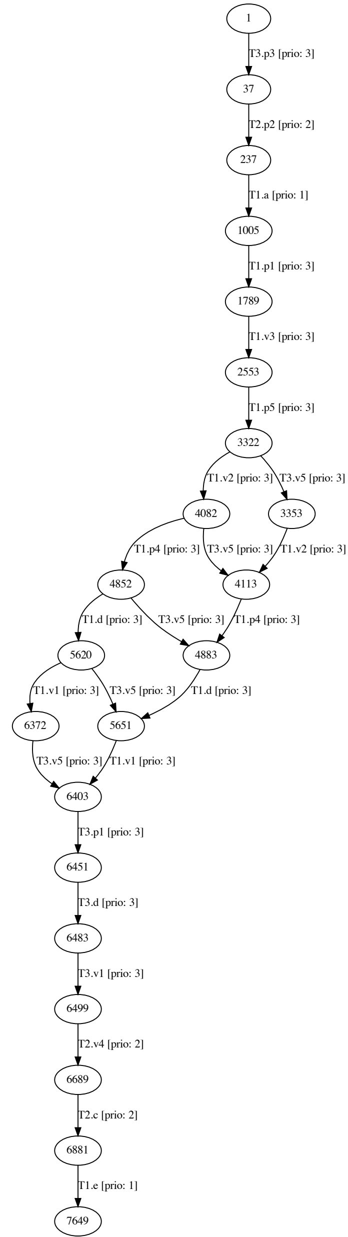


Figure 8.30: Graphical representation of Eq. (8.13) with adjusted priorities. (Priority Ceiling)

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

8.6.5 How to Spot Priority Inversion in the Graph?

One question that remains is how to spot priority inversion automatically. The answer lies in the order of the issued labels in the resulting graph. Let us define that all process labels are represented by e. The semaphores that protect the critical sections we call S_c and their labels p_c, v_c while c is the number of the critical semaphore. In addition, we summarise all synchronisation semaphores as S_N and their labels as s_l. The priorities we define with l = low, m = middle and h = high priority.[5]

For priority inversion to happen, a low priority (l) takes the lock p_c and blocks a high priority (h) process from executing. Therefore, we need to find a specific label sequence to spot priority inversion in the resulting graph. Figure 8.31 shows the pattern in the form of a CFG. The first state change happens by a process taking a critical lock; after that, an arbitrary number of synchronisation and process labels can appear. The next critical step is when at least one middle-priority process (m) issues a label. This state change can also be accompanied by synchronisation and process labels. The last two state changes represent the lower process releasing the lock and the higher priority process taking the lock.

Clearly we can utilise Kronecker Skip to find such a sequence in the final graph. If the sequence is not present, Kronecker Skip will produce an empty graph. The checking would need to be done for all critical semaphores and process combinations. Nevertheless, in the worst case, the process remains polynomial, while certain combinations are not required to be checked, e.g., check low-priority processes against each other.

In Figure 8.32, we coloured the relevant states and labels. The reader will agree that the sequence conforms to the CFG in Figure 8.31, and Kronecker Skip would create a positive result.

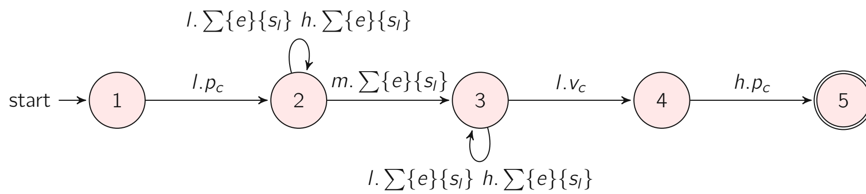


Figure 8.31: Critical label sequence for spotting priority inversion in the resulting graph.

8.6.6 Introduce the Priority Ceiling Protocol

As described, one mechanism to prevent priority inversion is the priority ceiling. In our example, changing the priority of label d in $T1$ to 3 creates such an effect, as it makes it impossible for $T2$ to interrupt $T1$, and $T3$ can access the lock after $T1$ is finished. Figure 8.30 shows the graphical representation of Eq. (8.13), but this time with adjusted priorities. In the final graph, the different synchronisation paths are visible between the processes, which leads up to ensuring the dependencies. After removing the synchronisation edges, Figure 8.33 shows how the processes execute.

If we Kronecker Skip the previously introduced label sequence on the new resulting graph, the result would be an empty graph as the sequence is not present. Implementing priority inheritance is impossible as it would require an adjustment of the priorities during execution.

[5] For better readability, can be extended or replaced with the priority number

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

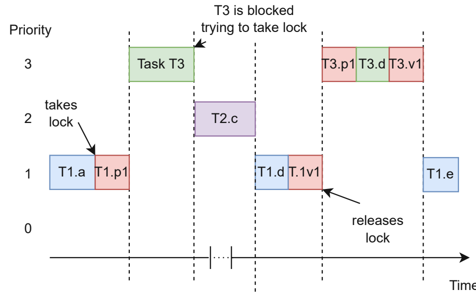


Figure 8.32: Priority Inversion Problem

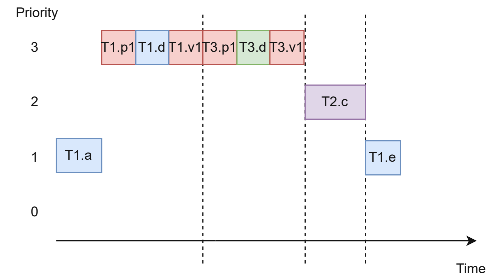


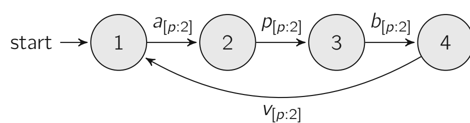
Figure 8.33: Priority Ceiling

8.7 Identify Starvation

Kronecker priority implementation can be used to show how a process can starve if never given access to the critical section. For this simple example, we use two processes $T1$ and $T2$ as shown in Figures 8.34 and 8.35. $T1$ is given priority two and $T2$ priority one. Each requests a semaphore $S1$ with two edges v and p.

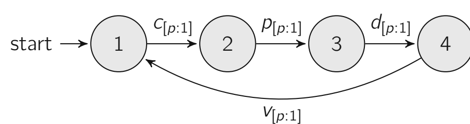
$$(T1 \oplus T2) \odot S_1 \tag{8.14}$$

By calculating Eq. (8.14) and providing only one CPU, Figure 8.36 shows the resulting graph. The graph only contains the edges of $T1$, in other words $T2$ never gets hold of the semaphore. If given two CPUs, the resulting graph shown in Figure 8.37, looks quite different. With two CPUs available, both processes can execute anytime. For this reason, several paths are visible in the graph, representing that behaviour.



$$T1 = \begin{pmatrix} \cdot & a & \cdot & \cdot \\ \cdot & \cdot & p & \cdot \\ \cdot & \cdot & \cdot & b \\ v & \cdot & \cdot & \cdot \end{pmatrix} \tag{8.15}$$

Figure 8.34: Process $T1$ with priority 2



$$T2 = \begin{pmatrix} \cdot & c & \cdot & \cdot \\ \cdot & \cdot & p & \cdot \\ \cdot & \cdot & \cdot & d \\ v & \cdot & \cdot & \cdot \end{pmatrix} \tag{8.16}$$

Figure 8.35: Process $T2$ with priority 1

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

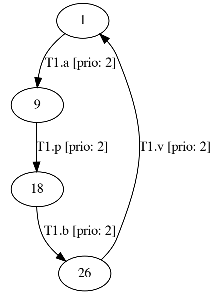


Figure 8.36: The graph of $(T1 \oplus T2) \odot S_1$ and one CPU. Processes $T1$ is executing while $T2$ starves.

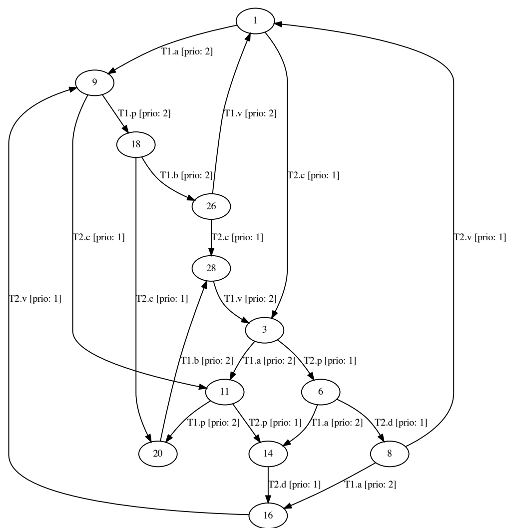


Figure 8.37: The graph of $(T1 \oplus T2) \odot S_1$ and two CPUs. Processes $T1$ and $T2$ are executing.

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

8.8 Discussion

This chapter discusses process priorities and the effect of priority inversion. We have adapted the existing lazy algorithm for identifying successor nodes to consider the edges' priorities. In addition, the adjustments allow a change in the available resources, which affects processes running in parallel. It could be shown that the changes can handle branching (if/else statements) and loops.

A synchronisation pattern that introduces dependencies between the edges of concurrently executing processes allowed us to identify priority inversion as described in [SRL90]. By altering the priorities of the edges involved, the priority inversion could be avoided, just as the priority ceiling protocol [GS88]. Implementing priority inheritance is impossible as Kronecker operations cannot display dynamic changes of an edge, i.e., an edge cannot change its priority within a graph.

Other restrictions are that processes containing loops must have edges of the same priority. Different priorities can be set; however, the algorithm will prioritise the first process to take the resource even though another process has a higher priority edge. This limitation could not be fixed as it would introduce indeterministic behaviour.

In summary, the adjustments described solve the problems mentioned in the problem definition in Section 8.4. As a side contribution, it could be shown that it is possible to identify process starvation when a process never gets access to a resource. An open task is to answer an interesting question, whether it would be possible to introduce a scheduler that controls the resources.

8.9 Related Work

Priority assignment-related work (for real-time systems) spans more than four decades [DCGBB16]. That includes real-time scheduling of complex models and schedulability analysis techniques developed to represent and analyse true-to-life systems, e.g., mixed-criticality systems, systems with deferred pre-emption, and probabilistic real-time systems with worst-case execution times. Moreover, topics such as pre-emptive and non-pre-emptive scheduling, single- and multi-processor systems, and networks are part of the activities. While this chapter provides solutions for handling priorities, the main intention was to contribute operations to Kronecker Algebra. In further activities, we intend to compare and explore the applicability of our approach in this research field. However, that is out of the scope of this dissertation. Therefore, we guide the interested reader to the survey by Davis et al. [DCGBB16], which provides an overview of the past and ongoing research activities.

Related work that includes priorities in the context of Kronecker Algebra is unknown to us. Closest is the research presented by Mittermayr and Blieberger [MB12, MB16a]. The authors use Kronecker operations to determine the worst-case execution time (WCET) of concurrently executing threads. There is a limited relation to Plateau's work in stochastic automata networks [Pla85].

8.10 Concluding Remarks

In the context of the research aim, the contribution of this chapter provides an essential mechanism for the next chapter. Nevertheless, the presented adaptation of the algorithm is not a sole extension; it enables further research ideas in the field of Kronecker operations.

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Chapter 9

Worst Case Execution Time Analysis

Within this chapter, we add another element in the search for unexpected system behaviour: the execution time of a process. As the execution time of a process is not constant, often the worst-case execution time (WCET) is assumed. The idea for this chapter builds upon Mittermayr and Blieberger [MB12, MB16a] and their work utilising Kronecker Algebra for WCET analysis of multi-core concurrent applications. First, we introduce the basics of WCET analysis, followed by explaining how the process of Mittermayr and Blieberger [MB12, MB16a, MB21] is affected by process priorities, and it is possible to calculate the WCET of a message path. An example of two processes synchronising over one semaphore acts as an evaluation case. We discuss the results compared to related work at the end of the chapter.

9.1 Introduction to timing analysis

While WCET analysis for sequential programs is widely considered as a solved issue [WEE+08], the scientific and industrial interest has shifted towards analysis and verification of multi-threaded applications. Nevertheless, the primary aim of timing analysis is to characterise the timing behaviour of programs or systems and provide guarantees on upper timing bounds [WEE+08]. Widely used time bounds are WCET- and best-case execution times (BCETs). WCET and BCET refer to the longest and shortest execution time of a program or task, respectively (cf. Figure 9.1). Worst case guarantees for programs or systems are larger than the WCET value. There are two main methods, static analysis and measurement-based, to obtain timing bounds [WEE+08].

In industry, a frequently used method for program timing analysis is by measurements [LGZ+09]. Numerous test-runs measure a program's execution time while varying the input parameters (cf. Figure 9.1). This kind of method provides an average timing behaviour or an approximate WCET value. However, there is a limitation of this type of analysis. Each run only follows one program path, i.e., if there are too many execution paths, this method is not suitable because the measurements underestimate the WCET. This method requires adding safety margins to ensure that the WCET value is not too low. These margins carry the risk of over- or under-provisioning of computing resources or cause scheduleability issues. Measurement equipment includes oscilloscopes, logic analysers, and in-circuit emulators on the hardware to measure execution times.

The static WCET analysis technique is more suitable for programs with multiple execution paths and stricter timing boundaries. This type of method does not execute the program but statically analyses the timing properties [PK89]. Static WCET tools tend to give larger WCET estimates (upper bounds) than the actual execution time without the need for additional margins. Simplified, a

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Figure 9.1: Basic notions of timing analysis [WEE+08].

WCET analysis consists of three parts: a flow analysis to distinguish the possible program execution paths, a low-level analysis to approximate times for atomic parts of the code (e.g., instructions, basic code blocks), and the calculation part to combine the two previous phases into a WCET approximation.

Flow analysis is about defining maximum loop bounds [GESL06] because the number of loop iterations affects the WCET estimates. Advanced tools include methods to determine loop bounds automatically; however, the manual annotation of loop bounds is still required in most cases. Another characteristic of the flow analysis is the possibility of recognising infeasible paths, i.e., paths that are feasible in the CFG, but inaccessible when studying the input data values and the semantics of the program [GESL06].

The low-level analysis considers that modern hardware features pipelines, caches, and out-of-order execution influence the timing behaviour of the program [SAA+15]. A technique to deal with these issues is using models, e.g., simulators, to enable the analysis without the actual hardware. Nevertheless, creating accurate models of a processor and hardware is complicated and sometimes introduces additional complexity into the analysis. Safe and straightforward processor models, on the other hand, lead to higher WCET bounds. The combination of flow and low-level analysis allows the calculation of the WCET. For more detailed information, readers are encouraged to consider reading the detailed survey about available methods and technological advances in determining timing guarantees by Wilhelm et al. [WEE+08].

With the increasing complexity of current software and hardware, there is a broad spectrum of research activities. The topics span from integer linear programming [Lis03], model checking [LGG+08], and tree-based calculation, [GESL06]. Other researchers investigate code conversion to WCET-analysable single-path code [PHP15] or specialized programming languages [LLK+08]. On the tooling side, there are a few commercial products available such as *aiT* [Abs21] from AbsInt or *RapiTime* [Rap21] from Rapita Systems, and academic open-source prototypes, such as T-CREST [SPH+18] or SWEET [GESL06].

9.2 Problem Definition

Now let us return to agent systems and their message interactions from the previous chapters. Figure 9.2 depicts agent A_1 receiving and sending two different message pairs on the left. In the simplest case, the time a message path requires is to add up the different sections, e.g., $a+b+c$.[1]

[1] We ignore for now the fact that the execution time and the message transmission time can vary.

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

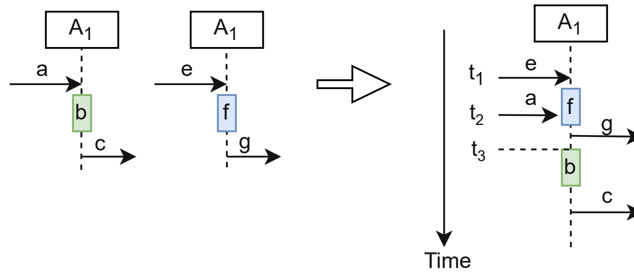


Figure 9.2: The agent A_1 receiving and sending two different message pairs.

However, as discussed in Chapter 6, messages can arrive at any time and might have different priorities, or the actions they trigger have different importance. Therefore, while a message might have arrived, the triggered action needs to wait before being executed. That, in turn, makes the message path time dependent on the other actions executed on the agent, e.g., $a + b + \Delta t + c$, while $\Delta t = t_3 - t_2$ is the time the process needs to wait for others to finish. In other words, the agent has become a critical section as in a multi-threaded application. This circumstance is best visible on the right side of Figure 9.2.

If the agents have only a first in, first out queue, the possible combinations will add up to a WCET, considering the most prolonged delay possible. If we add the priorities of the previous chapter, a process can be interrupted or scheduled before another process arrives earlier. However, the interactions between agents show the same effects as in multi-threaded concurrent software.

In the context of basic emergence, the effects of variations in execution time and message passing times have not been explored. Therefore, the problem definition is to extend Kronecker Algebra with the ability to determine the WCET of a message path while considering different priorities of the edges and available resources.

9.3 WCET Analysis of Shared Memory Concurrent Programs Running on a Multi-Core Architecture

Compared to the above-described sequential program analysis, the WCET analysis of multi-threaded concurrent software is more challenging. The main reason is that processes have connection points (communication) in the form of synchronisation, e.g., via shared memory accesses protected by critical sections. Other approaches assume little to no synchronisation except forking and joining [ORS13].

As mentioned at the beginning of the chapter, Mittermayr and Blieberger [MB12, MB16a, MB21] approach these difficulties using Kronecker Algebra. The author's approach is based on a concurrent program graph (CPG) data structure, which describes all possible interleavings and incorporates synchronisation while preserving completeness. In more detail, the approach uses reachable CPGs (RCPGs), a form of CPGs that only contains reachable nodes. RCPGs represent concurrent and parallel programs similar to control flow graphs (CFGs) for sequential programs.

Based on the RCPGs, it is possible to create dataflow equations for the timing analysis [Bli02]. Each RCPG node is represented by a data flow equation, which can then be solved with a dataflow-solver. The authors show that their approach can handle critical sections, loops, stalling times

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

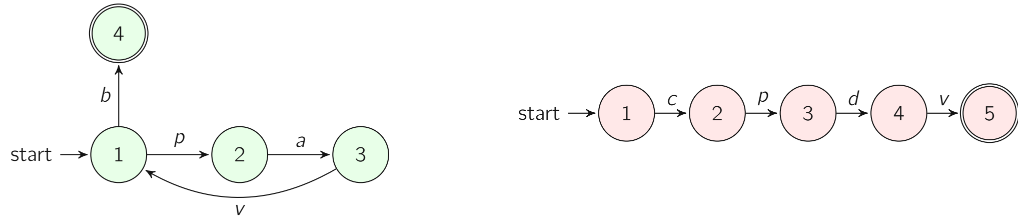


Figure 9.3: The two processes A and B with synchronisation labels p and v.

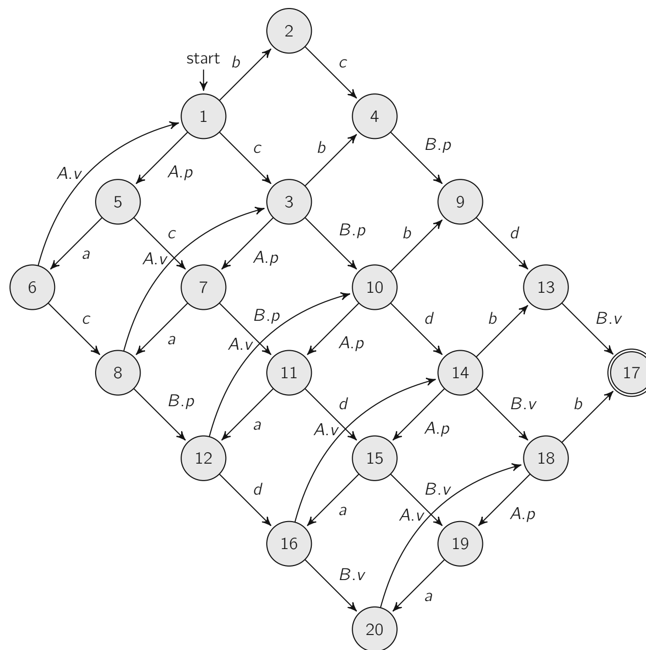


Figure 9.4: Result of $(A \oplus B)$

(e.g., caused by lock contention), forking and joining. In the following, we utilise the same ideas to determine the WCET of prioritised processes and point out the differences.

9.3.1 Modelling All Interleavings

As in the previous chapters, we resort to the fact that Kronecker Sum calculates all possible interleavings of two concurrently executing FSMs or general CFGs, including conditionals and loops [Küs91, Dav81, Gra18]. Figure 9.3 depicts two processes, A and B (both including semaphore calls), and Figure 9.4 depicts the result of $(A \oplus B)$. As expected, the resulting graph represents all interleavings between the two processes. However, it is interesting to note that the loop in A is copied five times which is the number of nodes present in B. As both threads have the same labels, p and v, we define that $X.(l)$ indicates that label l belongs to process X. Otherwise, the process that is calling the synchronisation label is unknown.

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

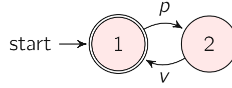


Figure 9.5: The binary semaphore S that is responsible for the synchronisation between the processes A and B.

9.3.2 Introduce Synchronisation

For creating a correct concurrent program's representation, it is required to introduce a synchronisation primitive. The reason is that Kronecker Sum produces interleavings containing all possible paths, even those containing semantically wrong uses of the semaphore operations. Therefore, a semaphore must be introduced by applying Kronecker Skip, as shown in Eq. (9.1). For this example, we use a binary semaphore S as shown in Figure 9.1. The skip operator synchronises only labels identical in the two input matrices. Therefore, the number of possible paths are limited so that the p- and v-operations are present in correct p-v-pairs (cf. Chapter 5.7).

$$(A \oplus B) \odot S = (A \otimes I_m + I_n \otimes B) \odot S \qquad (9.1)$$

Figure 9.6 depicts the result of Eq. (9.1). The loop in A is now only copied three times. As a side note, not only semaphores can be synchronisation primitives [MB12, MB11]. Burgstaller and Blieberger [BB14] use Ada's protected objects, and Mittermayr and Blieberger [MB16b] use barriers as synchronisation primitives. In addition, the later publication also shows that it is possible to model initially locked and unlocked semaphores in Kronecker Algebra.

9.3.3 WCET Analysis on RCPGs

Several additional steps are required to calculate the WCET of a concurrent program, which we summarise for completeness as we only need one of the steps below. For more details, we refer the reader to [MB21].

- The first step is determining the execution counts $e(k \to n)$ of each edge in the RCPGs. In other words, how often is edge $k \to n$ taken compared to the other outgoing edges of node k? The execution frequency is a rational number with a value $0 \leq e(k \to n) \leq 1$.
- The second step involves defining the number of loop iterations of each loop, loop iteration, and loop exit constraints. These numbers and constraints must be specified or calculated during the later maximisation process.
- As a third step, the authors define the synchronising nodes since they can create blocking situations.
- Based on the previous steps, each node of the RCPGs is assigned a data flow variable, and a data flow equation is set up. A vector represents the data flow variable, and each component of the vector reflects a processor (single thread) and is used to calculate the WCET of the corresponding thread. The final equations are solved by applying an algorithm presented in [HT66].

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

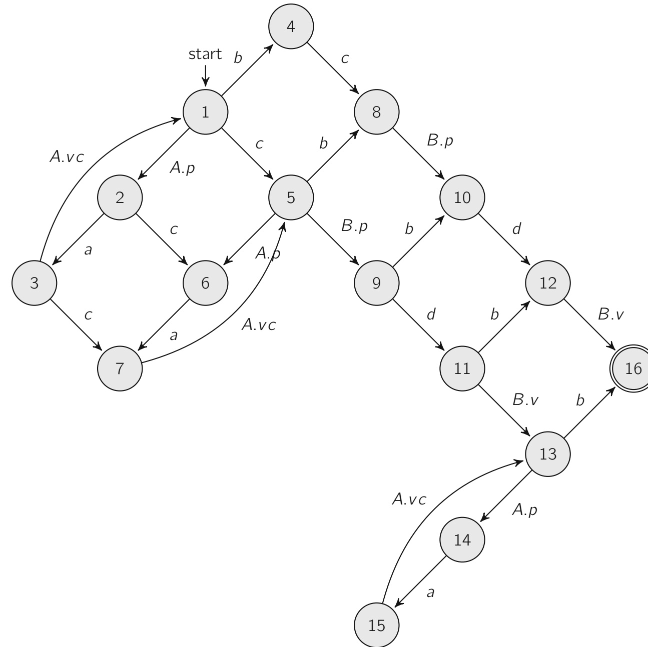


Figure 9.6: Result of $(A \oplus B) \odot S$

- Sometimes, the vp synchronising nodes have at least one outgoing loop entry edge. If so, the corresponding loop must be partially unrolled to be statically present in the RCPGs equations.
- The solution is differentiated in the last step, and the missing values are calculated in a maximisation process.

In summary, the presented approach allows determining an exact WCET analysis of shared memory concurrent programs running on a multi-core architecture. We utilise our priority functionalities in the following to calculate the WCET of two processes on a single-core CPU.

9.4 WCET Analysis with Priorities

Now, let us return to the example presented in Section 9.2 and align it with the two processes, A and B. The simplest way to calculate the WCET of a sequential process is to count all parts together. However, as introduced before, even in sequential programs, that can become difficult if loops or other constructs are involved.

In our example, process A contains one loop. The only way to deal with that is to introduce a variable that tells us how many times a loop is executed. Let $\mathbb{N}_0$ be a set of natural numbers, including zero (i.e., $\mathbb{N}_0 = 0,1,\ 2,\ \ldots$). We use from now on the variable $l_i \in \mathbb{N}_0$ as the counter for the number of loop iterations. We assume that this number is constant and statically known. For example, if the loop in A is only once executed $l_i = 1$, therefore we can calculate a simplified WCET of A by summing up the times given to the edge labels. The $WCET_A$ would be $WCET_A = 1 * (A_p + a + A.v) + b$.

