GRAMMAR-BASED SET-THEORETIC FORMALIZATION OF EMERGENCE IN COMPLEX SYSTEMS

LUONG BA LINH

(B.Sc. (Hons), Ho Chi Minh City University of Technology, Vietnam)

A THESIS SUBMITTED
FOR THE DEGREE OF MASTER OF SCIENCE

DEPARTMENT OF COMPUTER SCIENCE
NATIONAL UNIVERSITY OF SINGAPORE

January 2014

Abstract

As complex systems are becoming ubiquitous and are growing, especially in terms of size and interconnectivity, the study of emergence in such systems is increasingly important. Emergence can be regarded as system properties that arise from the interactions of system components, but that cannot be derived from the properties of the individual components. Despite a long history of research on complex systems, there is still a lack of consensus on the definition of emergence. A plethora of emergence definitions hinders the understanding and engineering of complex systems. This thesis proposes a grammar-based set-theoretic approach to formalize and verify the existence and extent of emergence without prior knowledge or definition of emergent properties. Our approach is based on weak emergence that is both generated and autonomous from the underlying components. In contrast to current work, our approach has two main advantages. First, in formalizing emergence, our grammar is designed to model components of diverse types, mobile components, and open systems. Second, by focusing only on system interactions of interest and feasible combinations of individual component behavior, and degree of interaction, state-space explosion is reduced. Theoretical and experimental studies using the Boids model and multi-threaded programs demonstrate the complexity of our formal approach. The Boids model has been validated up to 1,024 birds. We also present and discuss open issues in the study of emergence, and highlight potential research opportunities.

Keywords:
Emergent behavior, multi-agent system, simulation, computational modeling

Guarantee

I undertake that all the material in this thesis is my own work and has not been written for me, in whole or in part, by any other person. I also undertake that any quotation or paraphrase from the published or unpublished work of another person has been duly acknowledged in the work which I present for examination.

Luong Ba Linh

Acknowledgement

First and foremost, I am heartily thankful to my principal supervisor, Associate Professor Teo Yong Meng, for his guidance, advice, and patience throughout my master program. He provides me encouragement and support in various ways for my best interest. I feel lucky to have such a very nice advisor.

I am grateful to Dr Claudia Szabo (The University of Adelaide), who acts as my co-supervisor. I thank her for introducing me to a promising area of modeling and simulation. I really appreciate her help, especially her feedbacks about my writing.

Besides, I thank my labmates: Le Duy Khanh, Saeid Montazeri, Vu Thi Thuy Trang, Vu Vinh An, Lavanya Ramapantulu, Bogdan Marius Tudor, and Cristina Carbunaru, to name a few. I am grateful for their friendship throughout my study, and I really enjoyed my time with them. I also want to say thank you to the other friends who shared great time at NUS with me.

Lastly, I thank sincerely and deeply my parents, who have taken care of me with great love, especially during my hard time.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

Systems with a large number of components and intricate interactions are pervasive, including natural systems, ranging from animal flocks [74] to human social systems [60], as well as sophisticated artificial systems such as power grid [17], the Internet [3], social networks [66], and large-scale distributed computer systems [62]. In these systems, the interactions of components may lead to some properties that are not derivable from the properties of individual components. These properties are often termed *emergent properties* or *emergence*. The hallmark of emergence, "not derivable from individual components", typically results in a high degree of non-linearity, making emergence too difficult to be solved using traditional analytical techniques [14]. Given an input, it is generally impossible to analytically know a priori what the expected output should be. Instead, the study of emergence has motivated the adoption of some computational techniques to model and analyze complex systems [44]. Emergence makes a system harder to analyze and design, and requires a structural formal approach for detecting and reasoning about its causes and nature [82, 84]. In this section, we introduce terminologies associated with complex systems and emergence, and the relationship between them. In the scope of this thesis,

for simplicity, we use the term emergence to refer to emergent *properties*, while other aspects of emergence such as emergent rules and emergent structures will be discussed in Section 5.2.1.

1.1 Complex Systems

Despite a long history of complex system research, the definition of a complex system is still not clear [49, 54]. Although it might be complicated to analyze and design a system, this does not necessarily make the system complex. To be regarded as *complex*, a system typically needs to possess the following characteristics [10, 44]: a large number of components, no central control nor global visibility, simple behavior rules for individual components, non-linear relationships of components, and emergent properties. A complex system usually consists of many interacting components without any central control or global visibility [44, 62]. These components interact with each other in the absence of a central controller or organizer; each component has only local knowledge about its neighborhood rather than a global view of the whole system.

A *component* is a stand-alone functional element that is defined by its input and output behavior [43]. The behavior of a component is the sequence of state changes it undergoes during a specified period of time [21]. Component behavior is characterized by a set of *behavior rules* that govern how a component acts and directly interacts with its neighbors. For example, a road traffic network includes vehicles and pedestrians that obey some movement rules to avoid collision with others and maximize the traffic flow. Although behavior rules can be paradoxically simple, interaction caused by these rules may be non-linear [44]. This non-linearity distinguishes *complex systems* from *complicated systems*. Intuitively, complex means non-independent, whereas complicated is the opposite of simple.