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

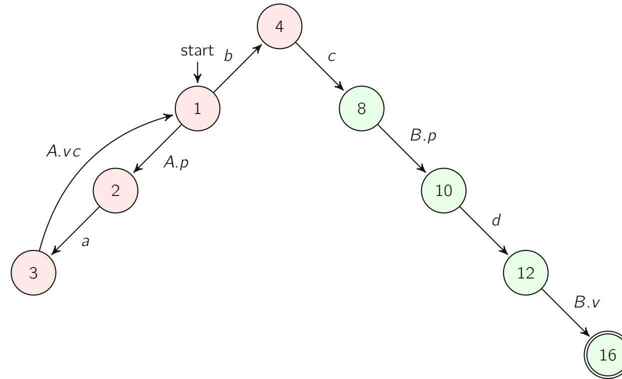


Figure 9.7: Result of $(A \oplus B) \odot S$ with A having the higher priority than B.

Now let us look at the graph in Figure 9.6. The reader will agree that starting node 1 is crucial as it decides which one of the two processes will get hold of the semaphore first. Under the premis, we only have one CPU available. If both processes have the same priority, the choice depends on the chance or time the process arrives in a waiting queue.

If we prioritise process A, the graph will change to the one Figure 9.7 depicts. Process A gets the semaphore first, and B has to wait until A finishes. That affects the execution times of both processes. For $WCET_A$, it remains the same as before if we still consider $l_i = 1$; for B, however, the execution time extends about the "waiting time", i.e., $WCET_B = c + B.p + d + Bv + WCET_A$. The situation changes if we switch priorities. Figure 9.8 shows the changed execution order of the two processes. In this case, the execution time changes for B to $WCET_B = c + B.p + d + Bv$ and for A to $WCET_A = l_i * (A_p + a + A.v) + b + WCET_B$.

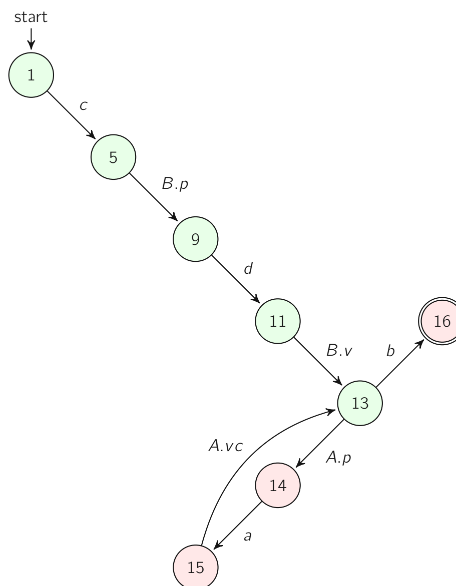


Figure 9.8: Result of $(A \oplus B) \odot S$ with B having the higher priority than A.

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

9.5 Discussion and Related Work

Within this chapter, we added priorities to the work presented by Mittermayr and Blieberger [MB21]. By adding the priorities and limiting the available shared resource to one CPU, the concurrently executing FSMs must wait until the resource is free. Depending on which FSM has the higher priority, the system graph gets reduced, and the WCET is a simple summation of the single-edge execution times. Automating the WCET calculation is possible by simply linking the times to the edge labels used for Kronecker operations.

The next step is adding more FSMs than available CPUs, and the FSMs have different priorities. However, for this step, a dataflow analysis like in Mittermayr and Blieberger [MB21] is required, which is out of the scope of this dissertation and part of future work. Moreover, other issues, such as barriers or protected elements, must be included. Chapter 10 shows that this limited approach can already impact the system's behaviour. Moreover, we show the first steps of building a test setup to verify the calculated WCETs with measured values.

Related work is limited to the publication of Mittermayr and Blieberger [MB21]; other work focuses on traditional WCET analysis not involving several CPUs and concurrently executing threads. Comparing our approach with standard WCET, analysis tools would go too far. Tools such as *aiT* [Abs21] from AbsInt or *RapiTime* [Rap21] from Rapita Systems can handle more sophisticated program constructs. The academic research covers many aspects unrelated to our aim when introducing priorities. For more detailed information about WCET analysis, readers are encouraged to consider the detailed survey about available methods and technological advances in determining timing guarantees by Wilhelm et al. [WEE+08].

9.6 Concluding Remarks

The contribution of this chapter is limited. Nevertheless, the results shown will be essential in the next chapter and are essentially an extension of the previous chapter. Moreover, the topic has the most significant potential for further research and will play an essential role in additional ideas related to Kronecker Algebra.

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

Chapter 10

Evaluation

This chapter presents example setups inspired by existing industrial communication systems. With the help of the previously introduced Kronecker operations, we can show that those systems can form simple patterns. Moreover, we present the work done on a time-predictable real-time communication platform. This platform will be essential for further research in WCET analysis supported by Kronecker operations. Lastly, we show the time complexity of all newly presented Kronecker operations in this dissertation.

10.1 Returning to the Blackboard

First, let us recapitulate the idea presented in Section 4.5. The basis of the idea is that interacting agents write their execution labels on the "environment" blackboard. Moreover, the chance for basic emergence exists if the sum of the agent languages differs from the systems language. Figure 10.1 shows the interacting agents writing on the blackboard. The connection to Kronecker Algebra is that FSMs can represent formal languages and be manipulated with all Kronecker operations introduced in the previous sections. However, the resulting graphs of Kronecker operations represent all possible execution paths of the agents over time. There is no time information like in a simulation which recodes the agent interactions step-by-step. That might appear as a negative aspect, yet with Kronecker operations, we get all execution paths, while in simulations, that might not be the case. The following presents how to utilise Kronecker Algebra to find patterns in interacting agent systems.

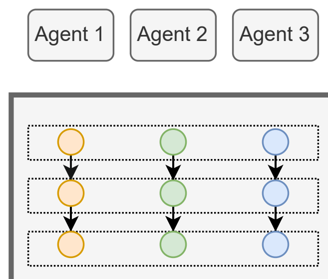


Figure 10.1: Three agents cooperating on a common blackboard (environment).

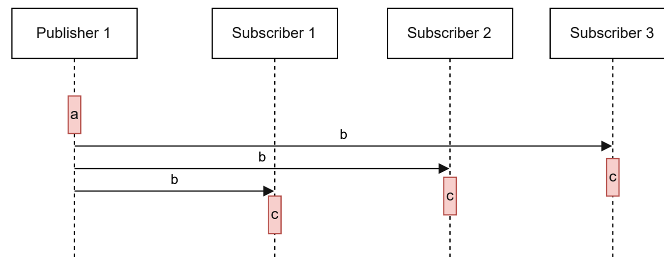


Figure 10.2: A simplified depiction of a publisher sending the same message to different subscribers.

10.2 Developing an Example

As a first step, we assume the agents communicate using a middleware that supports a publish and subscribe communication pattern. In publish and subscribe, the publisher sends a message into the network without waiting for an acknowledgement from the receiver. Only subscribers that subscribe to a topic will wait for a specific type of message and process it further, while the others will drop it. Figure 10.2 depicts a publisher sending the same message to different subscribers and the processing of it. PubSub strongly contrasts the client-server communication pattern, where the sender establishes a connection with the receiver before sending a message.

10.2.1 Creating the Basic Structure

To create the base for the further steps, we construct a publish-subscribe message exchange with FSMs. As we cannot forge the broadcasting of messages, we use as a publisher a binary semaphore P that allows repeatedly "sending" of a message to the subscribers (cf. Figure 10.3). Note that we omit the actual message to the subscribers as taking p of the semaphore has the same effect. The subscribers receive p and will execute an action, in our case, issue a label m. To not block the publisher, each subscriber needs to rerelease the semaphore with an edge v. There is no priority nor limiting amount of CPUs, as each entity is a compute node. Figure 10.4 shows the first subscriber, while further subscribers only differ in the labels they issue.

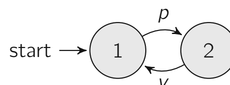


Figure 10.3: The binary semaphore P that represents the publisher.

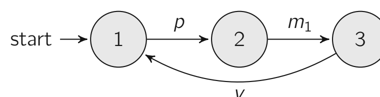


Figure 10.4: Subscriber 1

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

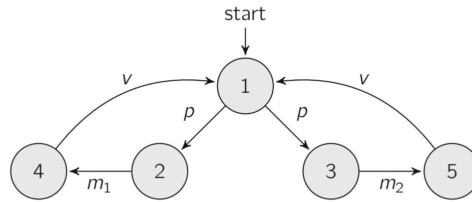


Figure 10.5: The resulting graph of Eq. (10.1)

Now, let us create a system with two subscribers. To crate a system representation, we can resort back to Kronecker Sum as in previous examples. Eq. (10.1) shows how to combine the subscribers SB_1 and SB_2 with the publisher semaphore P.

$$(SB_1 \oplus SB_2) \odot P \tag{10.1}$$

Figure 10.5 shows the graph representing the entire system. The edge p is either taken by one or the other subscriber that issues their message labels and returns over the edge v. While not entirely representing a publish-subscribe environment, the structure is sufficient for further steps. Moreover, the design is easily scalable (i.e., adding more subscribers), a feature we use in Section 10.7 to demonstrate the scalability of the lazy algorithm.

10.2.2 Introducing Interaction

In the basic structure above, the subscribers do not interact. As interacting agents are necessary for emergence to appear (cf. Section 2.3.4), we adjust the subscribers to interact indirectly as a next step.

The agents can interact again via a shared resource, as we used in Section 10.2.1. Let us assume a simple CFG takes the shared resource p, issues a label and finishes with v. The second CFG does the same. If we apply the same Eq. (10.1) on those two CFGs, only one CFG will execute first, followed by the other. The graph has, therefore, two paths, like in a condition.

However, that is not what we seek; the subscribers must react on each others actions. Additional semaphores between the subscribers can achieve such a connection. By adding a second semaphore into the subscribers, we get subscribers as shown in Figure 10.6. Note: In this way we can add several subscribers together, by simply adding semaphores between them. Now as we have two semaphores we need to adjust our equation as follows:

$$(SB_1 \oplus SB_2) \odot (SEM_1 \oplus SEM_2) \tag{10.2}$$

Figure 10.7 depicts the result of Eq. (10.2). The two subscribers execute now only in one order, depending on how the second semaphore is placed. By adding semaphores, it is possible to create interesting interactions between the single CFGs or FSMs. A circumstance that we will use in the next subsection to create a first pattern.

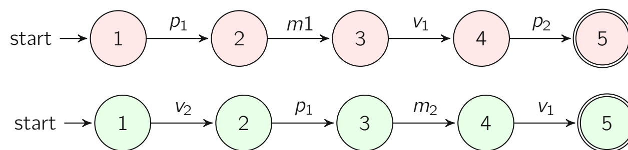


Figure 10.6: The two subscribers SB_1 and SB_2

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

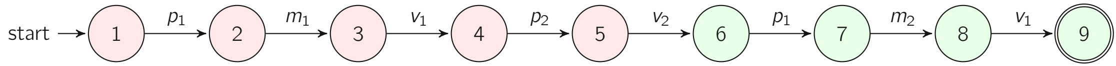


Figure 10.7: The resulting graph of $(SB_1 \oplus SB_2) \odot (SEM_1 \oplus SEM_2)$

10.2.3 A Pattern Emerges

Now let us return to our publish-subscribe model and introduce semaphores as described before. Figure 10.8 shows the first subscriber; here, we added another label a, as a starting point, and edge b provides a better distinction where the subscriber ends. The second semaphore we add as a possible exit at node two p_2. As we learned, the second semaphore creates an order between the subscribers. Therefore the second subscriber (cf. Figure 10.9) waits until the first subscriber issues p_2. As a side note, the position of the second semaphore is flexible depending on the desired effect. The second semaphore creates the graph displayed in Figure 10.10 after applying Eq. (10.2).

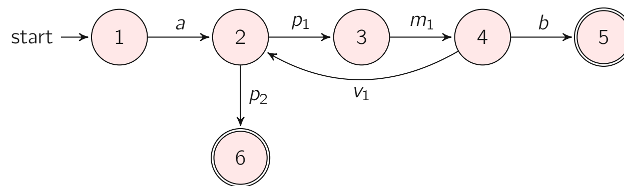


Figure 10.8: Subscriber 1 (SB_1)

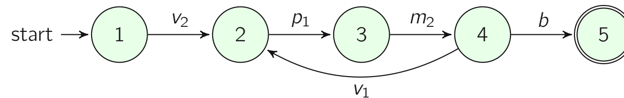


Figure 10.9: Subscriber 2 (SB_2)

The graph shows the expected behaviour that SB_2 can only execute when SB_1 issues the label p_2. Adding other agents, in the same way, can create different patterns. For example, the graph would extend to the left side by adding another semaphore exit on node two at SB_2 and adding another subscriber. There is also the possibility to add back paths in the subscribers; however, that will increase the amount of paths in the resulting graph which is not the aim of this simple example.

Another interesting pattern created by Eq. (10.2) is visible in the output matrix. Figure 10.11 shows the terminal output representing all entries created by Kronecker Operations. However, as mentioned before, not all labels contribute to the final graph, as only a handful of edges are reachable. We marked the relevant edges and added some lines for better visibility. The triangles that appear represent the "loops" within SB_1 and SB_2. We do not know how often the loops will be executed; therefore, the total system could generate a repetitive pattern. While not that impressive initially, the reader might agree that despite the applied restrictions (semaphores) between the agents, the system already shows a behaviour the single agents cannot perform. For example, the word (a, p_1, m_1, v_1, p_2) is part of SB_1's, and the word (v_2, p_1, m_2, b) of SB_2's language. However, the subscribers cannot execute the word $(a, p_1, m_1, v_1, p_2, v_2, p_1, m_2, b)$, which is only executable in the total system.

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

For an observer, the system's behaviour cannot be determined by the agents' actions alone. This condition partly fulfils the idea in Section 4.5, where the entire systems language differs from the sum of the agent languages.

This example also fulfils some other basic requirements for emergent patterns to appear. The reader will agree that the two agents are interacting [CDF+20] via the common semaphore. If there is no interaction, the two agents will execute in parallel, but no pattern will arise that cannot be deducted to one of the agent's languages. Another requirement is the absence of centralised control [Ode02a] that tells the agents what to do. While we designed the agents to behave that way, there is no control during the execution. We will see that more clearly in the following example. Moreover, if we consider the loop patterns in the matrix as a higher-level system behaviour, the possibility exists that the system shows a micro-macro effect [Gol99]. Nevertheless, if the system fulfils the requirement of autonomy [Sha01] and adaptability or robustness w.r.t. changes [Gol99] cannot be confirmed. The reason is that the example only contains two agents; if one fails, the other will not continue to interact. Another issue here, as Kronecker Algebra provides us with all possible paths at once, it is impossible to remove an agent during execution.

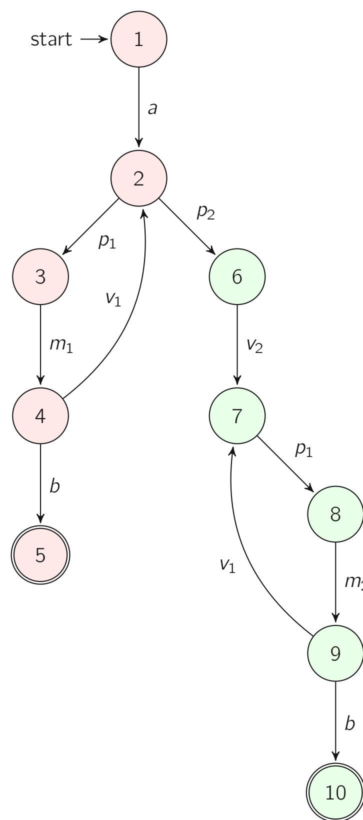


Figure 10.10: The resulting graph of $(SB_1 \oplus SB_2) \odot (SEM_1 \oplus SEM_2)$

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

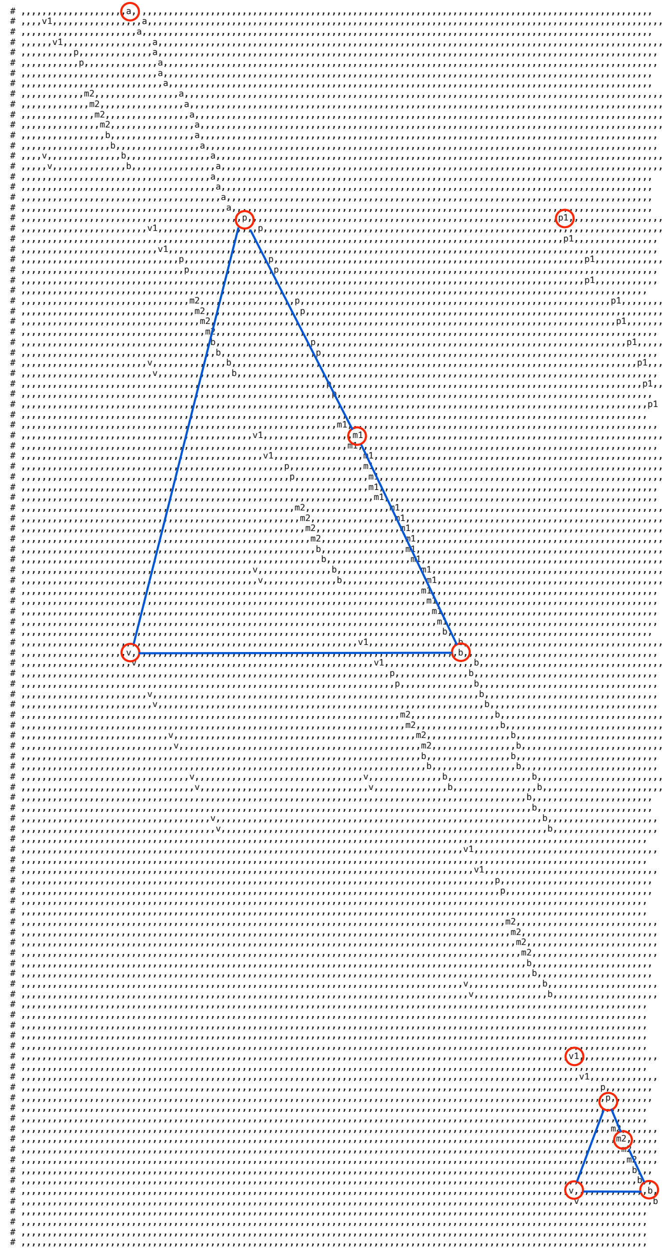


Figure 10.11: Terminal output of $(SB_1 \oplus SB_2) \odot (SEM_1 \oplus SEM_2)$

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

10.3 A Second Pattern

After showing that the previous simple setup can show patterns, let us now adjust the publish-subscribe environment. The idea is to increase the possibilities for the agents to interact while still keeping it simple enough to fit into these pages. First, we adjust our publisher semaphore to a self-loop read-write state machine.[1] As follows, we want to adjust the concept of how the message passing between the agents happens. The publisher shall write (w) a value into a memory bank (the semaphore) and creates a new value to publish (n). After seizing the value (r), the subscriber executes a task ($m1$) and is ready to read another value. In the most straightforward case, the subscriber can read the value as many times as possible until the publisher blocks the memory again. The same applies to the publisher. Therefore, the semaphore must accept an arbitrary number of read and write access from the publisher or the subscriber. Figure 10.12 shows all three FSMs. Without any precautions, which we now explicitly leave out of consideration, we get the graph shown in Figure 10.13 after applying $(P \oplus S_1) \odot S$.

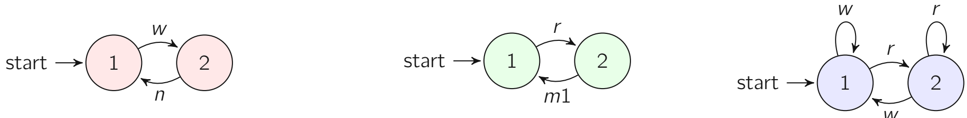


Figure 10.12: On the left the publisher P in the middle a subscriber S_1 and on the right the adjusted semaphore S

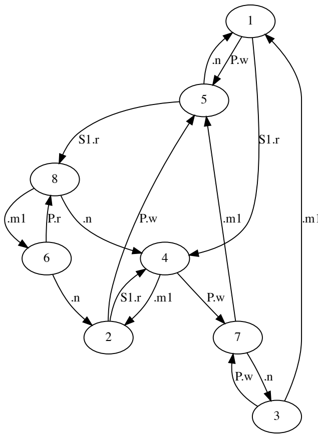


Figure 10.13: Resulting graph of $(P \oplus S_1) \odot S$.

[1]The same behaviour could be achieved with one node and self-loops; however, our implementation requires at least two nodes

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

10.3.1 Analysing the Paths

The resulting graph in Figure 10.13 shows some differences from the previous example. First, there is no final node, i.e., the total system will never stop executing, a behaviour well suited to create patterns. Another quite visible difference is the loops between some nodes, which are also visible in the matrix output shown in Listing 10.1. In the following, we listed the four node pairs that have loops:

$$1 \triangleright 5 \triangleright 1 \quad | \quad 2 \triangleright 4 \triangleright 2 \quad | \quad 3 \triangleright 7 \triangleright 3 \quad | \quad 6 \triangleright 8 \triangleright 6$$

If we now follow the different execution paths in the total system graph, we receive several possibilities. We ignored the possibilities of a return loop between the nodes above and only considered edges "moving" forward. In total, we found nine paths starting from node one and returning. We also added to each path the edge labels the system would issue.

- $1 \triangleright 4 \triangleright 7 \triangleright 5 \triangleright 1 \quad = rwm_1n$
- $1 \triangleright 4 \triangleright 2 \triangleright 5 \triangleright 1 \quad = rm_1wn$
- $1 \triangleright 4 \triangleright 7 \triangleright 3 \triangleright 1 \quad = rwnm_1$
- $1 \triangleright 5 \triangleright 8 \triangleright 4 \triangleright 2 \triangleright 5 \triangleright 1 \quad = wrnm_1wn$
- $1 \triangleright 5 \triangleright 8 \triangleright 4 \triangleright 7 \triangleright 5 \triangleright 1 \quad = wrnwm_1n$
- $1 \triangleright 5 \triangleright 8 \triangleright 4 \triangleright 7 \triangleright 3 \triangleright 1 \quad = wrnwnm_1$
- $1 \triangleright 5 \triangleright 8 \triangleright 6 \triangleright 2 \triangleright 5 \triangleright 1 \quad = wrm_1nwn$
- $1 \triangleright 5 \triangleright 8 \triangleright 6 \triangleright 2 \triangleright 4 \triangleright 7 \triangleright 5 \triangleright 1 \quad = wrm_1nrwm_1n$
- $1 \triangleright 5 \triangleright 8 \triangleright 6 \triangleright 2 \triangleright 4 \triangleright 7 \triangleright 3 \triangleright 1 \quad = wrm_1nrwnm_1$

As mentioned, the system does not have final nodes and, therefore, can execute endlessly and, in this case, always returns to the start node. In search for patterns, we transformed the last path into a graph and removed edges not taken in this case. Figure 10.14 shows that this path forms a ring interrupted by return loops. If the system, due to an unknown reason, follows this path in the same manner, it would produce a message pattern that represents a higher level of information. The word produced this way could only be executed by the total system and not by the agents alone. In addition, the created pattern is equal to an increase of information indicating an emergent pattern and, therefore, the system shows a micro-macro effect [Gol99]. Like the previous example, the system fulfils some basic requirements for emergent patterns to appear, e.g., interaction and no centralised control.

Listing 10.1: Terminal output of $(P \oplus S_1) \odot S$

```
#  , , , r , w, , ,
#  , , , r , w, , ,
# m1, , , , , , w,
#  ,m1, , , , , w,
# n , , , , , , , r
#  ,n , , , , , , r
#  , , n , , m1, , ,
#  , , , n , , m1, ,
```


Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

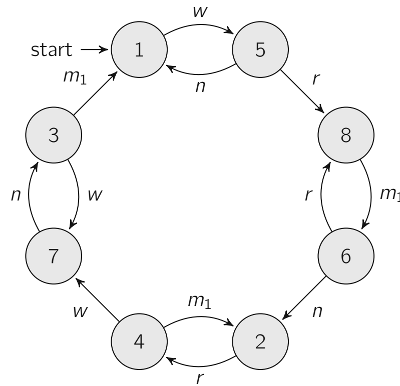


Figure 10.14: The resulting graph of the last execution path

10.3.2 Adding Another Subscriber

Let us now add another subscriber S_2 which will access the same memory block of the publisher. After calculating $(P \oplus S_1 \oplus S_2) \odot S$, we get the graph in Figure 10.15. The resulting graph contains more node pairs with loops in between, which are:

$1 \triangleright 9 \triangleright 1$ | $2 \triangleright 6 \triangleright 2$ | $3 \triangleright 11 \triangleright 3$ | $6 \triangleright 8 \triangleright 6$ | $5 \triangleright 13 \triangleright 5$ | $7 \triangleright 15 \triangleright 7$

$14 \triangleright 16 \triangleright 14$ | $10 \triangleright 14 \triangleright 10$ | $10 \triangleright 12 \triangleright 10$ | $16 \triangleright 12 \triangleright 16$ | $2 \triangleright 4 \triangleright 2$ | $4 \triangleright 8 \triangleright 4$

By identifying again paths that start and end by the final node, we get this time different types of paths that are interrupted by the return loops. In the following a few example paths, with a $\triangleright$ indicating the position of a return loop.