A component/system has properties that are anything of the component/system that can be detected. When many components come together to form a system, they, as a whole, likely exhibit emergent properties that are more than the sum of the properties of the constituent components [28]. Emergence is a crucial ingredient of complex systems. For example, an accident at a point of a road may negatively result in a long traffic congestion, which is largely known as an emergent property, involving a large number of vehicles for several hours [28].

Complex systems are often characterized using information theory. The more complex a system is, the more information we need to describe or reproduce it. The complexity of a system can be evaluated in terms of system complexity measures or design complexity measures [20]. On the one hand, *system complexity measures* capture how much information is needed to describe the system itself. *Design complexity measures*, on the other hand, relate to the design of system components and the relationships among them. Traditionally, in systems that are not complex, system complexity measures can be established analytically from the design complexity measures. This inference is not applicable to complex systems because of emergent properties that are unpredictable from the design of the system. Emergence occurs when the system shifts from one level of design complexity to another level of system complexity without any external input [16, 21].

1.2 Modeling Complex Systems

Computational modeling is a potential alternative to analytical modeling for understanding complex systems [14]. There are three main approaches of computational modeling, namely, macroscopic, mesoscopic, and microscopic [41, 62]. Differences among these approaches lie in the levels of system description at which abstraction/modeling occurs:

macro level, meso level, and micro level. At the *macro level*, also referred to as global level, details of the interactions of system components are often not concerned. The focus is to examine the behavior of a system as a whole. In contrast, at the *micro level*, also known as local level, the unit of analysis is individual components and their interactions. Each component is rigorously characterized, in terms of its local properties and how it interacts with other components. The *meso level* falls between the macro level and the micro level in the sense that the meso level deals with the unit of a group of components or the unit of individual components but at a lower level of detail compared to the micro level.

In accordance to the above levels of system abstraction, there are three main computational modeling techniques. *Macroscopic* modeling simplifies details of components at the micro level, but focuses on system management and control at the macro level. For example, Moncion et al. [59] builds a dynamic graph to represent an interaction network of components. At the micro level, there is no characterization of what behavior a component has, and the interactions of components are simply represented by weighted labeled edges. At the macro level, self-organization is largely examined and it likely forms when the mean degree of the graph increases. While its simplicity is appealing, macroscopic modeling is less powerful in getting insights of the system properties, including emergent properties, because of its simplification of microscopic details.

Mesoscopic modeling describes a system by its individual components but at a lower level of detail of components and their interactions compared to the micro level. Cellular automata [89] is a well-known representative of this approach. *Cellular automata* model dynamic spatial systems in which the environment is typically a 2D grid. Each component is located in a cell of the grid, and changes its state based on the states of its neighbors (including itself) with respect to a set of behavior rules. Moreover, time is treated

discretely. Conway's Game of Life is a widely studied example with discrete component states, deterministic behavior rules, and a synchronous state updating scheme [36]. Cellular automata have advantages such as appealing visualization, Turing-completeness [73], and programming ease. However, they are not potential in representing the relationships and interactions of components. In cellular automata, it is not straightforward to model continuous spatial relationships among components because components are assumed to be located in separate cells of the same size. Furthermore, components are typically homogeneous and simultaneously perform actions at constant time steps. This requirement of homogeneity and synchronous updating might not applicable to many systems where components are heterogeneous and autonomous.

Microscopic modeling looks at a system using a high level of detail of individual components, enabling a behavioral-based description of the system. In contrast to cellular automata, which only allow discrete environments in which an environment is divided into non-overlapping cells, microscopic modeling does not make any assumptions about the environment, i.e. the environment can be discrete or continuous. A class of microscopic modeling that has been getting significant attention in the context of complex systems is *agent-based modeling* (ABM) [41]. ABM models a system as a collection of autonomous agents interacting in an environment. Agents interact with others and make decisions on their own. One promising feature of ABM is that a system to be studied can be analyzed at different levels of description, such as individual agents or groups of agents. A high level of detail of system components offers a better understanding of the cause-and-effect of emergent properties [39]. However, ABM requires a significant amount of efforts in modeling and simulation. Fortunately, these issues are somewhat solved because of the recently relevant advances in technology: data are organized into databases at finer levels of granularity, popularity of object-oriented scheme, and increasing computational power,

among others. Another challenging issue of ABM is validation. Compared to discrete-event modeling, which tends to model the designed behavior of a system consisting of relatively homogeneous components, validation in ABM is more difficult. This can be attributed to the heterogeneity, autonomy, and emergent properties generated from interactions of agents [70, 90].

1.3 Emergence

Not all properties of a complex system are trivial; some are emergent and others are not. The Greek philosopher Aristotle stated that the whole is sometimes more than the sum of its parts, and emergence is the difference between the whole and the sum. In other words, emergence appears if "more is different" such that there are properties of a system that cannot be explained by the properties of the individual components. Starting out from philosophy, emergence eventually spread throughout several disciplines, ranging from biology, chemistry, and social sciences to computer science. Consciousness is an emergent phenomenon that is surprisingly a result of a large number of simple neurons. In chemistry, the smell of rotten eggs of hydrogen sulphide is a property that neither of its atoms, hydrogen and sulphur, possesses. Examples of emergence in social sciences are social conventions in human societies, such as shaking hands when meeting someone, and collective behavior happening in groups of people. Emergence is pervasive in computer systems, in particular in artificial intelligence. A well-known example is the emergence of patterns in the Game of Life (e.g. gliders, spaceships, and puffer trains) from simple rules [36]. We also see flocking behavior in simulated birds [74], team behavior (foraging, flocking, consuming, moving material, and grazing) in autonomous, mobile robots [5], and the formation of a "highway" created by the artificial Langton ants, from simple movement

rules [81].

Despite a plethora of ideas of emergence, we still lack of consensus on what emergence is and where it comes from. In the literature, there are four main schools of thought of emergence. First, emergence is defined as *unexpected* properties of the whole that are not possessed by any of the individual components making up the whole [7, 13]. This definition seems to be fairly broad in the sense that emergence includes aggregation properties that can be calculated by summing the properties of fundamental components at the micro level. Second, emergence is both *unexpected* and *undesirable*. In addition to being not of the system design and users' expectation, emergence should have negative effects on the system [54, 58]. This definition, however, implies that emergence is totally harmful. Third, emergence is *unanticipated* [29]. According to this perspective, emergence is something that cannot be predicted through analysis at any level simpler than that of the system as a whole, thus it is impossible to anticipate the system behavior before executing the system. Finally, emergence lacks a reductionist explanation in the sense that it cannot be derived from the individual components [52], although it is generated from the interactions between them. In contrast to the first three views, which do not mention the causes and nature of emergence, this view highlights the importance of interactions of components while describing the discontinuous characteristic of emergence from the micro level.

Possible causes of emergent properties are listed below: interactions of components, a large number of components, breaking threshold parameters, spontaneous synchronization. Emergence is not imposed from the outside; it results from the interactions of components. Interactions of components are widely accepted as the key source of emergence [44, 52]. Without component interaction, a system is simply a set of separate components acting individually, and properties of the system can be fully understood given knowledge of its components. Surprisingly, intricate interactions may originate from relatively simple rules.

The flocking behavior of birds, which has aerodynamic advantages, obstacle avoidance, and predator protection, is characterized by three simple rules [74]. Moreover, a small number of laws in rule-governed systems can generate unpredictable system configurations. For example, in traditional 3-by-3 tic-tac-toe, the number of distinct legal configurations exceeds 50,000 [44]. In addition to interaction, a large number of components may result in a very large number of legal system configurations, including those that go beyond what the designer intends. These configurations likely exhibit emergent properties. Furthermore, feedback loops between components may amplify changes in the system, thus breaking some threshold parameters such as capacity limits [52, 68]. This un-designed situation is likely the source of a new property. Examples are buffer overflows, epidemics with exponential growth (disease, fads, DoS attacks), and cascade effects that involve unanticipated chains of events (avalanche, waves at ball games, traffic jams), to name a few [34, 61, 68]. Another source of emergence is the universal tendency to synchronize actions that can also violate the threshold parameters in the system. London's Millennium Footbridge had to be closed on its first day because of "unexpected excessive lateral vibrations" that resulted from an unexpected synchronization between the footfalls of pedestrians and the fluctuation of the bridge [26].

Everything has advantages and disadvantages; and emergence is not an exception. Indeed, the literature is moving from considering emergent properties as only unexpected [14] to both desired and undesired [49]. The notion of "unexpected" makes the study of emergence ambiguous in the sense that emergence is in the eye of the beholder. What is a wholly unexpected property from one view may be obvious from another. To avoid the dependence on the observer, emergence is considered from the perspective of its importance, i.e. desired or undesired. On the one hand, emergent properties can be *desired* such that they confer additional functionalities on the system [31]. Consequently, users adapt

these functionalities to support tasks that designers never intended, making the products more competitive. Some artificial intelligence computer applications, for example, utilize emergent phenomena to model collective animation of a group of entities. Additionally, emergence sometimes appears in the form of self-organization that transforms the system from disorder to order, thus reducing the system complexity [21]. The ability to engineer emergence makes a system more scalable and robust. On the other hand, due to its unpredictable nature [76], emergence makes a system less credible and harder to analyze, design, and control. In fact, it is difficult to anticipate what we have never seen before. According to Dyson [29], emergent behavior cannot be predicted through analysis at any level simpler than that of the system as a whole. Unforeseeable and unexpected failures [58, 86] and security vulnerabilities [37] are examples. The main difficulty is to predict this sort of emergent properties without prior knowledge of them. The problem becomes challenging if the properties are substantially different from the past properties.