- Paths with one loop ($\triangleright$):
  - $1 \triangleright 6 \triangleright 13 \triangleright 9 \triangleright 1 \quad = r_1 w m_1 n$
  - $1 \triangleright 4 \triangleright 11 \triangleright 9 \triangleright 1 \quad = r_2 w m_1 n$
- Paths with two loops $\triangleright$:
  - $1 \triangleright 4 \triangleright 2 \triangleright 9 \triangleright 1 \quad = r_2 m_2 w n$
  - $1 \triangleright 6 \triangleright 2 \triangleright 9 \triangleright 1 \quad = r_1 m_1 w n$
- Paths with three loops $\triangleright$
  - $1 \triangleright 6 \triangleright 2 \triangleright 4 \triangleright 11 \triangleright 3 \triangleright 1 \quad = r_1 m_1 r_2 w n w$
  - $1 \triangleright 6 \triangleright 8 \triangleright 4 \triangleright 11 \triangleright 3 \triangleright 1 \quad = r_1 r m_1 w n w$
  - $1 \triangleright 4 \triangleright 2 \triangleright 6 \triangleright 13 \triangleright 5 \triangleright 1 \quad = r_2 m_2 r_1 w n m_1$
  - $1 \triangleright 4 \triangleright 8 \triangleright 6 \triangleright 13 \triangleright 5 \triangleright 1 \quad = r_2 r_1 m_2 w n m_1$
- Paths start and end in the same loop $\triangleright$:

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

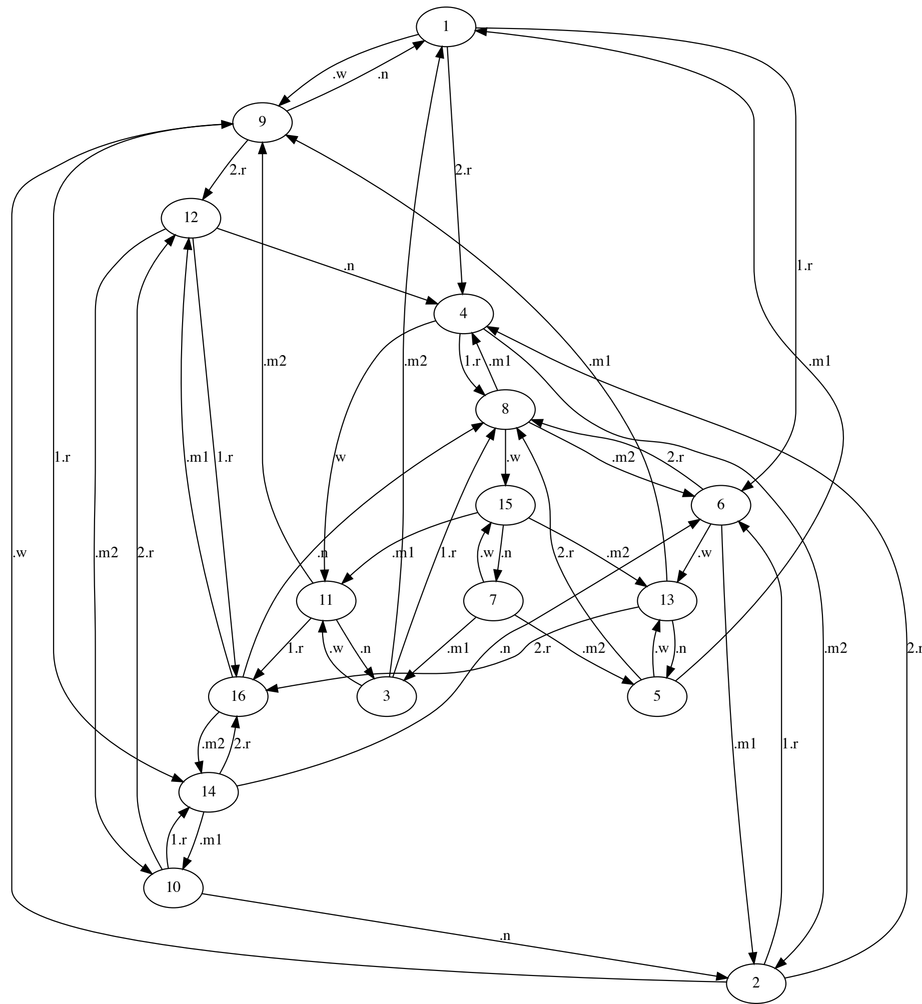


Figure 10.15: Resulting graph of $(P \oplus S_1 \oplus S_2) \odot S$.

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

– $1 \triangleright 9 \triangleright 12 \triangleright 10 \triangleright 2 \triangleright 9 \triangleright 1 \quad = wr_2m_2nww$
– $1 \triangleright 9 \triangleright 14 \triangleright 6 \triangleright 13 \triangleright 9 \triangleright 1 \quad = wr_1nwm_1w$
– $1 \triangleright 9 \triangleright 12 \triangleright 4 \triangleright 11 \triangleright 9 \triangleright 1 \quad = wr_2nwm_2n$

- Paths with four return loops $\triangleright$ and one straight edge in between:

→ $1 \triangleright 9 \triangleright 12 \triangleright 16 \triangleright 8 \triangleright 4 \triangleright 11 \triangleright 3 \triangleright 1 \quad = wr_2r_1nm_1wm_2w$
– $1 \triangleright 9 \triangleright 14 \triangleright 16 \triangleright 8 \triangleright 6 \triangleright 13 \triangleright 5 \triangleright 1 \quad = wr_1r_2nm_2wnm_1$
– $1 \triangleright 9 \triangleright 12 \triangleright 10 \triangleright 2 \triangleright 4 \triangleright 11 \triangleright 3 \triangleright 1 \quad = wr_2m_2nr_2wnm_2$
– $1 \triangleright 9 \triangleright 14 \triangleright 10 \triangleright 2 \triangleright 6 \triangleright 13 \triangleright 5 \triangleright 1 \quad = wr_1m_1nr_1wnm_1$
– $1 \triangleright 6 \triangleright 8 \triangleright 15 \triangleright 7 \triangleright 3 \triangleright 11 \triangleright 9 \triangleright 1 \quad = r_1r_2wnm_1wm_2n$
→ $1 \triangleright 6 \triangleright 8 \triangleright 15 \triangleright 7 \triangleright 5 \triangleright 13 \triangleright 9 \triangleright 1 \quad = r_1r_2wnm_2wm_1n$

When drawing some of the paths, namely the ones indicated by an →, we receive in the first case Figure 10.16. Despite having a second subscriber, the circle shape is identical to the previous one with different edge labels. Looking at the second path shown in Figure 10.17, we find a similar circle that starts instead with a loop with a straight edge. In addition, due to nodes such as the number nine or eight, the system could create shapes like eight or several loops in a row. For example, if the system chooses every second time to choose either edge r_2 or n, the created path looks like an eight.

While also in this example, the created patterns are not as exciting as those of shells, fish and mammals or lichen growth, it already shows some simple patterns. One fascinating factor is that only three agents interact with each other and barely have any rules on how to do that. That aligns with organisms that use relatively simple behavioural rules to generate structures and patterns on the global system level. Those structures are more complex than the components and processes they emerge [Pag88]. Nevertheless, despite how interesting it is to identify patterns, it is not the primary aim of this dissertation. Therefore, let us explore what happens if we change the system by adding a condition.

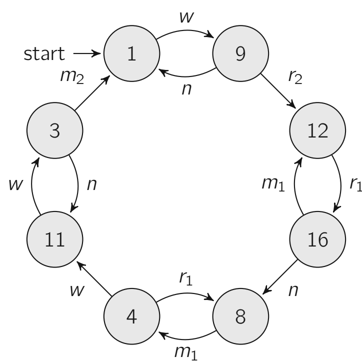


Figure 10.16: The resulting graph of the first path with four loops and straight edge in between.

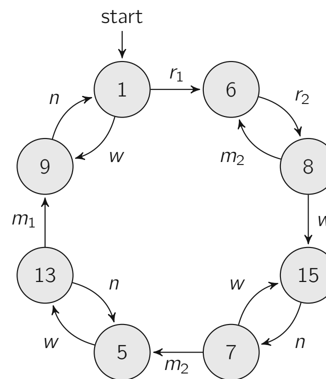


Figure 10.17: The resulting graph of the last path with four loops and straight edge in between.

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

10.4 Change the Basic Setup

In this setup we add another publisher to the system and remove one subscriber. The idea is to demonstrate how the systems behaviour changes when introducing new elements and conditions. The new subscriber contains a condition, i.e., the subscriber can only read the value of one publisher and issues a specific label before another value can be read. Figure 10.18 shows the new agent configurations. As there are now two publishers we also need to introduce a second semaphore where the messages can be read by the subscriber. The new operation to get a systems graph changes accordingly to $(P_1 \oplus P_2 \oplus S_{U1}) \odot (S_1 \oplus S_2)$.

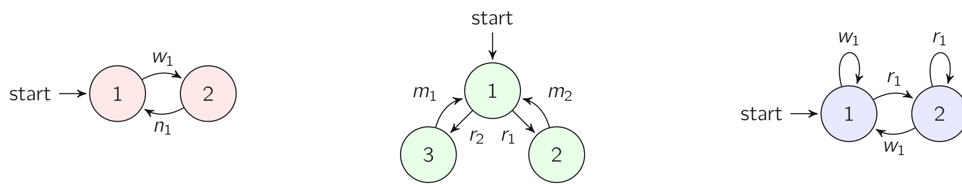


Figure 10.18: On the left the publisher P_1 in the middle a subscriber S_{U1} and on the right the adjusted semaphore S_1

The result shown in Figure 10.21 is, at first sight, not as admirable as the previous one. A fact that is visible is that no apparent loops can be used to find patterns. However, as the graph does not have end nodes, the chance is that a repetitive pattern is hidden in the paths. The reader will agree that following each path and writing down the labels would get out of hand. Nevertheless, we can again use the strength of Kronecker Skip Operation to search for a pattern in the graph. We have learned in the previous example that a specific sequence of edge labels forms a loop. Therefore we can check if the sequence r_2, w_2, n_2, m_2 exists in the large graph. We need to consider that the sequence needs to be in a loop; otherwise, we will block other loops in the graph. Figure 10.19 shows the label sequence L_1 and Figure 10.22 the result of $S_{Total} \odot L_1$.

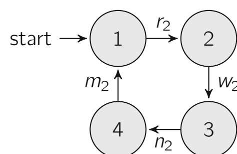


Figure 10.19: The label sequence L_1

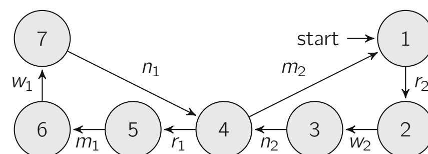


Figure 10.20: The label sequence L_2

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

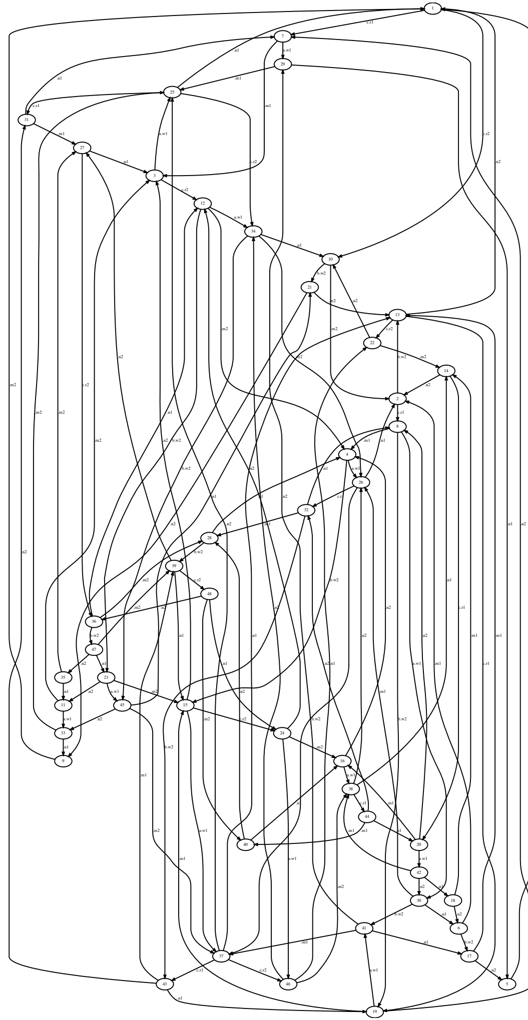


Figure 10.21: Resulting graph of $S_{Total} = (P \oplus S_1 \oplus S_2) \odot S$.

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

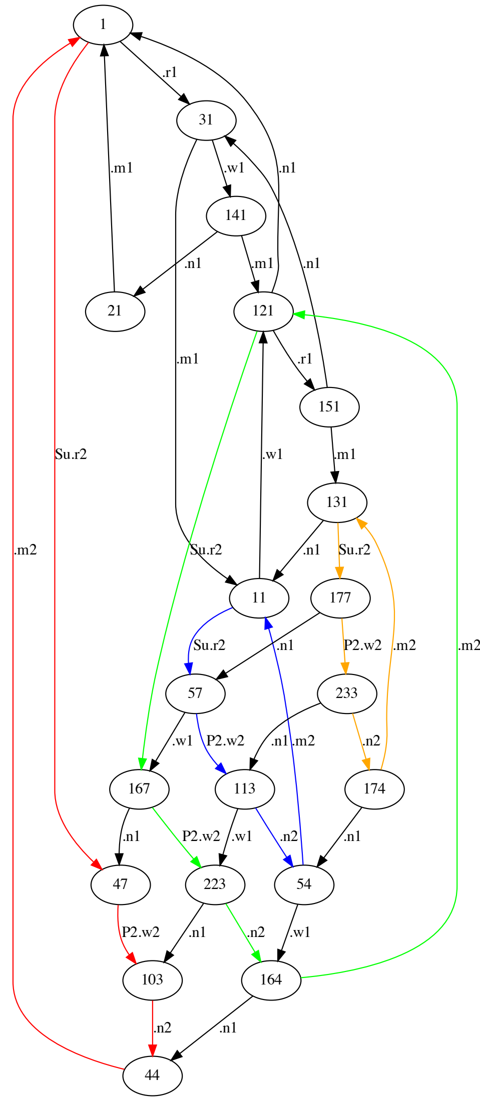


Figure 10.22: Resulting graph of $S_{Total} \odot L_1$

The reduced graph shows four loops in S_{Total} have the given label sequence. We know now that the system can produce such a pattern in certain circumstances. We can dissect the graph for other patterns if we apply different sequences. Figure 10.20 shows another sequence L_2, creating a loop between the two publishers, a behaviour expected to happen in such a setup. As we assume L_2 to be present in S_{Total} we apply this time Kronecker product $S_{Total} \odot L_2$. The resulting graph is $\simeq L_2$ which shows that L_2 can be executed by S_{Total}.

Besides discovering some limited patterns in this setup, there is another indirect finding. While the S_{Total} is relatively large and allows various execution paths, repetitive patterns are limited. One reason could be the introduced condition in the subscriber, as it limits the interleavings between the paths. It would require further research to determine what type of structures within the agents create specific patterns. Next, we will see if priorities between the agents influence the pattern formation.

10.5 Apply Priorities to the First Example

In the following, we reuse the setup in Section 10.3 with two subscribers. The idea is to give the subscribers different priorities and observe the effects on pattern formation. Moreover, the possibility to adjust the number of CPUs influences the environment the agents "live" in. In other words, the agents cannot write to the blackboard whenever they want; they need to wait until they get the resource. Adjusting the priorities of each edge or label of an agent opens up many variations.

The reader will agree that one CPU and each agent having the same priority will lead to the same outcome as in Figure 10.15. Giving the agents three different priorities and one CPUs will only allow the highest priority agent to execute. This happens because the publisher and the subscribers have loops and if both edges have the same higher priority the loop will be executed constantly.

Let us now adjust the priorities in the following way. The publisher gets a higher priority than the two subscribers, plus we restrict the available CPUs to two, to see some effects. Figure 10.23 shows a different graph as in the previous section. A first visible difference is that in the graph the number of loops is decremented by one, despite having the same amount of nodes.

The loops can be found between:

$$1 \triangleright 9 \triangleright 1 \quad | \quad 2 \triangleright 4 \triangleright 2 \quad | \quad 3 \triangleright 11 \triangleright 3 \quad | \quad 4 \triangleright 8 \triangleright 4 \quad | \quad 5 \triangleright 13 \triangleright 5 \quad | \quad 6 \triangleright 8 \triangleright 6$$

$$7 \triangleright 15 \triangleright 7 \quad | \quad 10 \triangleright 14 \triangleright 10 \quad | \quad 10 \triangleright 12 \triangleright 10 \quad | \quad 12 \triangleright 16 \triangleright 12 \quad | \quad 14 \triangleright 16 \triangleright 14$$

As most of the loops have not changed, we checked again for similar paths we found in the previous section. Interestingly, there are longer paths in the graph than before. Figure 10.24 shows the marked "→" path, and there is now one additional double loop. However, considering the existence of similar paths in this priority-adjusted example indicates that the pattern formation of the system still exists.

- → $1 \triangleright 9 \triangleright 14 \triangleright 16 \triangleright 8 \triangleright 6 \triangleright 2 \triangleright 4 \triangleright 11 \triangleright 3 \triangleright 1 \quad = wr_1r_2nm_2m_1r_2wnm_2$
- $1 \triangleright 9 \triangleright 14 \triangleright 10 \triangleright 2 \triangleright 4 \triangleright 11 \triangleright 3 \triangleright 1 \quad = wr_1m_1nr_2wm_2m_1n$
- $1 \triangleright 6 \triangleright 8 \triangleright 15 \triangleright 7 \triangleright 3 \triangleright 11 \triangleright 9 \triangleright 1 \quad = r_1r_2wnm_1wm_2n$
- $1 \triangleright 6 \triangleright 8 \triangleright 15 \triangleright 7 \triangleright 5 \triangleright 13 \triangleright 9 \triangleright 1 \quad = r_1r_2wnm_2wm_1n$

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

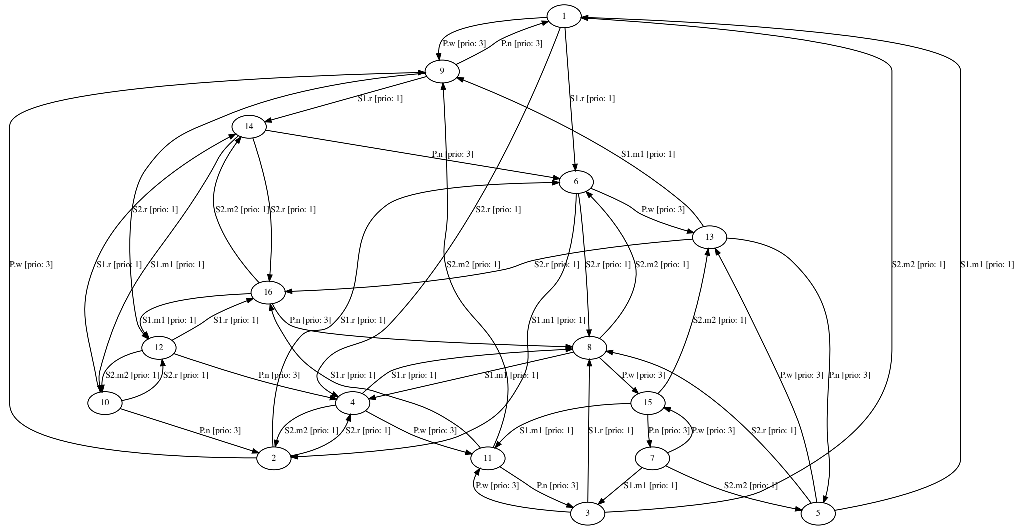


Figure 10.23: Resulting graph of $(P \oplus S_1 \oplus S_2) \odot S$ with adjusted priorities.

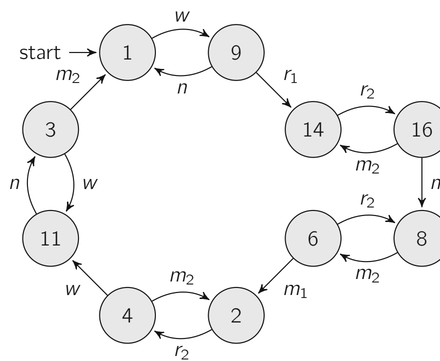


Figure 10.24: The resulting graph of the first path with five loops and straight edge in between.

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Another possibility to adjust the priorities is to focus on particular edges. For example, the edges *w* and *r* get a higher priority, and we reduce the available resource to one CPU. Figure 10.25 shows the new graph. By changing the edge priorities and not the priorities of an entire agent, there is no starvation of the agents, as another can claim the resource next. However, as the agents cannot freely interact, the possible interactions are severely reduced, as visible in the now smaller graph. Some paths still show patterns but not with several loops between the nodes. In addition, no path leads back to the starting node; i.e., the remaining loops are between fewer nodes not connected to the starting node. The system requires a few executions until it reaches a state that creates a stable pattern.

In the case of increasing the resources to two CPUs, the system shows a similar behaviour, requiring a few executions to reach a state space where patterns can form. The respective graph is not shown as there is no more information to gain. Further experiments are required to investigate the effects of various prioritised edges.

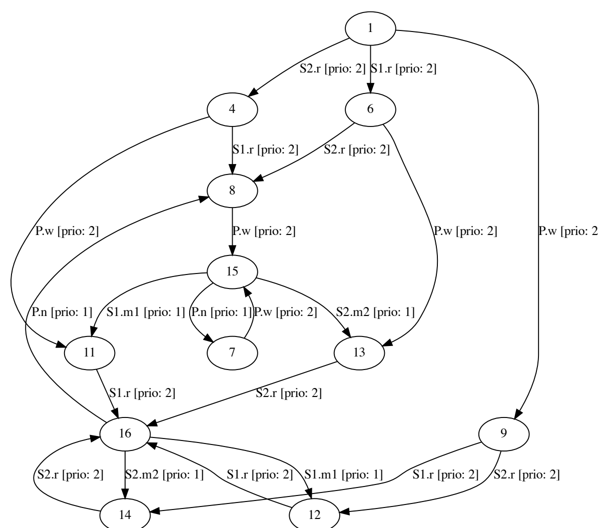


Figure 10.25: Resulting graph of $(P \oplus S_1 \oplus S_2) \odot S$, with specific edge priorities.

10.6 Adding Execution Time

The last open issue is how the execution time of the edge labels affects the pattern formation in such a system. As indicated in Section 9, the complexity increases after considering parallel execution paths. We already defined this issue as out of the scope of this dissertation. Nevertheless, there are first findings that provide the base for future research. A significant effort has been put into building an evaluation setup representing a time-predictable publish-subscribe environment. This setup aims to verify Kronecker Algebra obtained theoretical results. We used widespread industrial middleware (OPC Unified Architecture (OPC UA)) for the publish-subscribe environment and the open-source T-CREST project as a time-predictable platform. Building the testing setup led to several publications, the bases for the following condensed overview.

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

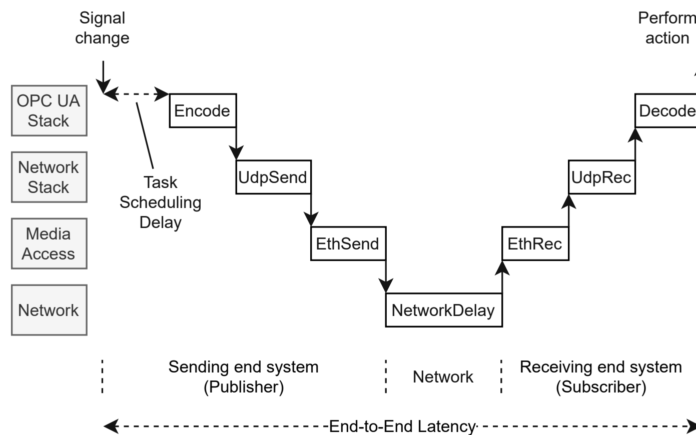


Figure 10.26: OPC UA PubSub end-to-end latency diagram

10.6.1 OPC Unified Architecture

OPC UA is the successor of the Open Platform Communications (OPC) protocol and regarded as one pathfinder to homogenise communication in the industrial domain. A typical view of the OPC UA architecture is one of two pillars [MLD09]. The first represents the Meta Model that enables information modelling. In contrast, the second pillar describes the Transport Mechanisms responsible for encoding data and exchanging messages between devices. OPC UA's primary communication paradigm is Client-Server but it also offers support for Publish-Subscribe (OPC UA PubSub) communication patterns. The Client-Server mechanism can invoke complex services like browsing the information model and calling methods. The newer OPC UA PubSub part minimises the communication overhead and is primarily intended for exchanging process data.

There are several OPC UA software stacks available in a variety of programming languages. A widely used and well-maintained open-source stack is the open62541 project. The open62541 supports most of the OPC UA features, implements the OPC UA PubSub specification, and allows easy porting to other hardware platforms. The stack is implemented in C, which is well-supported by the T-CREST project and other WCET analysis tools.

10.6.2 End-to-end latency

The exchange of an input signal from one device to another via OPC UA PubSub involves numerous delays at the sending end system (i.e., the publisher), the receiving end system (i.e., the subscriber), and the network connection. These delays are illustrated in Figure 10.26 and have to be analysed to obtain a guaranteed upper limit for the end-to-end latency of transmissions.

The analysis starts at the publisher with the change of a signal representing some operational data relevant for another device. In most real-time systems, software tasks are executed at predefined points in time rather than triggered by external events. This concept avoids unpredictable behaviour in high-load scenarios, e.g., if the input signal changes very rapidly. However, it introduces the *TaskSchedulingDelay*, which is the time elapsing between a change of the signal and the subsequent execution of the publisher task. The TaskSchedulingDelay is limited by the task's period, i.e., the time between two consecutive activations. It can be reduced close to zero if the signal represents some internal data value or if changes to the input signal can be synchronised to

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

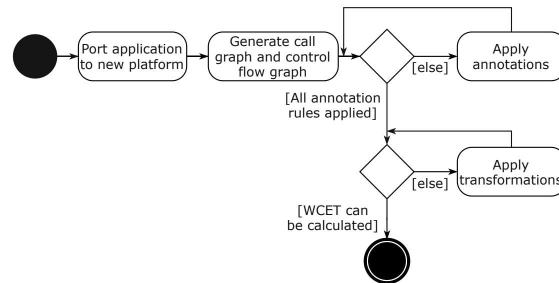


Figure 10.27: Process for adjusting existing software for WCET analysis [DFK+21a]

the publisher task.