1.4 Objective

Given the importance and increasing attention on emergence from various research fields due to the increasing demand on complex systems [12, 49, 58], there is a need for detecting and reasoning about its cause-and-effect to make systems more credible and robust, and to advance our understanding of emergence. It is important to *detect* undesirable phenomena as soon as possible to minimize their potential negative consequences. Despite a long history of research on complex systems, most studies focus only on post-mortem observation of emergence of an available system, rather than on detecting emergent properties on the fly. This is because it is too difficult to formally define emergence [72]. *Reasoning* of emergence, on the other hand, is even more challenging, but more appealing than detecting it.

The system properties at the macro level can be far from the properties of its components at the micro level due to interactions of the components. Reasoning of the cause-and-effect of emergent properties is still in its infancy.

The study of emergence includes several challenges: lack of consensus on emergence definition and an increase in the size and complexity of systems. There are different perspectives of emergence [84], including observer-dependent [80], and others are associated with theories in specific disciplines [35, 45, 85]. Although there are observer-independent definitions that are operational and can be implemented, the computational simulation suffers from increasing state-space explosion, especially when problem size increases and the connectivity between components becomes non-trivial.

The objective of this thesis is to formalize emergence properties in complex systems. This formalization comprises two main elements: a formal definition of emergence, and a way to detect or identify emergence. The former specifies what emergence is and the latter explains how emergence is exposed. The formalization unifies different emergence concepts into a single formal operational view, at least with respect to the perspective of science, in particular computer science. To be operational, emergence should be defined in a way such that the mechanism for detecting emergence can be implemented, and the state-space problem is mitigated.

1.5 Contributions

The key contributions of this thesis are:

1. **Grammar-based Set-theoretic Approach to Determine Emergent Property States**

   We extended Kubik's approach to determine emergence in complex systems. Unlike

Kubik's approach, which regards emergent properties as system states, we consider these system states as emergence, an emergent property state set, from which emergent properties can be deduced. Given a system, emergence is defined as a set of system states that arise from the interactions of the components of the system, but cannot be derived by summing the state of individual components. We also extended Kubik's approach to consider different types of components and open systems. A system is modeled as a multi-agent system of interacting agents of different types, including mobile agents. The set of emergent property states is the difference between: the set of system states reachable from the initial state due to interactions of agents, and the set of all system states obtained by summing state of individual agents. We applied and validated the proposed approach to derive bird flocking states and deadlock in multi-threaded programs.

2. **Reduction of Search Space**

We proposed to reduce the state space to be searched in two aspects: the definition of emergence and the derivation of emergent property states. Emergence is considered with respect to the system designer's interest, i.e. the system model, rather than to the real system. The multi-agent model of the system abstracts only parts of the system of interest, and ignores details that are not of the designer's interest, thus constructing a smaller state space. Furthermore, relied on the observation that the state space of summing individual components is the key source of the state-space explosion problem, but it does not contribute much to the derivation of emergence, we use degree of interaction of agents as an emergence criterion, thus eliminating the unnecessary calculation of the sum. By associating agent interaction with system state, interaction degree is defined as the difference between system states. This idea enables a measurable and

computational manner of studying emergence.

1.6 Thesis Organization

The outline of this thesis is presented as follows.

Chapter 2 - Related Work

We present different perspectives of emergence, including philosophy, natural and social sciences, and computer science. Our conclusion is that a scientific study of emergence should be observer-independent, rely on agent-based simulation, and enable the reasoning of the causes and effects of emergence. We also review several classifications of emergence and propose a more comprehensive classification with respect to the feedback from the macro level to the micro level. Three state-of-the-art formalizations of emergence: *variable-based*, *event-based*, and *grammar-based* are discussed. Contrary to variable-based and event-based approaches, grammar-based approach does not require prior knowledge of emergence. Our proposed approach extends and addresses many limitations of the grammar-based approach.

Chapter 3 - Grammar-based Set-theoretic Approach

We present our strategy to overcome limitations of the current grammar-based emergence formalization. The main aim is to broaden the application domain and mitigate the state-space explosion problem. Compared to current methods, our approach can deal with more general systems in which components have different types, are mobile, and can join and leave the system over time. The proposed approach considers only the behavior rules of interest and eliminates the computation of system states that will never happen in practice, thus reducing the system state space to be searched. We illustrate how to determine the set of emergent system states that expose flocking phenomena of a group of birds of two

types. The experimental results give us intuition of the state-space explosion problem.