Next, during *Encode*, the OPC UA publisher encodes the signal in a message, following the structure defined in Part 6 and Part 14 [OPC18] of the specification. The message is then handed from the OPC UA stack to the network stack, which adds additional information like the UDP, IP, and Ethernet datagram headers (*UdpSend*). All operations performed during Encode and UdpSend are implemented in software. Therefore, upper limits for their execution times can be determined by applying WCET analysis.

The Ethernet frame is then processed and sent over the network by the Ethernet controller, typically implemented in hardware. Therefore, the delay introduced in this step (*EthSend*) is predictable, as long as the Ethernet controller is not occupied with the processing of other messages. This behaviour has already been analysed in other studies [FSS15]. The next timing to be considered is the *NetworkDelay*. If the message shall be transmitted with bounded delays, a real-time Ethernet protocol (e.g., time-sensitive networking (TSN)) is mandatory.

The delays occurring at the subscriber are analogous to the ones introduced at the publisher. They are caused by the *EthRec*, *UdpRec*, and *Decode* functions, whereby only the latter two are implemented in software and need to be analysed for WCET.

10.6.3 WCET Analysis Process

Although existing WCET analysis tools like Absint *aiT* or the *T-CREST* platform offer significant support, determining the WCET cannot be fully automated. In practice, static WCET analysis of existing code that has not been written for real-time applications often requires additional manual work to calculate the WCET. Furthermore, finding reasonable tight bounds to make the code applicable in real-time applications may require considerable effort.

Figure 10.27 illustrates the essential steps to prepare existing program code for WCET analysis in a simple process derived from modern WCET analysis tools. The process is intended to serve as an abstract guide and a helpful starting point for static WCET analysis. It consists of four main steps, which are presented in detail in [DFK+21a]: *(A)* porting the existing code to a time-predictable platform, *(B)* examining the code structure via the call graph and the control-flow graph (CFG), *(C)* applying code annotations, and *(D)* code transformations.

Most importantly, the process steps cover annotation rules to define upper bounds and enable WCET analysis of while loops, do-while loops, for loops, and direct recursions. Furthermore, code transformation rules suggest how to adjust indirect recursions, jump tables, callback functions, and other non-analyzable code for WCET analysis. Applying this process to an OPC UA publisher and subscriber allows determining the WCET for sending and receiving PubSub messages.

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Table 10.1: Programming constructs and number of occurrences for the OPC UA Publisher and Subscriber

Programming construct	Number of occurrences	
	Publisher	Subscriber
While loop	1	0
Do-While loop	0	0
For loop	1	11
Indirect recursion	1	3
Jumptable	1	1
Other, non-WCET-analyzable code	6	7

10.6.4 Adjusting the OPC UA Publisher

The OPC UA publisher receives information from the application, encodes the information in an OPC UA PubSub network message, and transmits/publishes the message via a network interface. Thereby, the OPC UA Specification Part 14: PubSub [OPC18] defines the message format. The specification limits the number of data fields in a PubSub message and each data field's length to 2,147,483,647 (max Int32). This value is not suitable for static WCET analysis, and a trade-off between the software stack's flexibility and the desire for a tight WCET bound has to be made.

Therefore, appropriate changes to the open62541 stack were necessary, reducing the maximum number of supported data fields per message. The modified version of the software supports a maximum of two data fields per message with a limited selection of data types. In addition, the application needs to set the open62541 UA_PUBSUB_RT_FIXED_SIZE flag, which defines that the message structure does not change. Furthermore, only data types of fixed size like Boolean, Integer, and Float may be used, but support for variable-length data types like String and Bytestring is removed. These limitations are considered acceptable because additional data values can easily be transmitted in separate messages and real-time-critical data, e.g., sensor values, typically are of fixed size.

The WCET analysis process presented in Section 10.6.3 was applied to the publisher of the open62541 [F. 23] OPC UA stack. Table 10.1 summarises how often each annotation and transformation rule was used. The analysis was conducted with the tools provided by the T-CREST project and yielded a WCET of 18,632 processor cycles. This value corresponds to about 232.9 µs for a processor operating at 80 MHz. A detailed discussion of the evaluation results is presented in [DFK+21a] and the code is available under [DLFS22].

10.6.5 Adjusting the OPC UA Subscriber

The OPC UA subscriber receives PubSub messages, decodes their contents, and hands the data values over to the application. The application can then act upon the received data values, e.g., by performing calculations and setting outputs. The same considerations regarding the number of data fields in a message and the supported data types mentioned for the publisher also apply for the subscriber.

However, the subscriber included in the open62541 stack uses dynamic memory allocation for each received frame and each data field. The standard library implementations that handle memory allocation in C (*malloc*, *free*, and related functions) fall in the category of non-WCET-analysable

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

code. Therefore, these functions had to be re-implemented. The new, WCET-analysable implementation of *malloc* provides only a fixed number of 32 memory blocks with 512 bytes each. Calling *malloc* returns a pointer to the next free memory block and marks it as *in use*. If no free memory block is available, *malloc* causes an out-of-memory error (ENOMEM), which the caller needs to handle. Furthermore, calling *free* releases the memory block corresponding to the address that is passed as a parameter. The new functions share the same method signature with the standard C library and require no additional changes to the remaining code.

Table 10.1 summarises how often each annotation and transformation rule had to be applied for the open62541 subscriber to obtain a WCET analysable implementation. The analysis yielded a WCET of 443,543 processor cycles, corresponding to about 5544.29 µs for a processor operating at 80 MHz.

10.6.6 Evaluation Setup

Figure 10.28 illustrates the evaluation setup. PublishCallback and SubscribeCallback handle publishing and receiving OPC UA PubSub messages via the open62541 stack. Note that the PublishCallback, in addition to Encode and UdpSend, performs some pre-processing and post-processing of its internal data structures. Likewise, the SubscribeCallback requires some extra operations in addition to UdpRec and Decode. The setup includes two Altera DE2-115 development boards featuring Cyclon IV FPGAs. A Patmos time-predictable processor, which is part of the T-CREST project, is instantiated on each FPGA and operating at a frequency of 80 MHz. The OPC UA publisher and subscriber are executed on these platforms. Furthermore, the two FPGA boards are directly connected with a 100 Mbit point-to-point Ethernet connection of 2 m in length. The publisher and subscriber set and reset general purpose input/output (GPIO) pins at relevant positions in their program flow. A Saleae Logic Pro 8 logic analyser logs these events on the GPIOs for the subsequent timing analysis. The evaluation setup exchanges a single Int32 counter value that is incremented every time before publishing a new message.

10.6.7 Execution Time Measurements

The histograms depicted in Figure 10.29 show the distributions of the execution times measured for PublishCallback, SubscribeCallback, encoding (Encode), sending (UdpSend), receiving (UdpRec), and decoding (Decode) 1000 PubSub messages. As the publisher handles encoding and sending messages, the added execution times of Encode and UdpSend must be lower than the theoretical WCET obtained in Section 10.6.4. The longest execution time for Encode + UdpSend recorded by the logic analyzer is 136.852 µs. Therefore, the WCET bound is approximately 70 % higher than the execution time obtained by the measurements.

Similarly, the subscriber receives and decodes messages via the UdpRec and Decode functions, respectively. Again, the combined execution time of these two functions is below the theoretical upper bound obtained via the WCET analysis in Section 10.6.5. The longest execution time measured is 513.08 µs. Therefore, the WCET bound is approximately 980 % higher than the execution time obtained by the measurements. The over-estimation in the case of the subscriber is worse than for the publisher because of its higher complexity, particularly regarding the challenges arising from dynamic memory allocation.[2]

[2] This issue is addressed in [LDFK23].

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

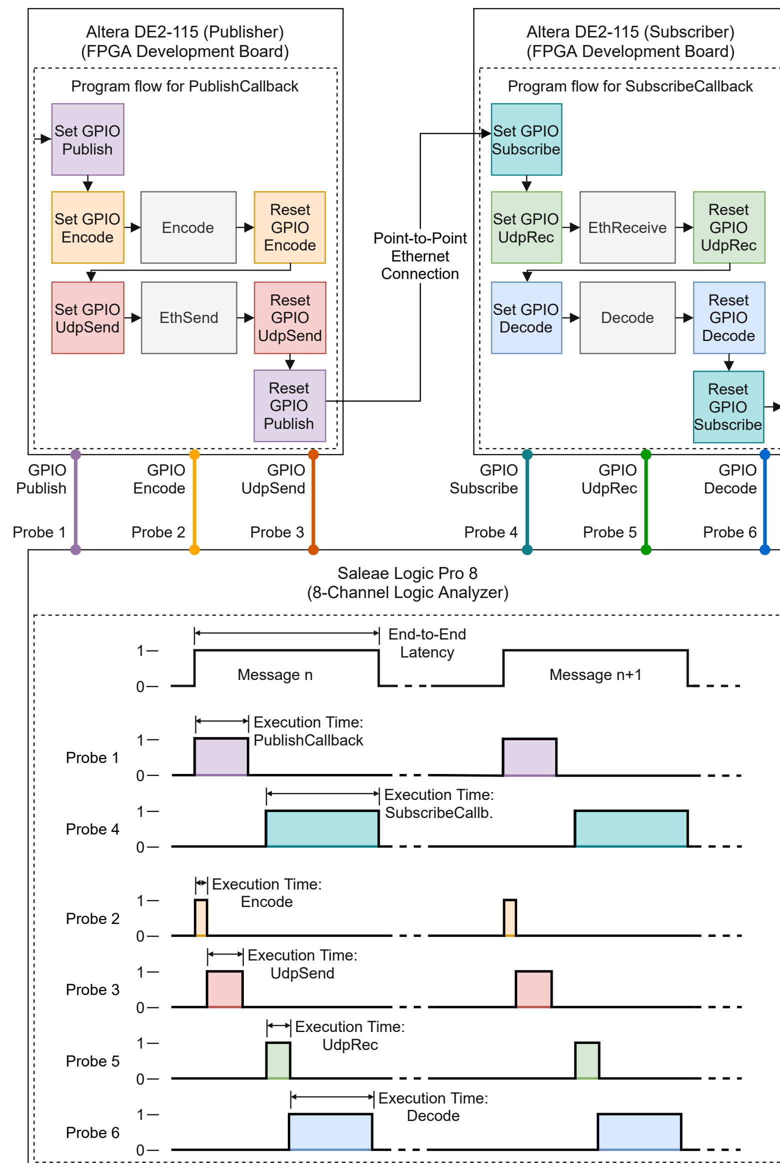


Figure 10.28: Evaluation Setup

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

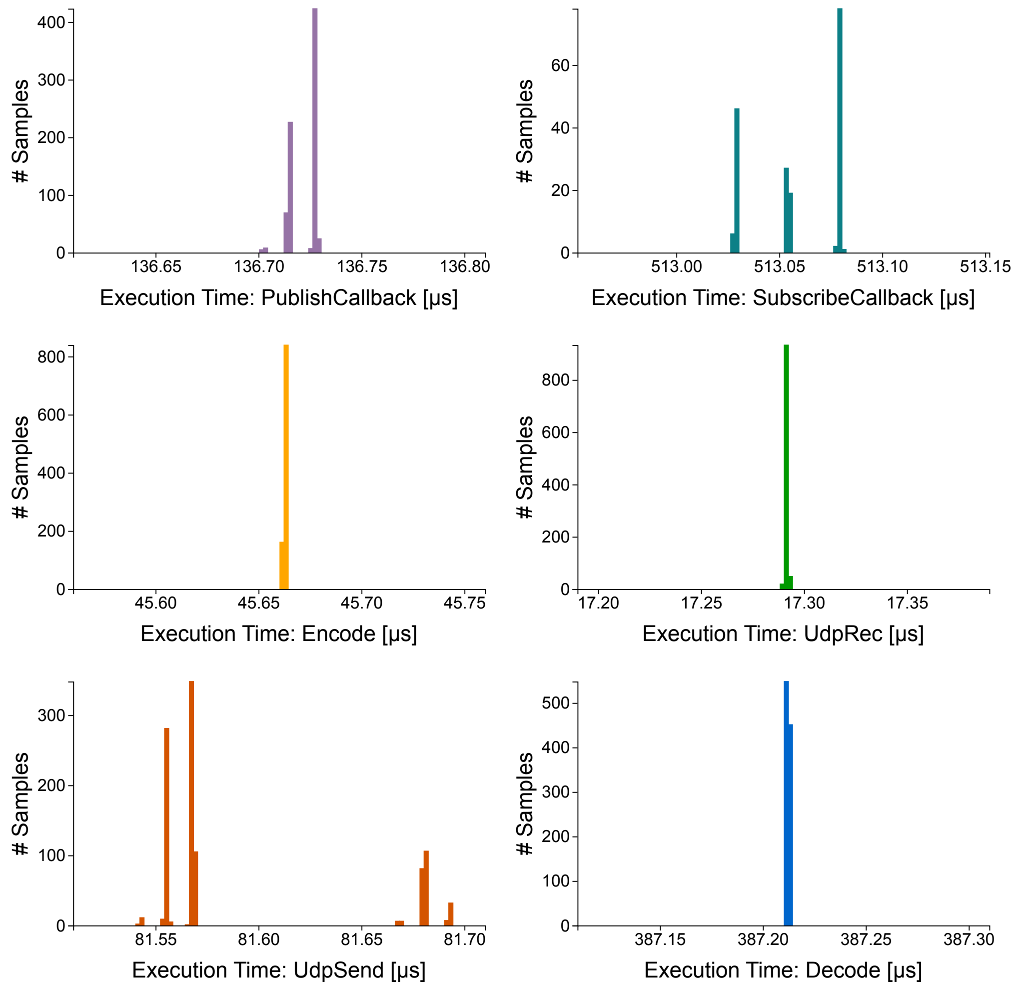


Figure 10.29: Histograms of measured execution times for PublishCallback, Encode, UdpSend, SubscribeCallback, UdpReceive and Decode.

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Table 10.2: Comparison of highest timing measurement results and theoretical upper bounds

SW/HW component	Upper bound	Measurement result
TaskSchedulingDelay	not relevant	not relevant
PublishCallback	232.9 µs	136.85 µs
▷ Encode	▷ -	▷ 45.67 µs
▷ UdpSend	▷ -	▷ 81.69 µs
▷ Pre- and postprocessing	▷ -	▷ 9.49 µs
EthSend + NetworkDelay + EthRec	HW-dependent	15.4 µs
SubscribeCallback	5544.29 µs	513.08 µs
▷ UdpRec	▷ -	▷ 17.29 µs
▷ Decode	▷ -	▷ 387.22 µs
▷ Pre- and postprocessing	▷ -	▷ 108.57 µs
End-To-End Latency	5777.19 µs + NetworkDelay	642.32 µs

10.6.8 End-to-End Latency Analysis

The measurements presented so far only verify the results of the two WCET analyses. However, as shown in Section 10.6.2, the OPC UA publisher and subscriber cause only a part of the end-to-end latency. Therefore, Table 10.2 sums up all the involved software and hardware delays. The first value TaskSchedulingDelay is not included in this analysis because the counter value used as a payload is only incremented right before triggering the Encode task (cf. Figure 10.26). The next entries in the table represent the theoretical WCET bounds and the longest execution times measured for Encode, UdpSend, UdpRec, and Decode, which have already been discussed. The remaining entries EthSend, NetworkDelay, and EthRec are specific to the evaluation platform and are briefly discussed in the following.

Sending an Ethernet frame using the Altera DE2-115 evaluation platform involves two distinct hardware components: Ethernet MAC and Ethernet PHY [PTLB15]. The Ethernet MAC functionality is implemented in the FPGA, while the Ethernet PHY uses a dedicated chip (Marvell 88E1111). The total delay caused by these components for transmitting a frame is subsumed as EthSend. At the subscriber, the time required for receiving an Ethernet frame is subsumed as EthRec. Furthermore, the NetworkDelay of the point-to-point Ethernet connection is in the order of nanoseconds and, therefore, neglectable. The longest total duration of EthSend + NetworkDelay + EthReceive was measured at 15.4 µs. As all of these components are implemented in hardware, the jitter observed is minimal, and the theoretical limit for this delay is set equal to the measurement result.

With an assumed NetworkDelay of $\leq$20 µs, the guaranteed upper bound for publishing, transmitting, and receiving a signal via OPC UA PubSub results in 5797.19 µs. The longest observed end-to-end latency is 642.32 µs and, therefore, within the expected bound. The measurement results for the end-to-end latency and the NetworkDelay are depicted in Figures 10.30 and 10.31, respectively.

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

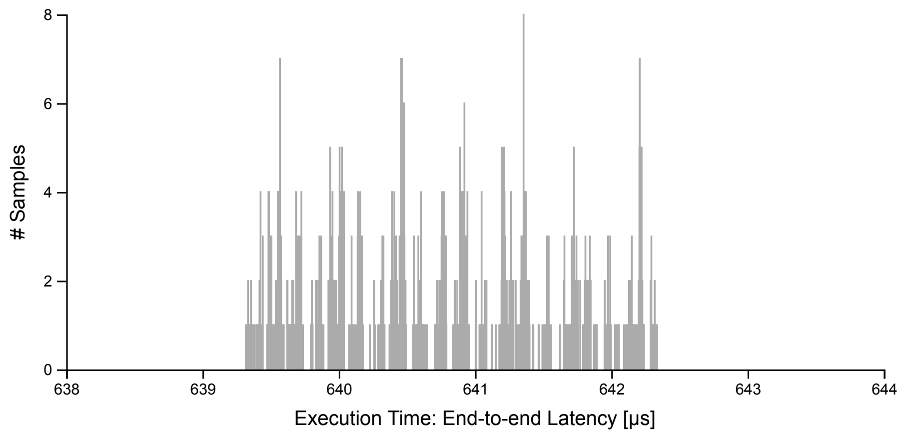


Figure 10.30: Histogram of measured end-to-end latency

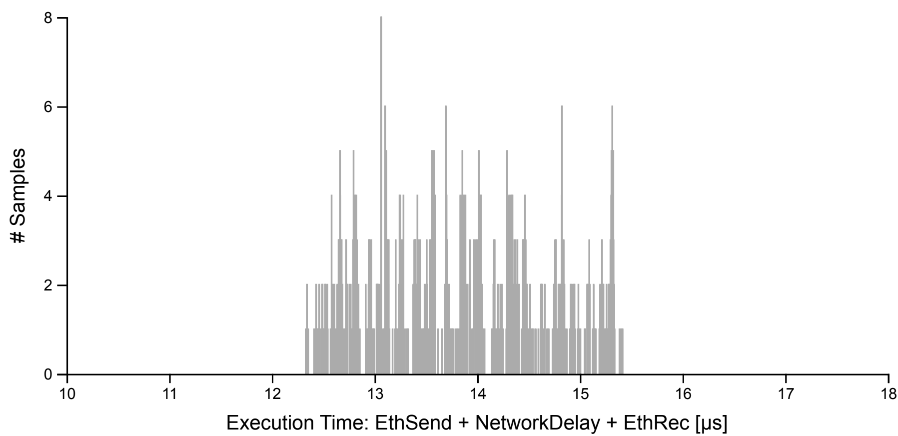


Figure 10.31: Histogram of measured NetworkDelay including hardware delays

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

10.6.9 Further Work

Aside from the basic adjustments of the publisher and subscriber of the OPC stack and collecting WCET and measurement timings, we added additional functionalities. The stack was adjusted in [DFS+22] to support TSN, and the effect of different message payloads was explored. In the paper, [LDFK23], we introduced a worst-case time predictable buffer management into the Patmos Platform. At the same time, we evaluated concurrent information model data access methods in [DAF+22]. Further activities include integrating a TCP/IP stack to test the publish-subscribe environment in a not isolated setting.

10.6.10 Connection to Kronecker Algebra

In Chapter 9, we presented the effects of priorities on the concurrently executing state machines and a simple way, to sum up, the timings to receive a WCET value of the entire execution path. If we apply the same simple approach to the previous examples, we can add to each label a value and calculate the execution time. For example, let us add the obtained WCET values to the simple setup from Section 10.3. For edge n, we use the time required to encode the message w; we use the times the publisher requires to publish the value to the network. In edge r, we include the time until the message reaches the subscriber and m_1 the message decoding. Figure 10.32, shows the allocations. For this reason, an entire cycle equals the end-to-end latency of the measurement setup. As each label can carry some additional information, we can easily calculate the different timings of the paths.

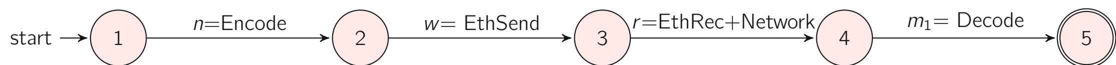


Figure 10.32: End-to-end latency of the simple setup from Section 10.3 with one publisher and one subscriber

However, being able to calculate the execution time of a path alone will not influence the pattern formation. Let us do the following thought experiment. We have two processes A and B (cf. Figure 10.33), both execute first a label (a,b) and then try to catch a shared resource p/v before issuing another label. It is evident that if the execution time of label a is shorter than b, process A will get the shared resource first. Such a time constraint would act similarly to the introduced priorities and face the same issues in a concurrently executing agent system. Nevertheless, there is another issue with execution times; as we saw, they are not constant and fluctuate. (Note: If we use WCET values, we can argue that we focus on finding the longest path). Therefore, a slight time variability in label a can make a difference to if process A or B gets the shared resource and influences the pattern formation processes. In research languages, LinguaFranka and Rebeca the analysis of timing effects on distributed systems is an essential part and a future interest in extending into Kronecker operations.

Another issue that execution time brings up is whether a process or task might be ready to execute. When introducing the priorities, we assumed the processes are ready to execute and must wait for the CPU to become available. However, a different scenario cannot be handled with Kronecker as it produces all execution possibilities. For this reason, another future extension has to introduce path probabilities where each edge has a probability. Based on that probability, a path can become more feasible than others and contribute to identifying the likelyhood of a pattern formation. The execution time will be essential when assigning such a probability to an edge label.

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

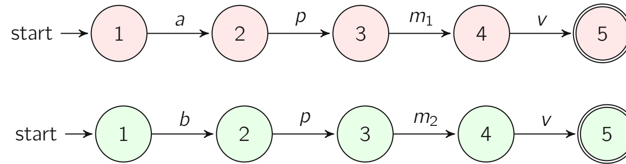


Figure 10.33: The two processes A and B

10.7 Time Complexity of Kronecker Operations

Before heading over to discuss the findings of this Chapter in relation to the research questions in Chapter 1, there is a need to shortly indicate how performant Kronecker operations are. As performance measures strongly depend on the used hardware, the time complexity is more suitable. In addition, Kronecker operations can be further optimised, affecting the performance results. Other researchers have applied Kronecker operations to significant scheduling problems (i.e., train systems [Vol14]) or adjusted them for GPU usage [SMP+17].

Lazy evaluation reduces the time complexity $\mathcal{O}(m^2n^2)$ to successor search. In a Kronecker operation that produces a matrix of size m-by-n, a successor search is now $O(pq)$, where the left child's matrix has $p \leq m$ successors and the right child's matrix has $q \leq n$ successors. The overall complexity is, therefore, $O(rpq)$, with $r \leq mn$ reachable nodes. In practice, the memory requirement is reduced to the number of reachable nodes and memory occupied by queued, unprocessed nodes.

As all Kronecker operations in this dissertation can be reduced to $\otimes$ and $+$, it is possible to perform a time complexity estimation. Each operation was examined to identify the operations and variables that affect the execution time of each algorithm. Table 10.3 provides an overview of the time complexities of each operation. A commonality for all operations is that the node IDs of the resulting graphs need to be stored. One ID is represented by l bits; therefore, $l = |G|$ whereby $|G|$ is the size of the resulting matrix. As handling integers $\leq$ 64bit can be done with machine arithmetic, l becomes relevant at $\geq$ 64bit (big integers). Therefore, Table 10.3 shows the time complexity for $\leq$ 64bit and $\geq$ 64bit.

Table 10.3: Time complexity of Kronecker operations

Operation		$\leq$ 64bit	$\geq$ 64bit
Skip ($\odot$)	Successor	$\mathcal{O}(\lvert A_e\rvert)$	$\mathcal{O}(l^2\lvert A_e\rvert)$
	Labels	$\mathcal{O}(\lvert A_e\rvert)$	$\mathcal{O}(l^2\lvert A_e\rvert)$
Sym. Skip ($\bar{\odot}$)	Successor	$\mathcal{O}(n\lvert F_e\rvert)$	$\mathcal{O}(nl^2\lvert F_e\rvert)$
	Labels	$\mathcal{O}(n\lvert F_e\rvert)$	$\mathcal{O}(nl^2\lvert F_e\rvert)$
Synthesize	Successor	$\mathcal{O}(\lvert M_e\rvert + m)$	$\mathcal{O}(l^2\lvert M_e\rvert + m)$
	Labels	$\mathcal{O}(\lvert M_e\rvert + m)$	$\mathcal{O}(l^2\lvert M_e\rvert + m)$

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

The elements relevant to each of the operations are the following: For Kronecker skip $\odot$, $|A_e|$ represents the number of edges present in the program/FSM A. In the operation Kronecker Symmetric Skip $\bar{\odot}$, $n = \{1, \ldots, i\}$ stands for the amount of the to be combined FSMs$_i$. While, $|F_e|$ is the number of edges in the final FSM. For Kronecker Synthesize operation, $|M_e|$ is the number of edges present in the FSM representation of the MSC. In addition, m is the set of dependency matrices D_i. In summary, it can be argued that the time complexity of all Kronecker operations involved is polynomial. The findings have already been verified in Mittermayr [MB16b]. Moreover, Kronecker operations have been used for train simulations with up to 300 000 000 nodes [Vol14].

10.7.1 Optimisations of Kronecker Algebra

As indicated beforehand, it is possible to optimise Kronecker operations. For the presented approach, it is possible to parallelise the successor search. This parallelisation is achievable because the successor identification has a dependency between a node and its successors. For this reason, a worker thread can process per node instead of per successor. Hash tables avoid duplication of work. Further optimisations are attainable based on specific Kronecker operations and matrix sizes. Interested readers are guided to Mittermayr [MB16b] for more optimisation options plus how the state explosion problem is reduced.

10.8 Concluding Remarks

This chapter focused on finding patterns in an MAS respectively an industrial publish-subscribe setup. The simplified publish-subscribe communication allowed the modelling of simple agents that interact with each other. By utilising Kronecker operations previously introduced, we could find simple pattern formations in the presented examples. The next chapter will discuss if those patterns qualify as emergent patterns. Nevertheless, the work done in building a time-predictable publish-subscribe environment based on OPC UA and T-CREST contributed twofold. First, the environment can be used to validate the theoretical findings built upon by Kronecker Algebra. Second, we contributed directly to ongoing research in industrial communication by providing insights into how to build real-time capable machine-to-machine communication based on publish and subscribe.

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Chapter 11

Discussion

The discussion chapter uses the research questions stated in Chapter 1 as guidance to discuss the findings of this dissertation. As each chapter contains a discussion specific to the related topic, this chapter focuses on the overall aim of the dissertation, namely self-organisation and emergent pattern formation in multi-agent system (MAS) and industrial systems.

11.1 Research Question 1

> **RQ1**: How are the various definitions of emergence and self-organisation in the different research disciplines interconnected and express the critical characteristics of these phenomena to identify emergent behaviour in the context of multi-agent systems (MASs)?

The question was answered in detail in Chapters 2 and 4 by the results of the systematic literature review. However, the following is a condensed summary of the findings as they provide the basis for the subsequent questions. The terms self-organisation and emergence describe behaviours of systems that cannot be easily explained by the sum of the behaviour of the system elements [Sha01]. Both phenomena are reoccurring research topics that have fascinated researchers repeatedly since the 1970s. While there are strong connections to complex systems theory [Mit13], the underlying concepts of self-organisation and emergence found their way into different research fields such as robotics, swarm, biology, social sciences or physics [For90, CM95, KMRF+03].

Essentially, both emergence and self-organisation appear in systems that contain several agents or components that interact. The interaction of the agents on the lower entity level creates a high-level system behaviour that shows an information gain or a pattern formation not deduced from the single entities. Good examples of pattern formations are found in biological systems, where simple agents like fish form complex patterns based on very simplistic rules [CDF+20]. While self-organisation and emergence are two phenomena that can exist alone, they are often found in combination [DWH04]. Self-organisation is an enabler for high-level emergent behaviour, while emergence stabilises the systems and makes them more resilient to changes.

The definition of De Wolf describes self-organisation quite well:

> *"Self-organisation is a dynamical and adaptive process where systems acquire and maintain structure themselves, without external control." [DWH04, p.7]*

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

The definition contains the required system's characteristics and qualities for self-organisation to appear [Hey02]. One characteristic is that the system needs to be autonomous without external control. That does not exclude other types of inputs as long they do not contain any control instructions [Hak98]. Adaptability or robustness w.r.t. changes are referred to the system's ability to react to changes and disturbances. The expectation is that a self-organising system can maintain its organisation autonomously during a change [Gol99]. Another neglected aspect is that every system exists in time. Therefore, the increase in order happens over time and not suddenly; it needs to be dynamic as it requires time to adjust to the changes [Sha01].

The definition of emergence points in a similar direction

> *"A system exhibits emergence when there are coherent emergents at the macro-level that dynamically arise from the interactions between the parts at the micro-level. Such emergents are novel w.r.t. the individual parts of the system." [DWH04, p.3]*

De Wolf's definition encompasses several aspects. First, emergence is the process that creates the conceptual term "emergent", which represents properties, behaviour structure or patterns. The second is, as with self-organisation, the importance of micro-macro-levels [Hol98]. The macro-level comprises the whole system, while the micro-level is the viewpoint of the individual entities representing the system's composition. Another characteristic of radical novelty is closely related to the previous one. In essence, it describes the novelty of the global behaviour w.r.t. the individual behaviours at the micro-level [Cru94a, Cru94b]. This means that the micro-level entities do not explicitly represent the global behaviour; by reductionist terms, the macro-level emergents are not reducible to the system's entities. Moreover, emergents have a time component; they arise over time [Gol99] and are relatively robust regarding errors [Ode02a]. The involved agents must actively interact [Ode02a, CDF+20] without any external control that influences the global behaviour of the system [Ode02a]. Other system characteristics encompass "organisational closure" [Hey02], or "coherence" that refers to the parts logical and consistent correlation.

The required characteristics for self-organisation and emergence in MAS and industrial systems do not change. However, not all such systems can show all types of emergence. Depending on how the agents interact with each other and the environment, being either static or evolving over time, a specific type of emergence can appear. For basic emergence, the agents remain static, and the environment does not influence the agents' behaviour. Basic emergence still requires the above-mentioned system's characteristics, yet not all to the same extent. For example, basic emergence focuses more on pattern formation. Therefore characteristics of robustness or radical novelty cannot be applied in the same way as to a system that produces a higher form of emergence.

11.2 Research Question 2

> **RQ2**: How can interacting agents in a MAS be formally represented to enable the detection of emergent patterns within the entire system while preserving the essential characteristics for pattern formation?

One outcome of the systematic literature review was that limited methods and tools could find, predict and assess emergent behaviour in MAS. Moreover, there are no good examples where emergent behaviour conclusively exists. Most of the tools build upon simulation, where the agent interactions are simulated based on given rules [Eps99]. There are methods available such as requirement [CLW92], mathematical [ONC17] or AI based [GSG+09]. Nevertheless, none of the methods ensures the formal criterion and the preservation of the emergent characteristics.

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

Most promising are formal languages to model MAS and emergent behaviour [Kub03]. Formal language theory examines formal language properties and their relation to model computational devices, such as finite state machines (FSMs) or Turing machines. The Chomsky hierarchy of formal languages [Cho56] connects the various languages with different computational devices. A central element of formal languages are grammars which define how to write symbols on a tape guided by rewriting rules. One grammar creates a language and includes nonterminal and terminal symbols. Strings that consist of terminal symbols produced by grammar are called words. Therefore, a language defined by grammar is a set of words and strings.

Combinations of different grammars evolve into a grammar system [Sal73]. The various grammars interact via the tapes they write upon, i.e., each grammar is an individual computational device with a set of rules and symbols [Sal73]. In this way, it is possible to model MAS with specific properties, with grammar systems. The single grammar represents an agent that operates on the tape (environment) and creates events based on their behaviour (rewriting rules) [CVDKP18]. Different grammars are available, each varying in communication type, amount of tapes or component grammars [ST99].

Using different cases of MAS as a framework and formal languages as guidance, we propose as an answer to RQ2 a cooperating array grammar system [DFP95] to identify basic emergence. The idea is that the symbols issued by the cooperating agents (automata) allow the identification of emergence in a MAS while preserving the essential system characteristics required for emergence. If there is a difference between the total system behaviour (written overtime on a blackboard) and the sum of the languages of the agents, basic emergence might be present. In this way, it is possible to observe the micro-macro behaviour of the agents formally. We can extend this idea to more complex types of emergence.

Limitations are that neither evolutionary processes of the agents nor the environment are considered, as well as hierarchical interactions. Moreover, we consider the environment passive and not influencing the agents' behaviours. The presented definition follows ideas from Kubí [Kub03], and Dassow et al. [DFP95]. Nevertheless, the differences are that Kubí [Kub03] focuses on the superimposition of the cooperating agents' languages while we use the summation of the agents' languages. Moreover, we can provide a total system representation by using Kronecker Algebra to compute all agent labels. The work of Dassow et al. [DFP95] is different in the perspective that the authors used their cooperating array grammars to identify patterns in images. Kronecker Algebra as a tool allows the continuation of a formal treatment of basic emergence based on formal languages. Moreover, it represents a significant difference from other research that mainly remained in the formal context or resorted to simulation to identify basic emergence.

11.3 Research Question 3

> **RQ3**: Can the combination of agent interactions (scenarios) with Kronecker Algebra create a system representation retaining the information essential to detect emergent patterns while ensuring consistency?

In Chapter 6, we presented a process that allows the identification of implied scenarios which some authors consider the first step toward identifying emergent systems behaviour [Ash47, DWH04]. An implied scenario results from several agent scenarios (i.e., agent interactions depicted in message sequence charts (MSCs)) combined, creating unexpected behaviour in the total system.

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

The newly introduced Kronecker Algebra Synthesize operation enables transforming MSCs to control-flow graphs (CFGs) while retaining the semantics of the MSC. A particular benefit compared to other approaches is that this model transformation can be automated similarly to all other Kronecker Algebra operations. Therefore, the process might be suitable for other applications, such as system changes or reconfigurations of cyber-physical production systems (CPPSs) at runtime. A limitation, however, is whether a new scenario is desired or harmful, and the resulting root cause search will remain the responsibility of the systems designer. A problem other researchers also point out in their solutions, e.g., Uchitel et al. [UK01, Uch09]. As Moshirpour et al. [MMEF12] reported for their solution, over-generalisation is not present in our process, as we include all intermediate states. A limitation, however, is related to the problem of state explosion.

While we implemented our Kronecker Algebra operation quite efficiently (cf. Section 5.8), there are limits to the number of agents and the involved messages that can be processed. Introducing reduction and filter techniques will be necessary to minimise the state space. Moreover, identifying and removing agents that will not cause new behaviour is a potential solution. Another option is to create subgroups where the boundaries depend on the number of interactions, i.e., agents heavily interacting will be grouped and analysed separately. Each group will be reduced to a new agent interacting with another group. Plateau [Pla85] conducted similar research.

A side contribution is that the process would allow a designer to check if an implemented system fulfils the desired scenarios. This type of problem often occurs in model-checking [AY01] and has been approached by researchers such as Kaufmann et al. [KKP+15, KKP+14] and Graaf and Van Deursen [GvD07]. Nevertheless, most of those approaches utilise satisfiability problem of propositional logic (SAT)-solvers, which require previous manual intervention, normalisation, and transformation. Another issue Moshirpour et al. [MMBH10] reported was the danger of introducing new scenarios during transformation activities. This conformance is an essential criterion for identifying implied scenarios since, in some cases, after scenarios have been laid out in MSCs, the implemented FSMs may no longer conform to the scenarios, therefore, may produce unexpected behaviour.

Chapter 7 presents an approach that ensures consistency between MSCs and FSMs. The novel Kronecker Symmetric Skip operation $\bar{\odot}$ allows combining multiple state machines into one FSM representing the entire system behaviour. Kronecker Synthesize creates a CFG based on a given MSC and allows adopting the underlying semantics. In the last step, Kronecker Skip $\odot$ compares the previous results for consistency. The resulting graph either confirms consistency or provides valuable insight to a programmer when tracing where the problems are occurring. Compared to the work of Graaf and Van Deursen [GvD07], the Kronecker approach requires no manual intervention and can be automated.

A current limitation is that only simple FSMs and MSCs are supported without advanced modelling capabilities that exist in Unified Modeling Language (UML). For example, constructs such as hierarchical or history states in state machines or "alt" and "loop" fragments in MSCs require further research. However, Kronecker Algebra has been used to handle alternatives and loops in worst-case execution time analysis [MB21].

In summary, we can answer RQ3 that the proposed process provides the means to identify implied scenarios, a potential root cause for emergent behaviour and pattern formation in MAS. Moreover, the introduced Kronecker operations ensure that all information essential to detect emergent patterns remains in the created system representation while ensuring consistency.

TU Bibliothek. Your knowledge hub. WIEN
Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.

11.4 Research Question 4

> **RQ4**: How do factors such as interaction priorities between agents, message transmission times and execution times of the agents influence the emergent pattern formation of a MAS?

The Chapters 8 and 9 provided the foundation for answering RQ4. In Chapters 8, we introduced the possibility of prioritising edge labels into Kronecker Algebra by adjusting our implementation. The changes allow each agent to receive a priority level, and the number of resources central processing unit (CPU) can be defined. In other words, the agents will execute either on one CPUs according to their priorities or in parallel on several CPUs. With additional constructs, such as synchronisation between the agents and conditions, it was possible to identify priority inversion between the three processes. In Chapter 9, we used the priorities to calculate the worst-case execution time (WCET) of two agents concurrently executing on one CPU.

The results in Chapter 10 provided further information. We showed that it is possible to utilise Kronecker Algebra to identify pattern formations in simple interacting agents. The used models represent an actual industrial publish-subscribe communication environment. As Kronecker Operations create all possible execution paths of the interacting agents, we could identify new paths that form patterns. The found paths are not part of the sum of languages of the single agents. This condition partly fulfils the idea in Section 4.5, where the entire systems language differs from the sum of the agent languages. For an observer, the system's behaviour cannot be determined by the agents' actions alone. Moreover, the gained system representation fulfils most of the requirements for emergence.

The first is that the agents are interacting [CDF+20]. If there is no interaction, the agents will execute in parallel, but no pattern will arise that cannot be reduced to one of the agent's languages. Another requirement is the absence of centralised control [Ode02a] that tells the agents what to do. While we designed the agents to behave that way, there is no control during the execution. Moreover, if we consider the found patterns in the matrix as a higher-level system behaviour, the possibility exists that the system shows a micro-macro effect [Gol99]. Nevertheless, if the system representation fulfils the requirement of autonomy [Sha01] and adaptability or robustness w.r.t. changes [Gol99] cannot be confirmed. The reason is that the examples only contain a maximum of two agents; if one fails, the other will not continue to interact. As Kronecker Algebra provides us with all possible paths at once, it is impossible to remove an agent during execution.

One fascinating factor is that only a limited amount of agents interact with each other and barely have any rules on how to do that. That aligns with organisms that use relatively simple behavioural rules to generate structures and patterns on the global system level. Those structures are more complex than the components and processes they emerge [Pag88].

When applying priorities to the examples, the effects can be either reducing all interactions to zero or creating a new system representation with fewer paths and different patterns. We did not execute a series of experiments; therefore, we cannot indicate how agent priorities affect the pattern formation process. However, a first tendency is that the priorities reduce the number of paths but simultaneously extend the remaining ones. The pattern remains in some cases, while in others, they change form. Future work needs to include different systems that show patterns and examine the effects of priorities.

The effects of execution time on pattern formation are similar. It was possible to calculate the execution times of paths in the system graph but only very rudimentary. We obtained the times from a real-time predictable publish-subscribe environment. The first indicators show that the execution time of an edge will influence the pattern formation process, especially when conditions are involved.

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

In other words, if an execution time determines which path is executed next. We plan to continue this course further in combination with edge probabilities.

In summary, we can answer RQ4 only inconclusively. The first results show that Kronecker operations can be utilised to find patterns in interacting agent systems. However, to which extent priorities, execution and message transmission times between the agents affect the formation process cannot be decided, yet. First indications show that those factors influence the formation process, but to what extent would require further research.

11.5 Limitations and Related Work

The presented work in this dissertation has limitations, some already discussed in the individual chapters (e.g., scenario synthesis methods and model checking). An overall limitation and a positive aspect is the absence of time in Kronecker operations. The possibility of creating a representation of all possible execution paths between interacting agents sets this approach apart from other research. Especially methods involving simulation cannot guarantee that all paths are found. Moreover, our approach requires modelling the system in state machines. Therefore, it is impossible to analyse "code", like in some model checkers. In addition, our approach can only find patterns, but deciding if those patterns are emergent was out of scope and requires further research.

The authors in Moshirpour et al. [MMBH10] present an approach to find implied scenarios but suffer from over-generalisation, which requires an additional algorithm to prevent it [MMEF12]. Compared to our approach, we benefit from higher efficiency in handling everything in Kronecker Algebra. Fard et al. [Hen13] builds upon the findings of Moshirpour et al. by storing message labels into interaction matrices. The matrices contain the component name, the sender, and the receiver identifier (ID) and what are the time dependencies. In some sense, similar to our matrices, however, the authors use a Markov chain to model the system's behaviour and identify paths among components of the scenarios. Time dependencies benefit their approach, but it misses the simplicity scalability of Kronecker operations. Research in model-checking is a large field concerned indirectly with finding unintended behaviour [AY01]; we discussed the differences in Chapter 6.

Comparing methods based on mathematical and statistical techniques to identify systems' emergence is more complicated. The main reason is that such methods focus on one specific set of emergent behaviour. On the one hand, what makes them very accurate, replicable and able to handle complex forms or emergence, yet very limited. O'Toole, Nallur, and Clarke [ONC17] utilise a form of distributed consensus, where several agents detect and decide whether a systems change is an emergent event. In another paper, the same authors [ONC14] use the system feedback from the macro and micro levels of the components to determine emergent behaviour. Both approaches collect several variables that are later statistically analysed for correlations. The main difference to our work is that those specific methods can identify higher levels of emergence. This benefit, however, limits their applicability and requires more effort when changes are required.

Another approach proposes the use of Semi-Boolean algebra. In Haglich et al. [HRP10, HPR10], the authors use Semi-Boolean algebra to identify and predict emergent or self-organising behaviours in extended social networks, such as money laundering or smuggling networks. The authors claim to be able to handle large social networks, which is a benefit, yet we have not explored the full capabilities of Kronecker Algebra. Similarly, Chen et al. [CCN10] propose a formal approach to characterise and examine emergent behaviours in complex agent-based simulations. The authors characterise various behaviours according to different abstraction levels, which allows the examination of relations between the levels, i.e., between higher-level behaviours and lower-level events. A clear benefit here is the possibility to introduce abstraction levels, which is currently not possible

TU Bibliothek Your knowledge hub WIEN
Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.

in our approach. The advantage of our approach is simplicity, as we do not require a complex framework that uses an X-machine similar to Petri nets and state diagrams.

So far, approaches based on artificial intelligence techniques have only focused on techniques such as clustering and machine learning. Those approaches observe and analyse the system and find interaction patterns, a completely different idea from our Kronecker Algebra approach. Examples are: Grossman et al. [GSG+09], analyses Internet Protocol (IP) data packets or Denzinger et al. [DK06], which use evolutionary algorithms to identify unwanted emergent behaviours in multi-agent systems. In Villani et al. [VFB+13], the authors look for dynamic structures of emergence in dynamic networks, or Gomez et al. [GSZ17] introduce a quantitative definition of emergence, where subsystems of a complex system are observed. We see the potential to use artificial intelligence techniques to identify patterns in large graphs produced by our operations. A task that so far lies in the hand of the system designer.

11.6 Reliability, Reproducibility and Generalisation

The reliability of the presented results in this dissertation is supported by the chosen research design presented in Chapter 3. Each chapter contributes knowledge that is individually evaluated, discussed, and in some cases, peer-reviewed in contributing publications. In addition, the chapters contribute to the overall evaluation, which provides the knowledge required for answering the research questions and fulfilling the aim of the dissertation. The theoretical background is built upon a systematic literature review which further strengthens the reliability of the results.

Regarding the reproducibility of the results, all newly introduced Kronecker operations are explained and derived from existing operations. All evaluations are explained in detail and can be redone by other researchers. In the case of the time-predicable publish-subscribe environment, the necessary code is available open source in a git repository. The same applies to the systematic literature review, as Chapter 3 explains.

The generalisation of the results is limited, mainly because the focus of this dissertation was to evaluate if Kronecker Algebra can be used to identify patterns in MAS. We did not execute large sets of experiments to determine the causes for the described patterns nor what agent interactions cause those. The absence of more extensive experiments is because the current implementation requires programming each example by hand and is, therefore, unsuitable to run extensive test runs as in simulations. Another issue is that systems develop large state spaces quickly, and the patterns are more difficult to identify without other tools, which would enter a field of pattern recognition far beyond this work's scope.

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Chapter 12

Conclusion

This dissertation presents findings that modern multi-agent system (MAS) and industrial systems can show emergent behaviours or patterns of a higher information level not reducible to the single components or agents. A systematic literature review led to the identification of system properties required for emergent behaviour or pattern formation. The relevant characteristics are dynamically interacting agents that, without external control, create a robust behaviour or pattern that is novel w.r.t. the individual parts of the system over time. Furthermore, the review results revealed inadequate tools and methods to identify such patterns in MASs. Using different MAS types as a framework and a cooperating array grammar system based on formal languages made it possible to build a bridge between pattern formation in MAS and Kronecker Algebra.

Kronecker Algebra manipulates matrices, representing state machines capable of executing formal language grammars. In addition, two newly introduced Kronecker operations, Kronecker Synthesise and Kronecker Symmetric Skip, enable agent scenario synthesis to identify implied scenarios in an overall system representation while ensuring consistency during transformation. Furthermore, by introducing execution priorities into Kronecker operations, it was possible to identify priority inversions between executing processes that share a common resource. The priority functionality was further used to calculate the worst-case execution time (WCET) of two state machines executing on a single-core central processing unit (CPU).

All newly introduced Kronecker operations were applied to a model representing an industrial publish-subscribe communication system. The resulting system representation shows pattern formations that individual agents cannot execute and fulfil the system properties required for emergent behaviour. Moreover, using priorities in agent interactions affects the pattern formation within the total system. Similar effects are visible when looking at the execution time of agent interactions. First experiments with a specific built time-predictable real-time capable publish-subscribe communication environment confirm the findings.

The overall results fulfil the aim of this dissertation to explore the applicability of Kronecker Algebra to identify emergent patterns in MAS or industrial communication systems. Moreover, it confirms the suitability of Kronecker Algebra to identify pattern formations in MASs or industrial communication systems. Limitations are that no extensive experiments were carried out, and the identified patterns are not checked to determine whether they can generate emergent system behaviour.

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

12.1 Future Work

Future work will focus on extending Kronecker Algebra operations. The first step is realising a WCET analysis of concurrently executing state machines with different priorities on a multi-core CPU. The findings will support the implementation of probabilities, which path is more likely to be executed in a system graph.

Another extension includes hierarchical abstraction and clustering of state-machine interactions in larger systems. Moreover, we plan to check the relation between Kronecker Algebra and linear time model checking.

The implementation of the lazy algorithm will be renewed to include large integers and other optimisations to increase the overall efficiency. Further activities are to search for new application fields for Kronecker Algebra, such as production processes or robot route planning.

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Chapter 13

Bibliography

[Abs21] AbsInt, *ait*, Available at https://www.absint.com/ait/, 2021.

[And72] Philip W. Anderson, *More Is Different: Broken symmetry and the nature of the hierarchical structure of science.*, Science **177** (1972), no. 4047, 393–396.

[Art90] Wallace Arthur, *Green Machine: Ecology and the Balance of Nature*, 1 ed., Blackwell Pub, Oxford, 1990.

[Ash47] W. Ross Ashby, *Principles of the Self-Organizing Dynamic System*, The Journal of General Psychology **37** (1947), no. 2, 125–128, PMID: 20270223.

[Ash56] ______, *An Introduction to Cybernetics*, 2 ed., Chapman & Hall, London, 1956.

[Ash62] ______, *Principles of the self-organizing system*, Principles of Self-Organization: Transactions of the University of Illinois Symposium (H. Von Foerster and G. W. Zopf Jr, eds.), Pergamon Press, London, 1962, pp. 255–278.

[ASU86] Alfred V. Aho, Ravi Sethi, and Jeffrey D. Ullman, *Compilers: Principles, Techniques, and Tools*, Addison-Wesley Publishing Co., 1986.

[AY01] Rajeev Alur and Mihalis Yannakakis, *Model Checking of Hierarchical State Machines*, ACM Trans. Program. Lang. Syst. **23** (2001), no. 3, 273–303.

[Baa94] Nils A. Baas, *Emergence, Hierarchies, and Hyperstructures*, Artificial Life III, Santa Fe Studies in the Sciences of Complexity, Proc. (Redwood City, Calif) (C. G. Langton, ed.), vol. XVII, Addison-Wesley, 1994, pp. 515–537.

[Bak10] Alan Baker, *Simulation-Based Definitions of Emergence*, Journal of Artificial Societies and Social Simulation 13 (JASSS) **13** (2010), no. 1, 9.

[Bal01] Philip Ball, *The Self-Made Tapestry: Pattern Formation in Nature*, Oxford University Press, 2001.

[Bar76] William W. Bartley, *The Philosophy of Karl Popper*, Philosophia **6** (1976), no. 3, 463–494.

[BB14] Bernd Burgstaller and Johann Blieberger, *Kronecker Algebra for Static Analysis of Ada Programs with Protected Objects*, Reliable Software Technologies – Ada-Europe

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

2014 (Cham) (Laurent George and Tullio Vardanega, eds.), Springer International Publishing, 2014, pp. 27–42.

[BCFV02] Guido Boffetta, Massimo Cencini, Massimo Falcioni, and Angelo Vulpiani, *Predictability: a way to characterize complexity*, Physics Reports **356** (2002), no. 6, 367–474.

[BD97] Eric Bonabeau and Jean-Louis Dessalles, *Detection and emergence*, La revue de l'Association pour la Recherche sur les sciences de la Cognition (ARCo) **2** (1997), no. 25, 85–94.

[BDG95a] Eric Bonabeau, Jean-Louis Dessalles, and Alain Grumbach, *Characterizing emergent phenomena (1): A critical review*, Revue Internationale de Systémique **9** (1995), 327–346.

[BDG95b] ———, *Characterizing emergent phenomena (2): A critical review*, Revue Internationale de Systémique **9** (1995), 347–371.

[BDM02] Simona Bernardi, Susanna Donatelli, and José Merseguer, *From UML Sequence Diagrams and Statecharts to Analysable Petri Net Models*, Proceedings of the 3rd International Workshop on Software and Performance (New York, NY, USA), WOSP '02, Association for Computing Machinery, 2002, pp. 35–45.

[BDTT99] Eric Bonabeau, Marco Dorigo, Guy Theraulaz, and Guy Theraulaz, *Swarm Intelligence: From Natural to Artificial Systems*, no. 1, Oxford University Press, 1999.

[Bed03] Mark Bedau, *DOWNWARD CAUSATION AND THE AUTONOMY OF WEAK EMERGENCE*, Principia **6** (2003), no. 1, 5–50.

[BEG+12a] Petra Brosch, Uwe Egly, Sebastian Gabmeyer, Gerti Kappel, Martina Seidl, Hans Tompits, Magdalena Widl, and Manuel Wimmer, *Towards Scenario-Based Testing of UML Diagrams*, Tests and Proofs (Berlin, Heidelberg) (Achim D. Brucker and Jacques Julliand, eds.), Springer Berlin Heidelberg, 2012, pp. 149–155.