We also propose a method to further mitigate the state-space explosion problem by avoiding the calculation of the sum of states of individual components. Instead of determining the difference between the whole and the sum explicitly, we calculate the intersection between the whole and the sum without taking the sum into consideration. The difference between the whole and the obtained intersection is the set of emergent property states. This method relies on the degree of interaction of components, which is measured as difference between system states. By applying the method, experiments are done up to 1,024 birds.

Chapter 4 - Example: Deadlock Emergence in Concurrent Programs

To minimize the critical drawback of our approach that emergent property states are relative to the model of the system, multi-threaded programs are considered. In contrast to the Boids model, a multi-threaded program is a more concrete specification of a problem provided by a user. Given a multi-thread program, the main goal is to detect all (emergent property) states that arise from the interleaving interactions among threads. As we will see in this chapter, our approach detects deadlock states.

Chapter 5 - Conclusion and Future Work

We summarize the key contributions of this thesis and discuss some of the major open issues, including the consensus on the definition of emergence, state-space explosion, emergence reasoning, and emergence validation.

Chapter 2

Related Work

2.1 Emergence Perspectives

Despite a long history of emergence research, there is no agreement on a definition of emergence. Emergence is studied in both *philosophy* and *science*. Scientific studies of emergence involve natural and social sciences, and computer science.

2.1.1 Philosophy

In philosophy, the key concept of emergence is *surprise*. The Greek philosopher Aristotle puts forward a seminal idea of emergence: "the whole is more than the sum of its parts". The main implication of this idea is that emergence cannot be defined as simple consequences of the underlying parts; it is something *surprising* [80]. The surprise comes from the discontinuity between the observer's mental image of the system's design and the observation of the system behavior [75]. Surprising, however, is *observer-dependent*. Emergent properties are subjective product of both the unexpected behavior of complex systems and the *limitations* of the *observer's knowledge* [49]. Certain strange phenomena

cannot be detected or understood with a given set of tools and knowledge, but can be detected or understood by exploiting newer tools and theories. Furthermore, the key in understanding emergence is the observer rather than the system itself in the sense that a phenomenon emerges when the observer begins to consider it at a certain scale [16]. For example, an observer may not detect the structure of a city when walking in the streets, whereas a satellite photograph of the city could reveal it [16]. The dependence on the eye of the beholder makes the root of emergence vague.

2.1.2 Natural and Social Sciences

Authors from natural and social sciences criticize the idea of *limitations of our knowledge* as it implies that we are scientifically unable to study emergent properties with the current theories and technologies. Another problem of this idea is that it is based on a temporary lack of knowledge of the observer. Instead, emergence should be *observer-independent* [25]. According to Abbott [2], an observer's surprise should be not associated with how we understand a problem.

Emergent phenomena seem to be everywhere in nature and society [62]. Flocks of birds, ant colonies, and schools of fish, among others, are examples of natural phenomena that cannot be reduced to the properties of individuals. Bird flocking, in particular, is frequently studied in the context of emergence [19, 74, 83, 84]. At the micro level, a bird only knows the position and velocity of its neighboring birds. The movement of each bird is governed by three simple flying rules: (1) separation - steer to avoid crowding neighbors, (2) alignment - steer towards average heading of neighbors, and (3) cohesion - steer towards average position of neighbors. At the macro level, a group of birds tends to form a flock, which has aerodynamic advantages, obstacle avoidance, and predator protection. These flocking properties are not obviously traced back from the individual birds with local

knowledge about their neighborhood and the flying rules.

Social sciences attempt to answer the question of how human behavior arises from the interactions of participants. Collective behavior of human, such as in stock markets [23], social networks [66], and condense crowds [51], to name a few, has been investigated for a long period [15]. Lane formation of pedestrians in shopping malls is another example [51]. Pedestrians follow three simple movement rules: (1) try to stay close to the shortest path between the source and the destination, (2) avoid collisions with obstacles and other pedestrians, and (3) avoid sharp and rapid changes of direction. The pedestrians as a whole, however, incidentally move in lanes.

Natural and social sciences mainly aim to understand and explain emergent properties of complex systems in reality. Two main theories used for understanding emergence are self-organization [85] and hierarchy [8]. *Self-organization* is a proof that individual autonomy and global order can coexist. Emergence is defined as the formation of order from disorder with greater coherence between components due to self-organization. When components are highly connected, i.e. connected to many others, degree of regularity among agents tends to increase, and the system likely generates certain form of structures or patterns, for example spatial patterns, or patterns in the form of repeated sequences of behavior. In fact, the notion of self-organization conforms to the idea that complex systems are neither completely random nor completely ordered [13, 42, 53]. Instead, complex systems are somewhere in between, being random and surprising in some aspects while predictable in others. Figure 2.1 shows the relationship between coherence between components and the probability that a system exhibits emergence in terms of structures or patterns.