[BEG+12b] ———, *Towards Semantics-Aware Merge Support in Optimistic Model Versioning*, Models in Software Engineering (Berlin, Heidelberg) (Jörg Kienzle, ed.), Springer Berlin Heidelberg, 2012, pp. 246–256.

[Bel97] Richard Bellman, *Introduction to matrix analysis*, 2nd ed., Classics in Applied Mathematics., Society for Industrial and Applied Mathematics (SIAM), 1997.

[BEL10] John J. Bartholdi, Donald D. Eisenstein, and Yun Fong Lim, *Self-organizing logistics systems*, Annual Reviews in Control **34** (2010), no. 1, 111–117.

[Ben86] Charles H. Bennett, *On The Nature And Origin of Complexity in Discrete, Homogeneous, Locally-Interacting Systems*, Foundations of Physics **16** (1986), no. 6, 585–592.

[Ber03] Alexander U. Berezhnoy, *Emergent Behavior in Multiagent Systems*, Proceedings of the 3rd Winona Computer Science Undergraduate Research Symposium, vol. 4, 2003, pp. 50–55.

[BG07] Fabio Boschetti and Randall Gray, *Emergence and computability*, Emergence: Complexity and Organization (E:CO) **9** (2007), no. 1, 120–130, ISCE Publishing 2007;.

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

[BGMK07] Puneet Bhateja, Paul Gastin, Madhavan Mukund, and K. Narayan Kumar, *Local Testing of Message Sequence Charts Is Difficult*, Fundamentals of Computation Theory (Berlin, Heidelberg) (Erzsébet Csuhaj-Varjú and Zoltán Ésik, eds.), Springer Berlin Heidelberg, 2007, pp. 76–87.

[BH09] Iztok Lebar Bajec and Frank H. Heppner, *Organized flight in birds*, Animal Behaviour **78** (2009), no. 4, 777–789.

[BK02] Peter Buchholz and Peter Kemper, *Efficient Computation and Representation of Large Reachability Sets for Composed Automata*, Discrete Event Dynamic Systems **12** (2002), no. 3, 265–286.

[BKB+07] Pearl Brereton, Barbara A. Kitchenham, David Budgen, Mark Turner, and Mohamed Khalil, *Lessons from applying the systematic literature review process within the software engineering domain*, Journal of Systems and Software **80** (2007), no. 4, 571–583, Software Performance.

[Bli02] Johann Blieberger, *Data-Flow Frameworks for Worst-Case Execution Time Analysis*, Real-Time Systems **22** (2002), no. 3, 183–227.

[Boo94] Grady Booch, *OBJECT-ORIENTED ANALYSIS AND DESIGN With applications*, 2nd ed., Addison-Wesley Longman Publishing Co., Inc., 1994.

[BP23] Mohammadreza Barzegaran and Paul Pop, *The fora european training network on fog computing for robotics and industrial automation*, 2023 Design, Automation & Test in Europe Conference & Exhibition (DATE), 2023, pp. 1–6.

[BPMG05] Fabio Boschetti, Mikhail Prokopenko, Ian Macreadie, and Anne-Marie Grisogono, *Defining and detecting emergence in complex networks*, Knowledge-Based Intelligent Information and Engineering Systems (Berlin, Heidelberg) (Rajiv Khosla, Robert J. Howlett, and Lakhmi C. Jain, eds.), Springer Berlin Heidelberg, 2005, pp. 573–580.

[Bro95] Rodney A Brooks, *Intelligence Without Reason*, The Artificial Life Route to Artificial Intelligence, Routledge, 1 ed., 1995, pp. 25–81.

[BSB12] Bernd Burgstaller, Bernhard Scholz, and Johann Blieberger, *A symbolic analysis framework for static analysis of imperative programming languages*, Journal of Systems and Software **85** (2012), no. 6, 1418–1439, Special Issue: Agile Development.

[BY97] Yaneer Bar-Yam, *Dynamics Of Complex Systems*, 1 ed., Addison-Wesley, 1997.

[CAL] CALResCo Group, *The Complexity & Artificial Life Research Concept*.

[Car89] Peter Anthony Cariani, *ON THE DESIGN OF DEVICES WITH EMERGENT SEMANTIC FUNCTIONS*, Ph.D. thesis, State University of New York Binghamton, NY, 1989.

[CBdSMP16] Luiz Fernando Carvalho, Sylvio Barbon, Leonardo de Souza Mendes, and Mario Lemes Proença, *Unsupervised learning clustering and self-organized agents applied to help network management*, Expert Systems with Applications **54** (2016), 29–47.

TU Bibliothek WIEN Your knowledge hub
Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.

[CCN10] Chih-Chun Chen, Christopher D. Clack, and Sylvia B. Nagl, *Identifying Multi-Level Emergent Behaviors in Agent-Directed Simulations using Complex Event Type Specifications*, SIMULATION **86** (2010), no. 1, 41–51.

[CDF+20] Scott Camazine, Jean-Louis Deneubourg, Nigel R. Franks, James Sneyd, Guy Theraula, and Eric Bonabeau, *Self-Organization in Biological Systems*, Princeton University Press, 2020.

[CF02] Markus Christen and Laura Rebecca Franklin, *The Concept of Emergence in Complexity Science: Finding Coherence between Theory and Practice*, Proceedings of the Complex Systems Summer School 4, 2002, pp. 1–15.

[CF03] James P. Crutchfield and David P. Feldman, *Regularities unseen, randomness observed: Levels of entropy convergence*, Chaos: An Interdisciplinary Journal of Nonlinear Science **13** (2003), no. 1, 25–54.

[CFR12] Timothy R. Colburn, James H. Fetzer, and Terry L. Rankin (eds.), *PROGRAM VERIFICATION: Fundamental Issues in Computer Science*, vol. 14, Studies in Cognitive Systems, no. 1, Springer Science & Business Media, 2012.

[Cha66] Gregory J. Chaitin, *On the Length of Programs for Computing Finite Binary Sequences*, J. ACM **13** (1966), no. 4, 547–569.

[Cha74] ______, *Information-Theoretic Limitations of Formal Systems*, J. ACM **21** (1974), no. 3, 403–424.

[Cha99] Matthew Chalmers, *Comparing Information Access Approaches*, Journal of the American Society for Information Science **50** (1999), no. 12, 1108–1118.

[Cha03] Gregory J. Chaitin, *The LIMITS of MATHEMATICS: A Course on Information Theory and the Limits of Formal Reasoning*, [4th print.]. ed., Springer Series in Discrete Mathematics and Theoretical Computer Science, Springer, London, 2003.

[Cha11] Wai Kin Victor Chan, *INTERACTION METRIC OF EMERGENT BEHAVIORS IN AGENT-BASED SIMULATION*, Proceedings - Winter Simulation Conference (S. Jain, R.R. Creasey, J. Himmelspach, K.P. White, and M. Fu, eds.), 2011, pp. 357–368.

[Cha15] Rafe Champion, *Reason and Imagination: Some thoughts of Karl Popper and William W Bartley*, 2 ed., CreateSpace Independent Publishing Platform, 2015.

[Che00a] Peter Checkland, *Soft Systems Methodology: A Thirty Year Retrospective*, Systems Research and Behavioral Science **17** (2000), S11–S58.

[Che00b] ______, *The Emergent Properties of SSM in Use: A Symposium by Reflective Practitioners*, Systemic Practice and Action Research **13** (2000), no. 6, 799–823.

[Cho56] Noam Chomsky, *THREE MODELS FOR THE DESCRIPTION OF LANGUAGE*, IRE Transactions on Information Theory **2** (1956), no. 3, 113–124.

[CJGK+18] Edmund M. Clarke Jr, Orna Grumberg, Daniel Kroening, Doron Peled, and Helmut Veith, *Model Checking*, 2 ed., Cyber Physical Systems Series, The MIT Press, 2018.

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

[CLW92] Minder Chen, Yihwa Irene Liou, and E. Sue Weber, *Developing intelligent organizations: A context-based approach to individual and organizational effectiveness*, Journal of Organizational Computing **2** (1992), no. 2, 181–202.

[CM95] James P. Crutchfield and Mitchell Melanie, *The evolution of emergent computation*, Proceedings of the National Academy of Sciences **92** (1995), no. 23, 10742–10746.

[CM99] Gianfranco Ciardo and Andrew S. Miner, *A data structure for the efficient Kronecker solution of GSPNs*, Proceedings 8th International Workshop on Petri Nets and Performance Models (Cat. No.PR00331), 1999, pp. 22–31.

[CN34] Morris R. Cohen and Ernest Nagel, *An Introduction To Logic And Scientific Method*, Harcourt, Brace And Company, New York, NY, USA, 1934.

[CNC07] Chih-Chun Chen, Sylvia B. Nagl, and Christopher D. Clack, *Specifying, Detecting and Analysing Emergent Behaviours in Multi-Level Agent-Based Simulations*, Summer Computer Simulation Conference 2007, SCSC'07, Part of the 2007 Summer Simulation Multiconference, SummerSim'07, vol. 2, ACM: Association for Computing Machinery, 2007, pp. 969–976.

[Cor02] Peter A. Corning, *The Re-emergence of "Emergence": A Venerable Concept in Search of a Theory*, Complexity **7** (2002), no. 6, 18–30.

[Cru94a] James P. Crutchfield, *Is Anything Ever New? Considering Emergence*, Complexity: Metaphors, Models, and Reality (Redwood City) (G. Cowan, D. Pines, and D. Melzner, eds.), SFI Series in the Sciences of Complexity XIX, Addison-Wesley, 1994, pp. 479–497.

[Cru94b] ———, *The calculi of emergence: computation, dynamics and induction*, Physica D: Nonlinear Phenomena **75** (1994), no. 1, 11–54.

[Cru06] Joseph Cruz, *Epistemology*, ch. Epistemology, p. 7, John Wiley & Sons, Ltd, 2006.

[CVDKP18] Erzsébet Csuhaj-Varjú, Jürgen Dassow, Jozef Kelemen, and Gheorghe Păun, *Grammar systems: A grammatical approach to distribution and cooperation*, Routledge, 2018.

[CVKKP97] Erzsébet Csuhaj-Varjú, Jozef Kelemen, Alica Kelemenová, and Gheorghe Păun, *Eco-Grammar Systems: A Grammatical Framework for Studying Lifelike Interactions*, Artificial Life **3** (1997), no. 1, 1–28.

[DADMS15] Francesco Luca De Angelis and Giovanna Di Marzo Serugendo, *A logic language for run time assessment of spatial properties in self-organizing systems*, Proceedings - 2015 IEEE 9th International Conference on Self-Adaptive and Self-Organizing Systems Workshops, SASOW 2015, 2015, pp. 86–91.

[DAF+22] Patrick Denzler, Mohammad Ashjaei, Thomas Frühwirth, Victor Nicholas Ebirim, and Wolfgang Kastner, *Concurrent OPC UA information model access, enabling real-time OPC UA PubSub*, 2022 IEEE 27th International Conference on Emerging Technologies and Factory Automation (ETFA), 2022, pp. 1–4.

[Dar94] Vince Darley, *Emergent Phenomena and Complexity*, Arificial Life **4** (1994.), 411–416.

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

[Dav81] Marc Davio, *Kronecker Products and Shuffle Algebra*, IEEE Transactions on Computers **C-30** (1981), no. 2, 116–125.

[DBK22] Patrick Denzler, Johann Blieberger, and Wolfgang Kastner, *Utilising Kronecker Algebra to Detect Unexpected Behaviour in Distributed Systems*, 2022 IEEE 25th International Symposium On Real-Time Distributed Computing (ISORC), 2022, pp. 1–8.

[DCGBB16] Robert I. Davis, Liliana Cucu-Grosjean, Marko Bertogna, and Alan Burns, *A review of priority assignment in real-time systems*, Journal of Systems Architecture **65** (2016), 64–82.

[Dem98] Mary Beth Linda Dempster, *A Self-organizing Systems Perspective on Planning for Sustainability*, Master's thesis, University of Waterloo, 1998.

[DFK+21a] Patrick Denzler, Thomas Frühwirth, Andreas Kirchberger, Martin Schoeberl, and Wolfgang Kastner, *Experiences from Adjusting Industrial Software for Worst-Case Execution Time Analysis*, 2021 IEEE 24th International Symposium on Real-Time Distributed Computing (ISORC), 2021, pp. 62–70.

[DFK+21b] ______, *Static Timing Analysis of OPC UA PubSub*, 2021 17th IEEE International Conference on Factory Communication Systems (WFCS), 2021, pp. 167–174.

[DFP95] Jürgen Dassow, Rudolf Freund, and Gheorghe Păun, *COOPERATING ARRAY GRAMMAR SYSTEMS*, International Journal of Pattern Recognition and Artificial Intelligence **09** (1995), no. 06, 1029–1053.

[DFS+22] Patrick Denzler, Thomas Frühwirth, Daniel Scheuchenstuhl, Martin Schoeberl, and Wolfgang Kastner, *Timing Analysis of TSN-Enabled OPC UA PubSub*, 2022 IEEE 18th International Conference on Factory Communication Systems (WFCS), 2022, pp. 1–8.

[DHFK21] Patrick Denzler, Siegfried Hollerer, Thomas Frühwirth, and Wolfgang Kastner, *Identification of security threats, safety hazards, and interdependencies in industrial edge computing*, 2021 IEEE/ACM Symposium on Edge Computing (SEC), 2021, pp. 397–402.

[Dij71] Edsger W. Dijkstra, *Hierarchical ordering of sequential processes*, Acta Informatica **1** (1971), no. 2, 115–138.

[Dij01] ______, *Under the spell of Leibniz's dream*, Information Processing Letters **77** (2001), no. 2-4, 53–61, In honor of Edsger W. Dijkstra.

[Dij02] ______, *Cooperating Sequential Processes*, The Origin of Concurrent Programming: From Semaphores to Remote Procedure Calls (New York, NY) (Per Brinch Hansen, ed.), Springer New York, 2002, pp. 65–138.

[Dijna] Edsger W. Dijkstra, *Over de sequentialiteit van procesbeschrijvingen.*, Circulated privately (n/a.), n/a.

[DK06] Jörg Denzinger and Jordan Kidney, *Evaluating Different Genetic Operators in the Testing for Unwanted Emergent Behavior Using Evolutionary Learning of Behavior*, 2006 IEEE/WIC/ACM International Conference on Intelligent Agent Technology, 2006, pp. 23–29.

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU Wien Bibliothek Your knowledge hub

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

[DK23] Patrick Denzler and Wolfgang Kastner, *Reference Architectures for Closing the IT/OT Gap*, Digital Transformation: Core Technologies and Emerging Topics from a Computer Science Perspective (Birgit Vogel-Heuser and Manuel Wimmer, eds.), Springer Berlin Heidelberg, Berlin, Heidelberg, 2023, pp. 95–123.

[DLFS22] Patrick Denzler, Christoph Lehr, Thomas Frühwirth, and Martin Schoeberl, *Source code OPC UA, T-CREST*, https://git.auto.tuwien.ac.at/rt-ua/ (2022), 1.

[DM11] Ralf Der and Georg Martius, *Self-Organization in Nature and Machines*, In: The Playful Machine. Cognitive Systems Monographs, vol. 15, Springer, Berlin, Heidelberg, 2011, pp. 9–21.

[DMLP79] Richard A. De Millo, Richard J. Lipton, and Alan J. Perlis, *Social Processes and Proofs of Theorems and Programs*, Communications of the ACM **22** (1979), no. 5, 271–280.

[DMSGK11] Giovanna Di Marzo Serugendo, Marie-Pierre Gleizes, and Anthony Karageorgos, *Self-organising Systems*, Self-organising Software: From Natural to Artificial Adaptation (Berlin, Heidelberg) (Giovanna Di Marzo Serugendo, Marie-Pierre Gleizes, and Anthony Karageorgos, eds.), Springer Berlin Heidelberg, 2011, pp. 7–32.

[Dog08] Radu Dogaru, *Emergence, Locating and Measuring It*, vol. 95, ch. 4, pp. 47–75, Springer Berlin Heidelberg, Berlin, Heidelberg, 2008.

[DP06] Jean-Louis Dessalles and Denis Phan, *Emergence in Multi-Agent Systems: Cognitive Hierarchy, Detection, and Complexity Reduction part I: Methodological Issues*, Artificial Economics (Berlin, Heidelberg) (M. Beckmann, H. P. Künzi, G. Fandel, W. Trockel, A. Basile, A. Drexl, H. Dawid, K. Inderfurth, W. Kürsten, U. Schittko, Philippe Mathieu, Bruno Beaufils, and Olivier Brandouy, eds.), Springer Berlin Heidelberg, 2006, pp. 147–159.

[DRK+20] Patrick Denzler, Jan Ruh, Marine Kadar, Cosmin Avasalcai, and Wolfgang Kastner, *Towards Consolidating Industrial Use Cases on a Common Fog Computing Platform*, 2020 25th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA), vol. 1, 2020, pp. 172–179.

[DRK21a] Patrick Denzler, Daniel Ramsauer, and Wolfgang Kastner, *Model-driven Engineering of Gateways for Industrial Automation*, Automation, Robotics & Communications for Industry 4.0 (2021), 47.

[DRK21b] ———, *Tunnelling and Mirroring Operational Technology Data with IP-based Middlewares*, 2021 22nd IEEE International Conference on Industrial Technology (ICIT), vol. 1, 2021, pp. 1205–1210.

[DRP+22] Patrick Denzler, Daniel Ramsauer, Thomas Preindl, Wolfgang Kastner, and Alexander Gschnitzer, *Comparing Different Persistent Storage Approaches for Containerized Stateful Applications*, 2022 IEEE 27th International Conference on Emerging Technologies and Factory Automation (ETFA), 2022, pp. 1–8.

[DRPK21] Patrick Denzler, Daniel Ramsauer, Thomas Preindl, and Wolfgang Kastner, *Communication and container reconfiguration for cyber-physical production systems*, 2021 26th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA), 2021, pp. 1–8.

[DSA93] Paolo Dario, Giulio Sandini, and Patrick Aebischer (eds.), *Swarm Intelligence in Cellular Robotic Systems*, Berlin, Heidelberg, Springer Berlin Heidelberg, 1993.

[DSRK21] Patrick Denzler, Daniel Scheuchenstuhl, Daniel Ramsauer, and Wolfgang Kastner, *Modelling protocol gateways for cyber-physical systems using Architecture Analysis & Design Language*, Procedia CIRP **104** (2021), 1339–1344, 54th CIRP CMS 2021 - Towards Digitalized Manufacturing 4.0.

[Dut12] Alain Dutech, *Self-organizing Developmental Reinforcement Learning*, From Animals to Animats 12. SAB 2012. Lecture Notes in Computer Science, (Berlin, Heidelberg.) (Hallam J. (eds) Ziemke T., Balkenius C., ed.), vol. 7426, Springer, 2012.

[DWF11] Clare Dixon, Alan Winfield, and Michael Fisher, *Towards Temporal Verification of Emergent Behaviours in Swarm Robotic Systems*, Towards Autonomous Robotic Systems (Berlin, Heidelberg) (Roderich Groß, Lyuba Alboul, Chris Melhuish, Mark Witkowski, Tony J. Prescott, and Jacques Penders, eds.), Springer Berlin Heidelberg, 2011, pp. 336–347.

[DWH04] Tom De Wolf and Tom Holvoet, *Emergence and Self-Organisation: a statement of similarities and differences*, Proceedings of the International Workshop on Engineering Self-Organising Applications 2004, 2004, pp. 96–110.

[DWH05] ______, *Emergence Versus Self-Organisation: Different Concepts but Promising When Combined*, Engineering Self-Organising Systems (Berlin, Heidelberg) (Sven A. Brueckner, Giovanna Di Marzo Serugendo, Anthony Karageorgos, and Radhika Nagpal, eds.), Springer Berlin Heidelberg, 2005, pp. 1–15.

[DWHS06] Tom De Wolf, Tom Holvoet, and Giovanni Samaey, *Development of Self-organising Emergent Applications with Simulation-Based Numerical Analysis*, Engineering Self-Organising Systems (Berlin, Heidelberg) (Sven A. Brueckner, Giovanna Di Marzo Serugendo, David Hales, and Franco Zambonelli, eds.), Springer Berlin Heidelberg, 2006, pp. 138–152.

[DWSHR05] Tom De Wolf, Giovanni Samaey, Tom Holvoet, and Dirk Roose, *Decentralised Autonomic Computing: Analysing Self-Organising Emergent Behaviour using Advanced Numerical Methods*, Second International Conference on Autonomic Computing (ICAC'05), 2005, pp. 52–63.

[Edm99] Bruce Edmonds, *Syntactic Measures of Complexity.*, Ph.D. thesis, University of Manchester, Manchester, UK., 1999.

[Egy06] Alexander Egyed, *Instant Consistency Checking for the UML*, Proceedings of the 28th International Conference on Software Engineering (New York, NY, USA), ICSE '06, Association for Computing Machinery, 2006, pp. 381–390.

[EHHS02] Gregor Engels, Jan Hendrik Hausmann, Reiko Heckel, and Stefan Sauer, *TESTING THE CONSISTENCY OF DYNAMIC UML DIAGRAMS*, Integrated Design and Process Technology, IDPT-2002 (Printed in the United States of America), Society for Design and Process Science, 06 2002.

[ÉK12] Zoltán Ésik and Werner Kuich, *Modern Automata Theory*, E104 - Institut für Diskrete Mathematik und Geometrie, 2012, http://hdl.handle.net/20.500.12708/20717.

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

[Eps99] Joshua M. Epstein, *Agent-Based Computational Models and Generative Social Science*, Complexity **4** (1999), no. 5, 41–60.

[F. 23] F. Palm et al., *open62541*, Available at https://github.com/open62541 (2023), 1.

[FC03] David P. Feldman and James P. Crutchfield, *Structural information in two-dimensional patterns: Entropy convergence and excess entropy*, Phys. Rev. E **67** (2003), 051104.

[Fel06] Bernard Feltz, *Self-Organization, Selection and Emergence in the Theories of Evolution*, SELF-ORGANIZATION AND EMERGENCE IN LIFE SCIENCES (GOUJON P. (eds) FELTZ B., CROMMELINCK M., ed.), Springer, Dordrecht, 2006, pp. 341–360.

[Fet88] James H. Fetzer, *Program verification: the very idea*, Communications of the ACM **31** (1988), no. 9, 1048–1063.

[Fet99] ______, *THE ROLE OF MODELS IN COMPUTER SCIENCE*, The Monist **82** (1999), no. 1, 20–36.

[FFH99] Henning Fernau, Rudolf Freund, and Markus Holzer, *REGULATED ARRAY GRAMMARS OF FINITE INDEX - Part I: Theoretical Investigations*, Grammatical Models of Multi-Agent Systems, 1999, pp. 284–296.

[FH03] Noria Foukia and Salima Hassas, *Towards self-organizing computer networks: A complex system perspective*, in: G. Di Marzo-Serugendo, A.Karageorgos, O.F. Rana and F. Zambonellini (Eds), proceeding of AAMAS'2003 Workshop on Engineering Self-Organizing Applications, 15 July 2003, Melbourne, Australia. pp 77-83. (Melbourne, Australia), 7 2003.

[FMG14] Nelson Fernández, Carlos Maldonado, and Carlos Gershenson, *Information Measures of Complexity, Emergence, Self-organization, Homeostasis, and Autopoiesis*, Guided Self-Organization: Inception (Berlin, Heidelberg) (Mikhail Prokopenko, ed.), Springer Berlin Heidelberg, 2014, pp. 19–51.

[For90] Stephanie Forrest, *Emergent computation: Self-organizing, collective, and cooperative phenomena in natural and artificial computing networks: Introduction to the proceedings of the ninth annual CNLS conference*, Physica D: Nonlinear Phenomena **42** (1990), no. 1, 1–11.

[FP86] J. Doyne Farmer and Norman H. Packard, *Evolution, games, and learning: Models for adaptation in machines and nature. An introduction to the Proceedings of the CNLS Conference, Los Alamos, May 1985*, Physica D: Nonlinear Phenomena **22** (1986), no. 1, vii–xii, Proceedings of the Fifth Annual International Conference.

[Fre00] Rudolf Freund, *ARRAY GRAMMAR SYSTEMS*, Journal of Automata, Languages and Combinatorics **5** (2000), no. 1, 13–29.

[Fri95] Uriel Frisch, *Turbulence: The Legacy of A. N. Kolmogorov*, Cambridge University Press, 1995.

[FSS15] Thomas Frühwirth, Wilfried Steiner, and Bernhard Stangl, *TTEthernet SW-based End System for AUTOSAR*, Proceedings of the 10th IEEE International Symposium on Industrial Embedded Systems (SIES) (Siegen, Germany), 6 2015, pp. 1–8.

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

[FV03] Thomas Huining Feng and Hans Vangheluwe, *Case study: Consistency problems in a UML model of a chat room*, Workshop on Consistency Problems in UML-based Software Development, 2003, p. 18.

[GBS13] Sebastian Gabmeyer, Petra Brosch, and Martina Seidl, *A Classification of Model Checking-Based Verification Approaches for Software Models*, Second Workshop on Verification Of Model Transformations, VOLT, 2013, pp. 1–7.

[Ger12] Carlos Gershenson, *The World as Evolving Information*, Unifying Themes in Complex Systems VII (Berlin, Heidelberg) (Ali A. Minai, Dan Braha, and Yaneer Bar-Yam, eds.), Springer Berlin Heidelberg, 2012, pp. 100–115.

[GESL06] Jan Gustafsson, Andreas Ermedahl, Christer Sandberg, and Bjorn Lisper, *Automatic Derivation of Loop Bounds and Infeasible Paths for WCET Analysis Using Abstract Execution*, 2006 27th IEEE International Real-Time Systems Symposium (RTSS'06), 2006, pp. 57–66.

[GH05] Carlos Gershenson and Francis Heylighen, *How can we think the complex*, Managing organizational complexity: philosophy, theory and application, vol. 3, Information Age Publishing Greenwich, CT, USA, 2005, pp. 47–62.

[GM81] Mart R. Gross and Anne M. MacMillan, *Predation and the evolution of colonial nesting in bluegill sunfish (Lepomis macrochirus)*, Behavioral Ecology and Sociobiology **8** (1981), no. 3, 163–174.