In *hierarchy* theory, emergence is the difference between observing and describing a system at multiple *levels of abstraction* (observation). Typically, emergence and hierarchy

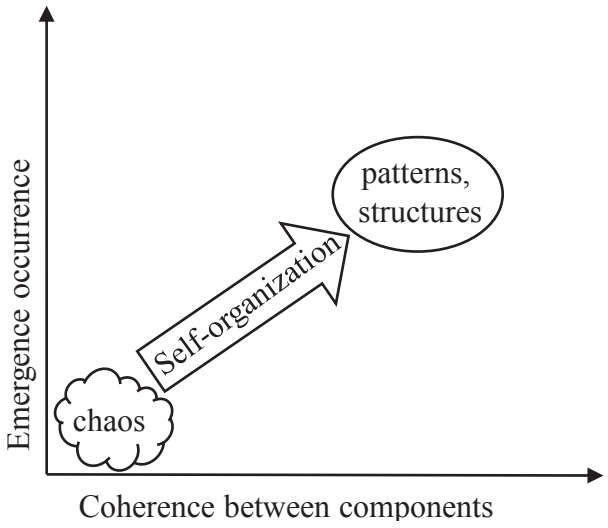

Figure 2.1: Self-organization

of observation are inseparable. A hierarchy of order N is given by:

$$S^N = R(S^{N-1}, Obs^{N-1}, Int^{N-1}, S^{N-2}, \ldots) \tag{2.1}$$

where S^N is the collection of components at level N, Obs^N is the observation mechanism for measuring the properties of components at level N [8, 50], and Int^N is the interactions of components at level N. The most common paradigm of hierarchy of observation is micro-macro. A macro level in one context might be a micro level in another [8]. Ryan [77] defines micro-macro relationship in terms of scope and resolution. The greater a scope is, the more accuracy we have to sacrifice. A property is a macro property of another if it has a smaller scope, a higher resolution, or both.

2.1.3 Computer Science

While emergence has been widely observed in natural and social sciences, it has been largely ignored in computer science [14]. Contrary to natural and social sciences, which focuses on understanding and explaining the world, computer science, as primarily an engineering science, concentrates on designing and optimizing engineered systems. In the

context of emergence, computer science attempts to detect, validate, and reason about the causes and nature of emergent properties in order to make systems more reliable, scalable, and robust. This aim is based on analyzing the system components' specification and the interactions of the system components. This analysis typically requires computational modeling, i.e. simulation, because of the high complexity of the interactions. Furthermore, as the simulation is done for a model instead of the real system, study of emergence in computer science perspective is relative to system model.

Emergent phenomena abound in computer systems [12, 24, 33, 44, 49, 58, 71]. For example, the distribution of links in the World Wide Web scales according to a power law in which a few pages are linked to many times and most are seldom linked to [3]. A related property of the network of links in the World Wide Web is that almost any pair of pages can be connected to each other through a relatively short chain of links [4]. Another example is priority inversion in operating systems. In priority-based scheduling, which assigns processes with a fixed priority, a high priority process can be blocked due to a resource held by a lower priority process. The unpredictable nature of emergence makes it more interesting and increasingly important in software engineering, especially in systems of systems that exploit emergence to achieve adaptability, scalability, and cost-effectiveness [67].

System complexity is increasing in terms of size, connectivity, and geographic distribution [6]. This growth makes emergent properties more common in reality. The undesired and unpredictable effects of emergent properties demand a formal and practical approach to understanding and validating emergence. However, traditional analytical techniques for addressing complex systems with non-linear processes are not readily available [12, 44, 48]. To support this view, Hyotyniemi [48] proposes a recursive non-linear function for the kernel of system complexity. The result of this function is argued to be intractable using

mathematical techniques when iterations cumulate. Instead, it requires some computational technique to observe and analyze the system gradually. Computational modeling, i.e. simulation, is considered to be a potential solution for the formal study of emergence [12, 27, 47]. For example, a practical method to check whether the so-called R pentomino, which is a five-cell pattern in The Game of Life, has an upper bound is simulation. By simulation, after 1,103 time steps we see that the R pentomino settles down to a stable state that just fits into a 51-by-109 cell region [13]. Furthermore, according to Darley [27], simulation is regarded as the most efficient way to predict emergent properties. A system is emergent if and only if the amount of computation without simulation needed for understanding the system is not smaller than the optimal amount of computation needed to simulate the system. Hovda [47] quantifies emergence in the terms of the amount of simulation needed to derive a fact.

Agent-based modeling (ABM) is believed to be an appealing approach to model and simulate complex systems exhibiting emergence [44]. ABM, as discussed in Section 1.2, provides a detailed description of the system, including its components and their interactions, thus facilitating the detecting and reasoning of the cause-and-effect of emergence. Moreover, ABM is relevant to complex systems in the sense that both rely on autonomous individual objects interacting with each other. The increase of the popularity of object-oriented paradigm and computational power fosters the potential of ABM in the field of emergence. Table 2.1 summarizes the three perspectives.

2.1.4 Summary: Observer-independent Perspective

The science of studying complex systems can be classified into two broad streams: traditional science that does not deal with emergence and science of emergence that handles emergent properties [63]. Table 2.2 presents a comparison between them.

Perspective	What	How
Philosophy [14]	surprise - limitations of our knowledge	observer at correct scale
Natural & Social Sciences [8, 25, 34]	observer-independent	statistical techniques, self-organization, hierarchy
Computer Science [38, 52]	arise from component interaction, relative to model	agent-based simulation

Table 2.1: Emergence Perspectives

Criteria	Traditional Science	Emergence Science
Domain	simple systems (reductionism, focus on components)	complex systems (holism, focus on interactions)
Goal	prediction	understanding, explanation
Analysis	top-down	bottom-up, different spatial and temporal scales
Tools	mathematics, measurement	agent-based modeling, simulation

Table 2.2: Traditional Science and Emergence Science

First, traditional science focuses on simple systems in which the properties of the whole system can be reduced to the properties of its components. This reductionism is due to linear interactions of components, and can be studied in terms of traditional analytical techniques. As a result, traditional science focuses on individual constituent components. In contrast, the science of emergence looks at complex systems that are non-deterministic and considers a system as a whole rather than at the level of individual components. Unlike traditional science, which studies simple cause-effect relationships, emergence science assumes that complex effects arise from simple causes through non-linear interactions of components. It is therefore not surprising that emergence science focuses on interactions of components. Second, the main aim of traditional science is *predicting* the behavior of the system under study. In contrast, emergence science is a new field of science whose goal is to *understand* and *explain* how non-linear interactions of components give rise to the holistic behavior of the system. Third, due to reductionism, traditional science usually ap-

plies top-down strategy to break down a system into separate components. The properties of the whole system are then derived from the properties of its constituent components. The top-down approach, however, cannot be applied to the study of emergence. Instead, a system exhibiting emergence is usually considered bottom up, or in other words from the components at the bottom to the holistic system at the upper level. For example, water is a bottom up emergent property of hydrogen and oxygen. Furthermore, the science of emergence looks at understanding indirect effects, both in space and in time, at different scales. Local interactions of components and with the environment, may cascade in a non-trivial way across different levels of space, ranging from local to global, as well as different levels of time, ranging from a few to many simulation steps [36]. Finally, systems to be studied in traditional science are usually represented in some mathematical form that is then solved to predict the system behavior. If the mathematical theory cannot be proved, some experimental measurements are carried out to strengthen the theory. Complex systems in science of emergence, as discussed earlier, are too sophisticated to be expressed using mathematical methods, instead should be modeled as multi-agent systems and observed through simulation [44].

2.2 Emergence Taxonomies

2.2.1 Current Taxonomies

In correspondence with several different perspectives of emergence, there are several types of emergence [11, 13, 18, 34, 38]. Chalmers distinguishes between *weak* and *strong* emergence [18]. Weak emergence is deducible but unexpected from the laws of the low-level domain, while strong emergence is not deducible even in principle. Bedau describes deducible feature of weak emergence in terms of derivability by simulation [13]. In addition

to strong and weak emergence, Bedau also introduces the notion of nominal emergence. As a general definition of emergence, *nominal* emergence is simply macro level properties that cannot be found at the micro level. Further, Bar-Yam distinguishes between four types of emergence: *Type 0*, *Type 1*, *Type 2*, and *Type 3* [11]. The first three types roughly correspond to nominal, weak, and strong emergence respectively. Type 3 defines emergent properties regarding the interaction between the system and the environment. Similarly, Fromm divides emergent properties into four categories: *simple*, *weak*, *multiple*, and *strong* based on different types of feedback from the macro level to the micro level [34]. Simple emergence contains no feedback. Weak emergence has positive or negative feedback, while multiple emergence has both positive and negative feedbacks. Strong emergence is similar to that in Bedau's taxonomy. Gore proposes an emergence taxonomy based on three dimensions: *reproducibility*, *predictability*, and *temporality* [38]. Behavior can be classified to be deterministic or stochastic, predictable or unpredictable, and materializing or manifested.

2.2.2 Downward Causation-based Taxonomy

Based on the classifications above, we introduce a comprehensive view of emergence with respect to downward causation. *Causation* is the relationship between cause and effect. The whole is generated from the parts through *upward causation* (UC), but the parts, meanwhile, are somewhat affected by the whole through *downward causation* (DC) [13]. For example, cows interact directly with each other to form a herd (UC). The cows also change the state of the environment such that they create a track when moving. The presence of the herd and the track reinforces the tendency of moving in a herd of the cows (DC). UC and DC define the mutual relationship between the macro and the micro level. The causation loop between UC and DC, i.e. UC from the micro level to the macro

level and DC in the converse direction, makes emergent properties irreducible from the individual components.