[GM16] Vahid Garousi and Mika V. Mäntylä, *A systematic literature review of literature reviews in software testing*, Information and Software Technology **80** (2016), 195 – 216.

[Gol99] Jeffrey Goldstein, *Emergence as a Construct: History and Issues*, Emergence, A Journal of Complexity Issues in Organizations and Management, vol. 1, The New England Complex Systems Institute, 1999, pp. 49–72.

[GP71] Paul Glansdorff and Ilya Prigogine, *Thermodynamic Theory of Structure, Stability and Fluctuations*, Journal of Fluid Mechanics, vol. 53, J. Willey & Sons, 1971, p. 400.

[GR08] Ross Gore and Paul F. Reynolds Jr., *Applying Causal Inference to Understand Emergent Behavior*, Proceedings of the 40th Conference on Winter Simulation, WSC '08, Winter Simulation Conference, 2008, pp. 712–721.

[Gra69] Sir Clive William John Granger, *Investigating Causal Relations by Econometric Models and Cross-spectral Methods*, Econometrica **37** (1969), no. 3, 424–438.

[Gra18] Alexander Graham, *Kronecker Products and Matrix Calculus with Applications*, dover edition ed., Courier Dover Publications, New York, 2018.

[GRTB07] Ross Gore, Paul F. Reynolds Jr., Lingjia Tang, and David C. Brogan, *Explanation Exploration: Exploring Emergent Behavior*, 21st International Workshop on Principles of Advanced and Distributed Simulation (PADS'07), 2007, pp. 113–122.

[GS88] John B. Goodenough and Lui R. Sha, *The Priority Ceiling Protocol: A Method for Minimizing the Blocking of High Priority Ada Tasks*, Ada Lett. **VIII** (1988), no. 7, 20–31.

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

[GSG+09] Robert L. Grossman, Michael Sabala, Yunhong Gu, Anushka Anand, Matt Handley, Rajmonda Sulo, and Lee Wilkinson, *Discovering Emergent Behavior From Network Packet Data: Lessons From The Angle Project*, Next Generation of Data Mining (2009), pp. 243–260.

[GSZ17] Sergio Gómez, Eugene Santos, and Yan Zhao, *Automatic Emergence Detection in Complex Systems*, Complexity **2017** (2017), 3460919.

[GvD07] Bas Graaf and Arie van Deursen, *Model-Driven Consistency Checking of Behavioural Specifications*, Fourth International Workshop on Model-Based Methodologies for Pervasive and Embedded Software (MOMPES'07), 2007, pp. 115–126.

[Hak77] Hermann Haken, *Synergetics: An Introduction. Nonequilibrium Phase Transitions and Self-Organization in Physics, Chemistry and Biology*, 1 ed., New York: Springer-Verlag, 1977.

[Hak78] ———, *An Introduction Nonequilibrium Phase Transitions and Self-Organization in Physics, Chemistry and Biology (Second Enlarged Edition)*, Springer-Verlag, New York, 1978.

[Hak84] ———, *The Science of Structure: Synergetics*, Van Nostrand Reinhold Company, 1984.

[Hak98] ———, *Information and Self-Organisation: A Macroscopic Approach to Complex Systems*, Springer Series in Synergetics (SSSYN), Springer Science & Business Media, 1998.

[Ham64] William D. Hamilton, *The genetical evolution of social behaviour. I, II*, Journal of Theoretical Biology **7** (1964), no. 1, 1–52.

[Har18] John Harvey, *The Blessing and Curse of Emergence in Swarm Intelligence Systems*, Foundations of Trusted Autonomy (Cham) (Hussein A. Abbass, Jason Scholz, and Darryn J. Reid, eds.), Springer International Publishing, 2018, pp. 117–124.

[HE12] David A. Harper and Anthony M. Endres, *The anatomy of emergence, with a focus upon capital formation*, Journal of Economic Behavior & Organization **82** (2012), no. 2, 352–367, Emergence in Economics.

[Hen13] Fatemeh Hendijani Fard, *Detecting and fixing emergent behaviors in Distributed Software Systems using a message content independent method*, 2013 28th IEEE/ACM International Conference on Automated Software Engineering (ASE), 11 2013, pp. 746–749.

[Hey89] Francis Heylighen, *Self-organization, Emergence and the Architecture of Complexity*, Proceedings of the 1st European Conference on System Science (Paris: AFCET), vol. 18, 1989, pp. 23–32.

[Hey02] Francis Heylighen, *THE SCIENCE OF SELF-ORGANIZATION AND ADAPTIVITY*, Knowledge Management, Organizational Intelligence and Learning, and Complexity: (L. D. Kiel, ed.), The Encyclopedia of Life Support Systems, vol. 1, EOLSS Publishers Co Ltd [http://www.eolss.net], Oxford, 2002, pp. 1–26.

TU WIEN Bibliothek Your knowledge hub
Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.

[HGB15] Derek R. Harp and Bengt Gregory-Brown, *IT / OT Convergence Bridging the Divide*, Tech. report, NexDefense, 2015.

[HIKer] Richard Hammack, Wilfried Imrich, and Sandi Klavžar, *Handbook of Product Graphs*, 2nd, ed., DISCRETE MATHEMATICS AND ITS APPLICATIONS, CRC Press, Boca Raton, FL, 2011. With a foreword by Peter Winkler.

[HK02] David Harel and Hillel Kugler, *SYNTHESIZING STATE-BASED OBJECT SYSTEMS FROM LSC SPECIFICATIONS*, International Journal of Foundations of Computer Science **13** (2002), no. 01, 5–51.

[HL12] David A. Harper and Paul Lewis, *New perspectives on emergence in economics*, Journal of Economic Behavior & Organization **82** (2012), no. 2, 329–337, Emergence in Economics.

[HM76] Peter Henderson and James H. Morris, Jr., *A Lazy Evaluator*, 3rd ACM Symposium on Principles of Programming Languages, POPL '76, January 1976, pp. 95–103.

[HMPR04] Alan R. Hevner, Salvatore T. March, Jinsoo Park, and Sudha Ram, *Design Science in Information Systems Research*, MIS Quarterly **28** (2004), no. 1, 75–105.

[HMS+20] Ke Huang, Xin Ma, Rui Song, Xuewen Rong, Xincheng Tian, and Yibin Li, *A self-organizing developmental cognitive architecture with interactive reinforcement learning*, Neurocomputing **377** (2020), 269–285.

[HMU06] John E. Hopcroft, Rajeev Motwani, and Jeffrey D. Ullman, *Introduction to Automata Theory, Languages, and Computation (3rd Edition)*, Addison-Wesley Longman Publishing Co., Inc., USA, 2006.

[Hoa69] Charles A. R. Hoare, *An Axiomatic Basis for Computer Programming*, Communications of the ACM **12** (1969), no. 10, 576–580.

[Hoa09] ______, *Viewpoint Retrospective: An Axiomatic Basis for Computer Programming*, Communications of the ACM **52** (2009), no. 10, 30–32.

[Hol75] John Henry Holland, *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence.*, 2 ed., MIT Press, Cambridge, Massachusetts, 1992 (First edition in 1975).

[Hol98] ______, *EMERGENCE: From Chaos to Order*, Oxford University Press, 1998.

[Hor07] Gregory S. Hornby, *Modularity, reuse, and hierarchy: Measuring complexity by measuring structure and organization*, Complexity **13** (2007), no. 2, 50–61.

[HPR10] Peter Haglich, Laura Pullum, and Christopher Rouff, *Detecting Emergent Behaviors with Semi-Boolean Algebra*, AIAA Infotech at Aerospace 2010, 2010.

[HRP10] Peter Haglich, Christopher Rouff, and Laura Pullum, *Detecting Emergence in Social Networks*, 2010 IEEE Second International Conference on Social Computing, 2010, pp. 693–696.

[HT66] Frank Harary and Charles A. Trauth, Jr., *Connectedness of Products of Two Directed Graphs*, SIAM Journal on Applied Mathematics **14** (1966), no. 2, 250–254.

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

[Hur94] Adolf Hurwitz, *Zur Invariantentheorie*, Mathemathische Annalen **45** (1894), 381–404.

[HVZ+04] Karuna Hadeli, Paul Valckenaers, Constantin Zamfirescu, Hendrik Van Brussel, Bart Saint Germain, Tom Hoelvoet, and Elke Steegmans, *Self-organising in multi-agent coordination and control using stigmergy*, Engineering Self-Organising Systems (Berlin, Heidelberg) (Giovanna Di Marzo Serugendo, Anthony Karageorgos, Omer F. Rana, and Franco Zambonelli, eds.), Springer Berlin Heidelberg, 2004, pp. 105–123.

[HW92] Andreas Huth and Christian Wissel, *The simulation of the movement of fish schools*, Journal of Theoretical Biology **156** (1992), no. 3, 365–385.

[HW06] Julianne D. Halley and David A. Winkler, *Classification of Self-Organization and Emergence in Chemical and Biological Systems*, Australian Journal of Chemistry **59** (2006), no. 12, 849–853.

[HW08] Julianne. D. Halley and David A. Winkler, *Classification of Emergence and Its Relation to Self-Organization*, Complexity **13** (2008), no. 5, 10–15.

[HWH99] Jim Hughes and Trevor Wood-Harper, *Systems development as a research act*, Journal of Information Technology **14** (1999), no. 1, 83–94.

[IA12] IoT-A, *IoT-A Internet of Things Architecture. http://www-iot-a.eu/*, VDI/VDE INNOVATION + TECHNIK GMBH (2012), 1.

[IMA+14] Kiyohiro Ikeda, Kazuo Murota, Takashi Akamatsu, Tatsuhito Kono, and Yuki Takayama, *Self-organization of hexagonal agglomeration patterns in new economic geography models*, Journal of Economic Behavior & Organization **99** (2014), 32–52.

[IMP01] Paola Inverardi, Henry Muccini, and Patrizio Pelliccione, *Automated check of architectural models consistency using SPIN*, Proceedings 16th Annual International Conference on Automated Software Engineering (ASE 2001), 2001, pp. 346–349.

[Joh06] Christopher W. Johnson, *What are emergent properties and how do they affect the engineering of complex systems?*, Reliability Engineering & System Safety **91** (2006), no. 12, 1475–1481, Complexity in Design and Engineering.

[JTSP13] John J. Johnson, Andreas Tolk, and Andres Sousa-Poza, *A Theory of Emergence and Entropy in Systems of Systems*, Procedia Computer Science **20** (2013), 283 – 289, Complex Adaptive Systems.

[JV11] Silver Juurik and Jüri Vain, *Model checking of emergent behaviour properties of robot swarms*, Proceedings of the Estonian Academy of Sciences **60** (2011), no. 1, 48–54.

[Kau93] Stuart A Kauffman, *The Origins of Order: Self-Organization and Selection in Evolution.*, Oxford University Press, USA, 1993.

[Kau95] Stuart Kauffman, *At Home in the Universe: the Search for the Laws of Self- Organization and Complexity.*, Oxford University Press, USA, 1995.

[KC06] Stuart Kauffman and Philip Clayton, *On emergence, agency, and organization*, Biology and Philosophy **21** (2006), no. 4, 501–521.

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

[KC20] Juliana Küster Filipe Bowles and Marco B. Caminati, *Correct composition in the presence of behavioural conflicts and dephasing*, Science of Computer Programming **185** (2020), 102323.

[KGSB99] Ingolf Krüger, Radu Grosu, Peter Scholz, and Manfred Broy, *From MSCS to Statecharts*, Distributed and Parallel Embedded Systems: IFIP WG10.3/WG10.5 International Workshop on Distributed and Parallel Embedded Systems (DIPES'98) October 5–6, 1998, Schloss Eringerfeld, Germany (Boston, MA) (Franz J. Rammig, ed.), Springer US, 1999, pp. 61–71.

[Kin10] Witold Kinsner, *System Complexity and Its Measures: How Complex Is Complex*, Advances in Cognitive Informatics and Cognitive Computing (Berlin, Heidelberg) (Yingxu Wang, Du Zhang, and Witold Kinsner, eds.), Springer Berlin Heidelberg, 2010, pp. 265–295.

[KKP+14] Petra Kaufmann, Martin Kronegger, Andreas Pfandler, Martina Seidl, and Magdalena Widl, *A SAT-Based Debugging Tool for State Machines and Sequence Diagrams*, Software Language Engineering (Cham) (Benoît Combemale, David J. Pearce, Olivier Barais, and Jurgen J. Vinju, eds.), Springer International Publishing, 2014, pp. 21–40.

[KKP+15] Petra Kaufmann, Martin Kronegger, Andreas Pfandler, Martina Seidl, and Magdalena Widl, *Intra- and interdiagram consistency checking of behavioral multiview models*, Computer Languages, Systems & Structures **44** (2015), 72–88, Special issue on the 6th and 7th International Conference on Software Language Engineering (SLE 2013 and SLE 2014).

[Kli91] Yu.L. Klimontovich, *Turbulent Motion and the Structure of Chaos: A New Approach to the Statistical Theory of Open Systems*, 1 ed., Fundamental Theories of Physics, no. 42, Springer Netherlands, 1991.

[KMRF+03] Soraya Kouadri Mostefaoui, Omer F. Rana, Noria Foukia, Salima Hassas, Giovanna Di Marzo-Serugendo, Chris Van Aart, and Anthony Karageorgos, *Self-Organising Applications: A Survey*, AAMAS'2003 Workshop on Engineering Self-Organizing Applications, 15 July 2003, Melbourne, Australia. pp 62-69. (Melbourne, Australia) (G. Di Marzo-Serugendo, A.Karageorgos, O.F. Rana, and F. Zambonellini, eds.), Springer Verlag, 7 2003.

[KNM20] Somayeh Kalantari, Eslam Nazemi, and Behrooz Masoumi, *Emergence phenomena in self-organizing systems: a systematic literature review of concepts, researches, and future prospects*, Journal of Organizational Computing and Electronic Commerce (2020), 1 – 42.

[Kop87] Moshe Koppel, *Complexity, Depth, and Sophistication*, Complex Systems **1** (1987), no. 6, 1087–1091.

[Kru96] Paul Krugman, *The self organizing economy*, John Wiley & Sons, 1996.

[KU76] John B. Kam and Jeffrey D. Ullman, *Global Data Flow Analysis and Iterative Algorithms*, J. ACM **23** (1976), no. 1, 158–171.

[Kub01] Aleš Kubí, *On Emergence in Evolutionary Multiagent Systems*, Advances in Artificial Life. Proceedings of the 6th European Conference on Artificial Life (Berlin,

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Heidelberg) (Jozef Kelemen and Petr Sosík, eds.), Springer Berlin Heidelberg, 2001, pp. 326–337.

[Kub03] ———, *Toward a Formalization of Emergence*, Artificial Life **9** (2003), no. 1, 41–65.

[Küs91] Gerhard Küster, *On the Hurwitz product of formal power series and automata*, Theoretical Computer Science **83** (1991), no. 2, 261–273.

[KW07] Alexander Knapp and Jochen Wuttke, *Model Checking of UML 2.0 Interactions*, Models in Software Engineering (Berlin, Heidelberg) (Thomas Kühne, ed.), Springer Berlin Heidelberg, 2007, pp. 42–51.

[KZV95] Taek Mu Kwon, Michael E. Zervakis, and Anastasios N. Venetsanopoulos, *Design and analysis of a class of self-organizing and trainable fuzzy controllers*, Journal of Intelligent and Robotic Systems **12** (1995), no. 3, 301–315.

[Lan86] Christopher G Langton, *Studying artificial life with cellular automata*, Physica D: Nonlinear Phenomena **22** (1986), no. 1, 120–149, Proceedings of the Fifth Annual International Conference.

[Lan90] ———, *Computation at the edge of chaos: Phase transitions and emergent computation*, Physica D: Nonlinear Phenomena **42** (1990), no. 1, 12–37.

[LBT12] Paulo Leitão, José Barbosa, and Damien Trentesaux, *Bio-inspired multi-agent systems for reconfigurable manufacturing systems*, Engineering Applications of Artificial Intelligence **25** (2012), no. 5, 934–944.

[LDFK23] Christoph Lehr, Patrick Denzler, Thomas Frühwirth, and Wolfgang Kastner, *Buffer management for tsn-enabled end stations*, 2023 IEEE 19th International Conference on Factory Communication Systems (WFCS), 2023, pp. 1–8.

[Lew75] George Henry Lewes, *Problems of Life and Mind*, vol. 2, Trübner & Company, 1875.

[LFK+14] Heiner Lasi, Peter Fettke, Hans Georg Kemper, Thomas Feld, and Michael Hoffmann, *Industry 4.0*, Business and Information Systems Engineering **6** (2014), no. 4, 239–242.

[LG13] Joseph P. Lancaster and David A. Gustafson, *Predicting the Behavior of Robotic Swarms in Search and Tag Tasks*, Procedia Computer Science **20** (2013), 77 – 82, Complex Adaptive Systems.

[LGG+08] Mingsong Lv, Zonghua Gu, Nan Guan, Qingxu Deng, and Ge Yu, *Performance Comparison of Techniques on Static Path Analysis of WCET*, 2008 IEEE/IFIP International Conference on Embedded and Ubiquitous Computing, vol. 1, 2008, pp. 104–111.

[LGZ+09] Mingsong Lv, Nan Guan, Yi Zhang, Qingxu Deng, Ge Yu, and Jianming Zhang, *A Survey of WCET Analysis of Real-Time Operating Systems*, 2009 International Conference on Embedded Software and Systems, 2009, pp. 65–72.

[Lis03] Björn Lisper, *Fully Automatic, Parametric Worst-Case Execution Time Analysis.*, WCET **3** (2003), 77–80.

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

[LLK+08] Ben Lickly, Isaac Liu, Sungjun Kim, Hiren D. Patel, Stephen A. Edwards, and Edward A. Lee, *Predictable Programming on a Precision Timed Architecture*, Proceedings of the 2008 International Conference on Compilers, Architectures and Synthesis for Embedded Systems (New York, USA), Association for Computing Machinery, 2008, pp. 137–146.

[LLW13] James Ladyman, James Lambert, and Karoline Wiesner, *What is a complex system?*, European Journal for Philosophy of Science **3** (2013), no. 1, 33–67.

[LMT09] Francisco J. Lucas, Fernando Molina, and Ambrosio Toval, *A systematic review of UML model consistency management*, Information and Software Technology **51** (2009), no. 12, 1631–1645, Quality of UML Models.

[LMZS06] Hui Liu, Zhiyi Ma, Lu Zhang, and Weizhong Shao, *Detecting Duplications in Sequence Diagrams Based on Suffix Trees*, 2006 13th Asia Pacific Software Engineering Conference (APSEC'06), 2006, pp. 269–276.

[Lon70] Ralph L. London, *Computer Programs can be Proved Correct*, Theoretical Approaches to Non-Numerical Problem Solving (R.B. Banerji and M.D. Mesarovic, eds.), Lecture Notes in Operations Research and Mathematical Systems, vol. 28, Springer, Berlin, Heidelberg, 1970, pp. 281–302.

[LP05] Vitus S. W. Lam and Julian Padget, *Consistency Checking of Sequence Diagrams and Statechart Diagrams Using the π-Calculus*, Integrated Formal Methods (Berlin, Heidelberg) (Judi Romijn, Graeme Smith, and Jaco van de Pol, eds.), Springer Berlin Heidelberg, 2005, pp. 347–365.

[LS86] John P. Lehoczky and Lui Sha, *Performance of Real-Time Bus Scheduling Algorithms*, SIGMETRICS Perform. Eval. Rev. **14** (1986), no. 1, 44–53.

[LS17] Edward A. Lee and Sanjit A. Seshia, *Introduction to Embedded Systems, A Cyber-Physical Systems Approach*, MIT Press, 2017.

[LSHL06] Zhengping Li, Cheng Hwee Sim, and Malcolm Yoke Hean Low, *A Survey of Emergent Behavior and Its Impacts in Agent-based Systems*, 2006 4th IEEE International Conference on Industrial Informatics, 2006, pp. 1295–1300.

[LSR+88] Douglass Locke, Lui Sha, Ragunathan Rajikumar, John Lehoczky, and Greg Burns, *Priority Inversion and Its Control: An Experimental Investigation*, Ada Lett. **VIII** (1988), no. 7, 39–42.

[Luc97] Chris Lucas, *Emergence and Evolution - Constraints on Form*, http://www.calresco.org/emerge.htm, 1997.

[Mac01] Donald A. MacKenzie, *Mechanizing Proof: Computing, Risk, and Trust (inside technology)*, MIT Press, Cambridge, Massachusetts, 2001.

[MAH10] Thomas Moncion, Patrick Amar, and Guillaume Hutzler, *Automatic characterization of emergent phenomena in complex systems*, Journal of Biological Physics and Chemistry **10** (2010), 16–23.

[Mah11] Michael S. Mahoney, *Histories of Computing*, 1 ed., Harvard University Press, 2011.

TU WIEN Bibliothek Your knowledge hub
Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.

[Mau97] S. Mauw, *ITU-TS Recommendation Z.120: Message Sequence Chart (MSC)*, ITU-TS, 1997 (English).

[MB89] Thomas J. McCabe and Charles W. Butler, *Design Complexity Measurement and Testing*, Commun. ACM **32** (1989), no. 12, 1415–1425.

[MB11] Robert Mittermayr and Johann Blieberger, *Shared Memory Concurrent System Verification using Kronecker Algebra*, Tech. report, Automation Systems Group, TU Vienna, 2011.

[MB12] Robert Mittermayr and Johann Blieberger, *Timing Analysis of Concurrent Programs*, 12th International Workshop on Worst-Case Execution Time Analysis (Dagstuhl, Germany) (Tullio Vardanega, ed.), OpenAccess Series in Informatics (OASIcs), vol. 23, Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2012, pp. 59–68.

[MB16a] ______, *A Generic Graph Model for WCET Analysis of Multi-Core Concurrent Applications*, Journal of Software Engineering and Applications **9** (2016), no. 5, 17.

[MB16b] Robert Mittermayr and Johann Blieberger, *Kronecker Algebra for Static Analysis of Barriers in Ada*, Reliable Software Technologies – Ada-Europe 2016 (Cham) (Marko Bertogna, Luis Miguel Pinho, and Eduardo Quiñones, eds.), Springer International Publishing, 2016, pp. 145–159.

[MB21] Robert Mittermayr and Johann Blieberger, *Deadlock and WCET analysis of barrier-synchronized concurrent programs*, Computing.Archives for Informatics and Numerical Computation **103** (2021), no. 5, 749–770.

[McA63] MH McAndrew, *On The Product of Directed Graphs*, Proceedings of the American Mathematical Society **14** (1963), no. 4, 600–606.

[MFT05] Gregory Madey, Vincent Freeh, and Renee Tynan, *Modeling the Free/Open Source Software Community: A Quantitative Investigation*, Free/Open Source Software Development (Stefan Koch, ed.), IGI Global, 2005, pp. 203–221.

[MG15] Jamie P. Monat and Thomas F. Gannon, *What is systems thinking? A review of selected literature plus recommendations*, American Journal of Systems Science **4** (2015), no. 1, 11–26.

[MG18] Jessica Moysen and Lorenza Giupponi, *From 4G to 5G: Self-organized network management meets machine learning*, Computer Communications **129** (2018), 248–268.

[Mit13] Saurabh Mittal, *Emergence in stigmergic and complex adaptive systems: A formal discrete event systems perspective*, Cognitive Systems Research **21** (2013), 22 – 39.

[Mit19] Saurabh Mittal, *New Frontiers in Modeling and Simulation in Complex Systems Engineering: The Case of Synthetic Emergence*, Summer of Simulation: 50 Years of Seminal Computer Simulation Research (Cham) (John Sokolowski, Umut Durak, Navonil Mustafee, and Andreas Tolk, eds.), Springer International Publishing, 2019, pp. 173–194.

[MKS00] Madhavan Mukund, K. Narayan Kumar, and Milind Sohoni, *Synthesizing Distributed Finite-State Systems from MSCs*, CONCUR 2000 —Concurrency Theory (Berlin, Heidelberg) (Catuscia Palamidessi, ed.), Springer Berlin Heidelberg, 2000, pp. 521–535.

TU WIEN Bibliothek Your knowledge hub
Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.

[MLD09] Wolfgang Mahnke, Stefan-Helmut Leitner, and Matthias Damm, *OPC Unified Architecture*, Springer Science & Business Media, 2009.

[MMBH10] Mohammad Moshirpour, Abdolmajid Mousavi, and Far Behrouz H., *A Technique and a Tool to Detect Emergent Behavior of Distributed Systems Using Scenario-Based Specifications*, 2010 22nd IEEE International Conference on Tools with Artificial Intelligence, vol. 1, Oct 2010, pp. 153–159.

[MMEF12] Mohammad Moshirpour, Seyedehmerhnaz Mireslami, Armin Eberlein, and Behrouz H. Far, *A method to detect and remove emergent behavior caused by overgeneralization*, 2012 IEEE International Conference on Systems, Man, and Cybernetics (SMC), 2012, pp. 2469–2474.

[MMS11] Moez Mnif and Christian Müller-Schloer, *Quantitative emergence*, Organic Computing —A Paradigm Shift for Complex Systems (Basel) (Christian Müller-Schloer, Hartmut Schmeck, and Theo Ungerer, eds.), Springer Basel, 2011, pp. 39–52.

[MRM17] Saurabh Mittal and José L. Risco-Martín, *Simulation-Based Complex Adaptive Systems*, Guide to Simulation-Based Disciplines: Advancing Our Computational Future (Cham) (Saurabh Mittal, Umut Durak, and Tuncer Ören, eds.), Springer International Publishing, 2017, pp. 127–150.

[MRS02] Philippe Mathieu, Jean-Christophe Routier, and Yann Secq, *Principles for Dynamic Multi-agent Organizations*, Intelligent Agents and Multi-Agent Systems (Berlin, Heidelberg) (Kazuhiro Kuwabara and Jaeho Lee, eds.), Springer Berlin Heidelberg, 2002, pp. 109–122.

[MS95] Salvatore T. March and Gerald F. Smith, *Design and natural science research on information technology*, Decision Support Systems **15** (1995), no. 4, 251–266.