The concept of DC, however, is contrary to reductionism, which reduces a complex system to the interactions of its components. Thus, some authors regard DC as a defining ingredient of emergence [12, 65]. Chalmers, corresponding to the notions of weak and strong emergence, defines *weak* and *strong* DC [18]. The former is the causal impact of the macro level on the micro level that is unexpected. The latter is not deducible even knowing the governing laws at the micro level. *Positive* DC and *negative* DC are defined in [34]. Positive DC reinforces UC while negative DC reduces the impact of UC on the system properties.

We extend the existing classifications with three types of DC: positive, negative, and complex. *Positive* DC amplifies UC and drives the system out of equilibrium, i.e. unstable states [69]. Systems with positive DC are sensitive to initial conditions in the sense that small changes in initial conditions can lead to very different overall system behavior. Second, *negative* DC weakens UC and stabilizes the system in equilibrium [74]. Lastly, *complex* DC makes the underlying components change their behavior rules in reaction to a changing environment. For example, living systems are known to have evolutionary processes in which living entities evolve, for example through mutation, to survive and expand in a new condition.

Figure 2.2 shows a taxonomy of emergence, consisting of simple, weak, and strong emergence, based on downward causation. In *simple* emergence, DC is too weak (approximately zero DC) to have significant effect on the underlying components. A property is simple emergent if it is not exhibited by any underlying components. For example, a large number of entities in aggregation are characterized by statistical quantities, which are inapplicable to the constituents. Gases, for instance, have volume and temperature,

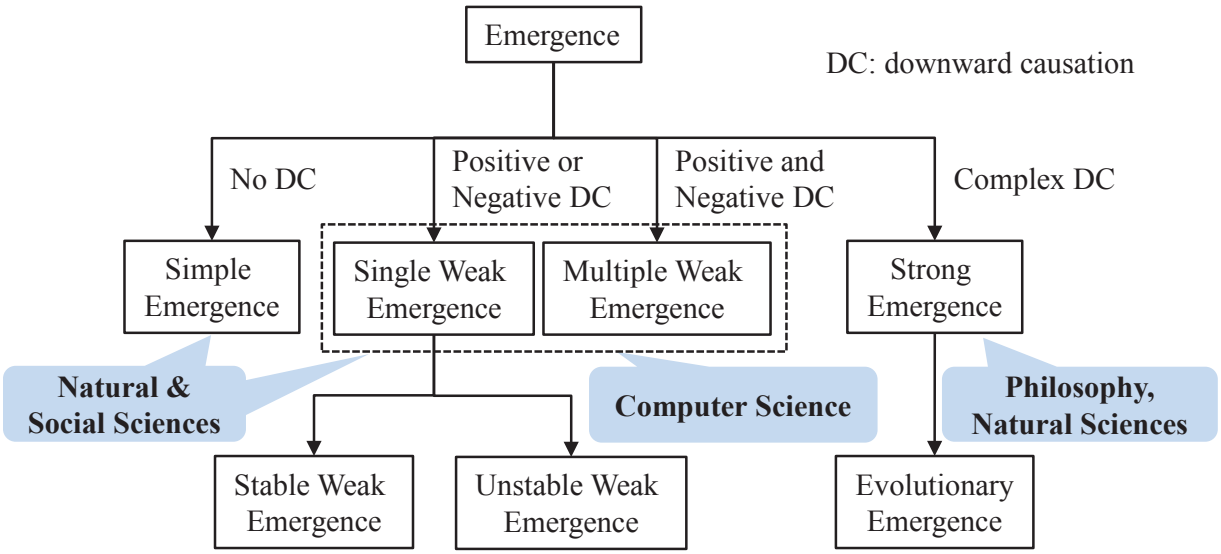

Figure 2.2: Downward Causation-based Taxonomy of Emergence

which are not possessed by gas particles. Systems with simple emergence usually consist of loosely coupled and equal components whereby a component's state is independent of the state of other components, the whole system, and the environment. In these systems, component behavior is somewhat random in the sense that the components are largely uncorrelated or their relationships are too chaotic to describe explicitly. Simple emergence is mainly studied in natural and social science using theories from physics and chemistry.

Weak emergence involves positive or negative DC (single weak emergence), or both (multiple weak emergence). Weak emergence is the notion of emergence that has gained immense attention in science [13, 18, 52, 84]. For one thing, this notion is philosophically acceptable because it meets the theory of cause and effect. A weak emergent property is both generated (through UC) and autonomous from the properties of the underlying components. Autonomy is expressed in the sense the causation loop between UC and DC, i.e. UC affects DC and DC in turn affects UC, gradually changes the effects of UC, thus making the macro level discontinuous and irreducible from the micro level. For another, weak emergence does not require an introduction of new fundamental laws to study [18].

Instead, weak emergence can be understood using the existing laws but with further levels of description and explanation [18