[MS04] Christian Müller-Schloer, *Organic Computing: On the Feasibility of Controlled Emergence*, Proceedings of the 2nd IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis (New York, NY, USA), CODES+ISSS '04, Association for Computing Machinery, 2004, pp. 2–5.

[MSB11] Glenford J. Myers, Corey Sandler, and Tom Badgett, *THE ART OF SOFTWARE TESTING*, 3rd ed., John Wiley & Sons Ltd., 2011.

[MSHD18] Radu-Casian Mihailescu, Romina Spalazzese, Clint Heyer, and Paul Davidsson, *A Role-Based Approach for Orchestrating Emergent Configurations in the Internet of Things*, 2018.

[Mur88] James D. Murray, *How the Leopard Gets Its Spots*, Scientific American **258** (1988), no. 3, 80–87.

[MYA+19] Akira Matsumoto, Tomoyuki Yokogawa, Sousuke Amasaki, Hirohisa Aman, and Kazutami Arimoto, *Consistency Verification of UML Sequence Diagrams Modeling Wireless Sensor Networks*, 2019 8th International Congress on Advanced Applied Informatics (IIAI-AAI), 2019, pp. 458–461.

[MYA+20] ______, *Synthesis and Consistency Verification of UML Sequence Diagrams with Hierarchical Structure*, Information Engineering Express **6** (2020), no. 2, 1–19.

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

[MZ06] Marco Mamei and Franco Zambonelli, *Self-organizing Approaches for Large-Scale Spray Multiagent Systems*, Software Engineering for Multi-Agent Systems IV (Berlin, Heidelberg) (Alessandro Garcia, Ricardo Choren, Carlos Lucena, Paolo Giorgini, Tom Holvoet, and Alexander Romanovsky, eds.), Springer Berlin Heidelberg, 2006, pp. 53–70.

[New96] David V. Newman, *Emergence and Strange Attractors*, Philosophy of Science **63** (1996), no. 2, 245–261.

[NF06] David Newth and John Finnigan, *Emergence and Self-Organization in Chemistry and Biology*, Australian Journal of Chemistry **59** (2006), no. 12, 841–848.

[Nic93] Gregoire Nicolis, *Physics of far-from-equilibrium systems and self-organization*, Flammarion, France, 1993.

[NP77] Gregoire Nicolis and Ilya Prigogine, *Self-Organization in Nonequilibrium Systems: From Dissipative Structures to Order through Fluctuations*, Wiley, New York, NY, USA, 1977.

[Oat05] Briony J Oates, *Researching Information Systems and Computing*, 2012 ed., SAGE Publications, Ltd., 2005.

[Ode02a] James Odell, *Agents and Complex Systems*, Journal of Object Technology **1** (2002), no. 2, 35–45.

[Ode02b] ______, *Objects and Agents Compared*, Journal of Object Technology **1** (2002), no. 1, 41–53.

[ONC14] Eamonn O'Toole, Vivek Nallur, and Siobhán Clarke, *Towards Decentralised Detection of Emergence in Complex Adaptive Systems*, 2014 IEEE Eighth International Conference on Self-Adaptive and Self-Organizing Systems, 9 2014, pp. 60 – 69.

[ONC17] ______, *Decentralised Detection of Emergence in Complex Adaptive Systems*, ACM Trans. Auton. Adapt. Syst. **12** (2017), no. 1, 1 – 31.

[OOO20] Toshitaka Odamura, Takayuki Omori, and Atsushi Ohnishi, *Supporting Change Management of Sequence Diagrams*, Knowledge-Based Software Engineering: 2020 (Cham) (Maria Virvou, Hiroyuki Nakagawa, and Lakhmi C. Jain, eds.), Springer International Publishing, 2020, pp. 35–46.

[OPC18] OPC Foundation, *OPC Unified Architecture Specification Part 14: PubSub, Release 1.04*, 2018.

[ORS13] Haluk Ozaktas, Christine Rochange, and Pascal Sainrat, *Automatic WCET Analysis of Real-Time Parallel Applications*, 13th International Workshop on Worst-Case Execution Time Analysis (Dagstuhl, Germany) (Claire Maiza, ed.), OpenAccess Series in Informatics (OASIcs), vol. 30, Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2013, pp. 11–20.

[Oud99] Pierre-Yves Oudeyer, *Self-Organization of a Lexicon in a Structured Society of Agents.*, Advances in Artificial Life. ECAL 1999. Lecture Notes in Computer Science (Berlin, Heidelberg) (D. Floreano, J.D. Nicoud, and F. Mondada, eds.), vol. 1674, Springer, 1999, pp. 725–729.

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

[Pag88] Heinz R. Pagels, *The Dreams of Reason: The Computer and the Rise of the Sciences of Complexity*, Simon & Schuster, New York, NY, USA, 1988.

[PBR09] Mikhail Prokopenko, Fabio Boschetti, and Alex J. Ryan, *An Information-Theoretic Primer on Complexity, Self-Organization, and Emergence*, Complexity **15** (2009), no. 1, 11–28.

[Pet62] Carl Adam Petri., *Kommunikation mit Automaten.*, Ph.D. thesis, Schriften des Institutes für Instrumentelle Mathematik, Bonn, 1962.

[PHP15] Daniel Prokesch, Stefan Hepp, and Peter Puschner, *A Generator for Time-Predictable Code*, 2015 IEEE 18th International Symposium on Real-Time Distributed Computing, 2015, pp. 27–34.

[PIM09] Patrizio Pelliccione, Paola Inverardi, and Henry Muccini, *CHARMY: A Framework for Designing and Verifying Architectural Specifications*, IEEE Transactions on Software Engineering **35** (2009), no. 3, 325–346.

[PK89] Peter Puschner and Christian Koza, *Calculating the Maximum Execution Time of Real-Time Programs*, Real-Time Syst. **1** (1989), no. 2, 159–176.

[Pla85] Brigitte Plateau, *On the Stochastic Structure of Parallelism and Synchronization Models for Distributed Algorithms*, SIGMETRICS Perform. Eval. Rev. **13** (1985), no. 2, 147–154.

[Pop05] Karl Popper, *The Logic of Scientific Discovery*, 2 ed., Routledge, 2005.

[Pri78] Ilya Prigogine, *Time, Structure, and Fluctuations*, Science **201** (1978), no. 4358, 777–785.

[Pro11] Addy Pross, *Toward a general theory of evolution: Extending Darwinian theory to inanimate matter*, Journal of Systems Chemistry **2** (2011), no. 1, 1.

[PTLB15] Luca Pezzarossa, Jakob Kenn Toft, Jesper Lønbæk, and Russell Barnes, *Implementation of an Ethernet-Based Communication Channel for the Patmos Processor*, Tech. Report No. 2, Technical University of Denmark. DTU Compute, 2015.

[Rap21] Rapita Systems, *Rapitime*, Available at https://www.rapitasystems.com/products/rapitime, 2021.

[RB95] Steen Rasmussen and Christopher L. Barrett, *Elements of a Theory of Simulation*, Advances in Artificial Life (Berlin, Heidelberg) (Federico Morán, Alvaro Moreno, Juan Julián Merelo, and Pablo Chacón, eds.), Springer Berlin Heidelberg, 1995, pp. 515–529.

[Rei12] Wolfgang Reisig, *Petri Nets: An Introduction*, Monographs in Theoretical Computer Science. An EATCS Series (EATCS, volume 4), vol. 4, Springer, Berlin, Heidelberg, 2012.

[RMNV18] Damian Roca, Rodolfo Milito, Mario Nemirovsky, and Mateo Valero, *Tackling IoT Ultra Large Scale Systems: Fog Computing in Support of Hierarchical Emergent Behaviors*, Fog Computing in the Internet of Things: Intelligence at the Edge (Cham) (Amir M. Rahmani, Pasi Liljeberg, Jürgo-Sören Preden, and Axel Jantsch, eds.), Springer International Publishing, 2018, pp. 33–48.

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

[RNN+16] Damian Roca, Daniel Nemirovsky, Mario Nemirovsky, Rodolfo Milito, and Mateo Valero, *Emergent Behaviors in the Internet of Things: The Ultimate Ultra-Large-Scale System*, IEEE Micro **36** (2016), no. 6, 36–44.

[RP86] Barbara G. Ryder and Marvin C. Paull, *Elimination Algorithms for Data Flow Analysis*, ACM Comput. Surv. **18** (1986), no. 3, 277–316.

[RP88] ______, *Incremental Data-Flow Analysis Algorithms*, ACM Trans. Program. Lang. Syst. **10** (1988), no. 1, 1–50.

[RS00] Edmund M. A. Ronald and Moshe Sipper, *Engineering, Emergent Engineering, and Artificial Life: Unsurprise, Unsurprising Surprise, and Surprising Surprise*, Artificial Life VII: Proceedings of the Seventh International Conference on Artificial Life (Cambridge) (Mark A. Bedau, John S. McCaskill, Norman H. Packard, and Steen Rasmussen, eds.), The MIT Press, 2000, pp. 523–528.

[RSC99] Edmund M. A. Ronald, Moshe Sipper, and Mathieu S. Capcarrère, *Design, Observation, Surprise! A Test of Emergence*, Artificial Life **5** (1999), no. 3, 225–239.

[RVH+04a] Christopher Rouff, Amy Vanderbilt, Mike Hinchey, Walt Truszkowski, and James Rash, *Properties of a Formal Method for Prediction of Emergent Behaviors in Swarm-Based Systems*, Proceedings of the Second International Conference on Software Engineering and Formal Methods, 2004. SEFM 2004., 2004, pp. 24–33.

[RVH+04b] ______, *Verification of Emergent Behaviors in Swarm-Based Systems*, Proceedings. 11th IEEE International Conference and Workshop on the Engineering of Computer-Based Systems, 2004., 2004, pp. 443–448.

[Rya07] Alex J. Ryan, *Emergence is Coupled to Scope, Not Level*, Complexity **13** (2007), no. 2, 67–77.

[SAA+15] Martin Schoeberl, Sahar Abbaspour, Benny Akesson, Neil Audsley, Raffaele Capasso, Jamie Garside, Kees Goossens, Sven Goossens, Scott Hansen, Reinhold Heckmann, Stefan Hepp, Benedikt Huber, Alexander Jordan, Evangelia Kasapaki, Jens Knoop, Yonghui Li, Daniel Prokesch, and et al., *T-CREST: Time-predictable multi-core architecture for embedded eystems*, Journal of Systems Architecture **61** (2015), no. 9, 449–471.

[Sal73] Arto Salomaa, *Formal Languages*, 1 ed., Academic Press Professional, Inc., San Diego, CA, 1973.

[Saw01] R. Keith Sawyer, *Emergence in Sociology: Contemporary Philosophy of Mind and Some Implications for Sociological Theory*, American Journal of Sociology **107** (2001), no. 3, 551–585.

[Saw02] ______, *Emergence in Psychology: Lessons from the History of Non-Reductionist Science*, Human Development **45** (2002), no. 1, 2–28.

[Saw05] ______, *Social Emergence: Societies As Complex Systems*, Cambridge University Press, 2005.

[SC01] Cosma Rohilla Shalizi and James P. Crutchfield, *Computational Mechanics: Pattern and Prediction, Structure and Simplicity*, Journal of Statistical Physics **104** (2001), no. 3, 817–879.

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

[SC13] Matthew Stephan and James R. Cordy, *A Survey of Model Comparison Approaches and Applications*, Proceedings of the 1st International Conference on Model-Driven Engineering and Software Development - Volume 1: MODELSWARD,, INSTICC, SciTePress, 2013, pp. 265–277.

[SC18] Rick L. Sturdivant and Edwin K. P. Chong, *The Necessary and Sufficient Conditions for Emergence in Systems Applied to Symbol Emergence in Robots*, IEEE Transactions on Cognitive and Developmental Systems **10** (2018), no. 4, 1035–1042.

[Sel59] Oliver G Selfridge, *Pandemonium: A Paradigm For Learning*, The Mechanisation of Thought Processes. (D. Blake and A. Uttley, eds.), National Physical Laboratory Symposia, Her Majesty's Stationary Office, London, 1959, pp. 511–529.

[Set08] Anil K Seth, *Measuring emergence via nonlinear Granger causality*, Artificial Life XI: Proceedings of the 11th International Conference on the Simulation and Synthesis of Living Systems, ALIFE 2008, 2008, pp. 545–552.

[SG13] Mauricio Salgado and Nigel Gilbert, *Emergence and Communication in Computational Sociology*, Journal for the Theory of Social Behaviour **43** (2013), no. 1, 87–110.

[Sha01] Cosma Rohilla Shalizi, *Causal Architecture, Complexity and Self-Organization in Time Series and Cellular Automata.*, Ph.D. thesis, University of Michigan, 2001.

[Sha03] ———, *Optimal Nonlinear Prediction Of Random Fields On Networks*, arXiv preprint math/0305160 (2003), 1–20.

[SJ02] Udo Seiffert and Lakhmi C. Jain (eds.), *Self-organizing Neural Networks: Recent Advances and Applications: Studies in Fuzziness and Soft Computing; hardbound*, vol. 78, Physica-Verlag, Heidelberg, 2002.

[SJB09] In-Gwon Song, Sang-Uk Jeon, and Doo-Hwan Bae, *A Graph Based Approach to Detecting Causes of Implied Scenarios under the Asynchronous and Synchronous Communication Styles*, 2009 16th Asia-Pacific Software Engineering Conference, 2009, pp. 53–60.

[SJHB11] In-Gwon Song, Sang-Uk Jeon, Ah-Rim Han, and Doo-Hwan Bae, *An approach to identifying causes of implied scenarios using unenforceable orders*, Information and Software Technology **53** (2011), no. 6, 666–681, Special Section: Best papers from the APSEC.

[SK14] Cosma Rohilla Shalizi and Kristina Lisa Klinkner, *Blind construction of optimal nonlinear recursive predictors for discrete sequences*, arXiv preprint arXiv:1408.2025 (2014), 504–511.

[SKJ18] Sebastian Schriegel, Thomas Kobzan, and Jürgen Jasperneite, *Investigation on a distributed SDN control plane architecture for heterogeneous time sensitive networks*, 2018 14th IEEE International Workshop on Factory Communication Systems (WFCS), 6 2018, pp. 1–10.

[SKM01] Timm Schäfer, Alexander Knapp, and Stephan Merz, *Model Checking UML State Machines and Collaborations*, Electronic Notes in Theoretical Computer Science **55** (2001), no. 3, 357–369, Workshop on Software Model Checking (in connection with CAV '01).

TU WIEN Bibliothek Your knowledge hub
Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.

[SMP+17] Wasuwee Sodsong, Robert Mittermayr, Yoojin Park, Bernd Burgstaller, and Johann Blieberger, *Lazy Parallel Kronecker Algebra-Operations on Heterogeneous Multicores*, Euro-Par 2017: Parallel Processing (Cham) (Francisco F. Rivera, Tomás F. Pena, and José C. Cabaleiro, eds.), Springer International Publishing, 2017, pp. 538–552.

[Sne98] Gregor Snelting, *Paul Feyerabend and software technology*, International Journal on Software Tools for Technology Transfer **2** (1998), no. 1, 1–5.

[SP05] Sahotra Sarkar and Jessica Pfeifer, *The Philosophy of Science 2-Volume Set: An Encyclopedia*, Routledge, 2005.

[SPH+18] Martin Schoeberl, Wolfgang Puffitsch, Stefan Hepp, Benedikt Huber, and Daniel Prokesch, *Patmos: A Time-Predictable Microprocessor*, Real-Time Systems **54** (2018), no. 2, 389–423.

[SRL90] Lui. Sha, Ragunathan Rajkumar, and John P. Lehoczky, *Priority Inheritance Protocols: An Approach to Real-Time Synchronization*, IEEE Trans. Comput. **39** (1990), no. 9, 1175–1185.

[SSB18] Andreas Schöbel, Christian Schöbel, and Johann Blieberger, *Kronecker Algebra for Managing Rail Capacity*, ISEP2018–26th International Symposium on Electronics in Transport, ISEP2018, 2018.

[SSH04] Cosma Rohilla Shalizi, Kristina Lisa Shalizi, and Robert Haslinger, *Quantifying Self-Organization With Optimal Predictors*, Physical review letters **93** (2004), no. 11, 118701.

[ST99] Dan A Simovici and Richard L Tenney, *THEORY OF FORMAL LANGUAGES WITH APPLICATIONS*, World Scientific Publishing Company, Signapore, 1999.

[ST13] Claudia Szabo and Yong Meng Teo, *Semantic Validation of Emergent Properties in Component-Based Simulation Models*, Ontology, Epistemology, and Teleology for Modeling and Simulation: Philosophical Foundations for Intelligent M&S Applications (Berlin, Heidelberg) (Andreas Tolk, ed.), Springer Berlin Heidelberg, 2013, pp. 319–333.

[Sta00] Ralph Stacey, *The Emergence of Knowledge in Organization*, Emergence **2** (2000), no. 4, 23–39.

[Sta12] William Stallings, *OPERATING SYSTEMS: Internals and Design Principles*, 9th ed., Person, 2012.

[Ste90] Luc Steels, *Cooperation between distributed agents through self-organisation*, EEE International Workshop on Intelligent Robots and Systems, Towards a New Frontier of Applications, 1990, pp. 8–14.

[Szo09] Jack W. Szostak, *Systems chemistry on early Earth*, Nature **459** (2009), no. 7244, 171–172.

[TDM18] Andreas Tolk, Saikou Diallo, and Saurabh Mittal, *COMPLEX SYSTEMS ENGINEERING AND THE CHALLENGE OF EMERGENCE*, ch. 5, pp. 78–97, John Wiley & Sons, Ltd, 2018.

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

[TFSR19] Evi Triandini, Reza Fauzan, Daniel O Siahaan, and Siti Rochimah, *Sequence Diagram Similarity Measurement: A Different Approach*, 2019 16th International Joint Conference on Computer Science and Software Engineering (JCSSE), 2019, pp. 348–351.

[TNN+16] Tadahiro Taniguchi, Takayuki Nagai, Tomoaki Nakamura, Naoto Iwahashi, Tetsuya Ogata, and Hideki Asoh, *Symbol emergence in robotics: a survey*, Advanced Robotics **30** (2016), no. 11-12, 706–728.

[Tol19] Andreas Tolk, *Limitations and Usefulness of Computer Simulations for Complex Adaptive Systems Research*, Summer of Simulation: 50 Years of Seminal Computer Simulation Research (Cham) (John Sokolowski, Umut Durak, Navonil Mustafee, and Andreas Tolk, eds.), Springer International Publishing, 2019, pp. 77–96.

[TSE94] Giulio Tononi, Olaf Sporns, and Gerald M. Edelman, *A measure for brain complexity: Relating functional segregation and integration in the nervous system*, Proceedings of the National Academy of Sciences **91** (1994), no. 11, 5033–5037.

[Tur50] A. Turing, *Computing machinery and intelligence*, Mind (1950), 433–460.

[Tur59] Herbert Westren Turnbull, *The Correspondence of Isaac Newton: 1661–1675*, vol. 1, p. 416, Published for the Royal Society at the University Press., London, UK, 1959.

[Uch09] Sebastian Uchitel, *Partial Behaviour Modelling: Foundations for Incremental and Iterative Model-Based Software Engineering*, Formal Methods: Foundations and Applications (Berlin, Heidelberg) (Marcel Vinícius Medeiros Oliveira and Jim Woodcock, eds.), Springer Berlin Heidelberg, 2009, pp. 17–22.

[UK01] Sebastian Uchitel and Jeff Kramer, *A workbench for synthesising behaviour models from scenarios*, Proceedings of the 23rd International Conference on Software Engineering. ICSE 2001, 2001, pp. 188–197.

[UNKC08] Muhammad Usman, Aamer Nadeem, Tai-hoon Kim, and Eun-suk Cho, *A Survey of Consistency Checking Techniques for UML Models*, 2008 Advanced Software Engineering and Its Applications, 2008, pp. 57–62.

[VDPB01] H. Van Dyke Parunak and Sven Brueckner, *Entropy and Self-Organization in Multi-Agent Systems*, Proceedings of the Fifth International Conference on Autonomous Agents (New York, NY, USA), AGENTS '01, Association for Computing Machinery, 2001, pp. 124–130.

[VDPB04] H. Van Dyke Parunak and Sven A. Brueckner, *Engineering Swarming Systems*, Methodologies and Software Engineering for Agent Systems: The Agent-Oriented Software Engineering Handbook (Boston, MA) (Federico Bergenti, Marie-Pierre Gleizes, and Franco Zambonelli, eds.), Springer US, 2004, pp. 341–376.

[VDPBS01] H. Van Dyke Parunak, Sven Brueckner, and John Sauter, *ERIM's Approach to Fine-Grained Agents.*, NASA/JPL Workshop on Radical Agent Concepts, Greenbelt (MD), 9 2001, pp. 19–21.

[VDPSR98] H. Van Dyke Parunak, Robert Savit, and Rick L. Riolo, *Agent-Based Modeling vs. Equation-Based Modeling: A Case Study and Users' Guide*, Multi-Agent Systems and Agent-Based Simulation (Berlin, Heidelberg) (Jaime Simão Sichman, Rosaria Conte, and Nigel Gilbert, eds.), Springer Berlin Heidelberg, 1998, pp. 10–25.

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

[VDPV97] H. Van Dyke Parunak and Raymond S. VanderBok, *Managing Emergent Behavior in Distributed Control Systems*, Presented at ISA-Tech '97 (1997), 1–9.

[VDSMSJ03] Ragnhild Van Der Straeten, Tom Mens, Jocelyn Simmonds, and Viviane Jonckers, *Using Description Logic to Maintain Consistency between UML Models*, UML 2003 - The Unified Modeling Language. Modeling Languages and Applications (Berlin, Heidelberg) (Perdita Stevens, Jon Whittle, and Grady Booch, eds.), Springer Berlin Heidelberg, 2003, pp. 326–340.

[VFB+13] Marco Villani, Alessandro Filisetti, Stefano Benedettini, Andrea Roli, David Lane, and Roberto Serra, *The detection of intermediate-level emergent structures and patterns*, Artificial Life Conference Proceedings 13, MIT Press, 2013, pp. 372–378.

[VM13] Andrew Vande Moere, *A Model for Self-Organizing Data Visualization Using Decentralized Multi-Agent Systems*, Advances in Applied Self-Organizing Systems (London) (Mikhail Prokopenko, ed.), Springer London, 2013, pp. 343–377.

[Vol14] Mark Volcic, *Energy-efficient optimization of railway operation : an algorithm based on Kronecker algebra*, Ph.D. thesis, TU Wien, 2014.

[VW71] Georg H. Von Wright, *Explanation and Understanding*, Cornell University Press, New York, 1971.

[Vya13] Valeri Vyatkin, *Software Engineering in Industrial Automation: State-of-the-Art Review*, IEEE Transactions on Industrial Informatics **9** (2013), no. 3, 1234–1249.

[Wan89] Patrick Shen Pei Wang, *ARRAY GRAMMARS, PATTERNS AND RECOGNIZERS*, Series in Computer Sciene, vol. 18, World Scientific Publishing Company, Signapore, 1989.

[WEE+08] Reinhard Wilhelm, Jakob Engblom, Andreas Ermedahl, Niklas Holsti, Stephan Thesing, David Whalley, Guillem Bernat, Christian Ferdinand, Reinhold Heckmann, Tulika Mitra, Frank Mueller, Isabelle Puaut, Peter Puschner, Jan Staschulat, and Per Stenström, *The Worst-Case Execution-Time Problem—Overview of Methods and Survey of Tools*, ACM Trans. Embed. Comput. Syst. **7** (2008), no. 3, 1–53.

[Wei62] Paul M. Weichsel, *The Kronecker Product of Graphs.*, Proceedings of the American Mathematical Society **13** (1962), no. 1, 47–52.

[WH02] Danny Weyns and Tom Holvoet, *A Colored Petri-Net for A Multi-Agent Application*, Proceedings of MOCA'02 **561** (2002), 121–141.

[Wil94] Theodore J. Williams, *The Purdue enterprise reference architecture*, Computers in Industry **24** (1994), no. 2, 141–158.

[Wit97] Ulrich Witt, *Self-organization and economics—what is new?*, Structural Change and Economic Dynamics **8** (1997), no. 4, 489–507.

[WS00] Jon Whittle and Johann Schumann, *Generating Statechart Designs from Scenarios*, Proceedings of the 22nd International Conference on Software Engineering (New York, NY, USA), ICSE '00, Association for Computing Machinery, 2000, pp. 314–323.

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

[WSJ17] Martin Wollschlaeger, Thilo Sauter, and Juergen Jasperneite, *The Future of Industrial Communication: Automation Networks in the Era of the Internet of Things and Industry 4.0*, IEEE Industrial Electronics Magazine **11** (2017), no. 1, 17–27.

[Wun12] Wilhelm Wundt, *An introduction to psychology*, (R. Pintner, Trans.) MacMillan Co., 1912.

[XWWW03] Wei Xu, Shi-Gang Wang, An-Lin Wang, and Guo-Bao Wang, *Towards an Efficient Self-organizing Reconfiguration Method for Self-reconfigurable Robots*, Journal of Intelligent and Robotic Systems **37** (2003), no. 4, 415–425.

[YC60] Marshall C. Yovits and Scott Cameron (eds.), *SELF-ORGANIZING SYSTEMS: Proceedings of an interdisciplinary conference, 5 and 6 May.*, 1 ed., Pergamon Press, New York, 1960.

[Zeh58] Johann Georg Zehfuss, *Über eine gewise Determinante*, Zeitschrift für Mathematik und Physik **3** (1858), 298–301.

[ZM16] Bernard P. Zeigler and Alexandre Muzy, *Some Modeling & Simulation Perspectives on Emergence in System-of-Systems*, Spring Simulation Multi-conference (SpringSim'16) (Pasadena, CA, United States), April 2016.

[ZPK00] Bernard P. Zeigler, Herbert Prähofer, and Tag Gon Kim, *Theory of Modeling and Simulation: Integrating Discrete Event and Continuous Complex Dynamic Systems*, Second ed., Academic Press, London, UK, 2000.

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub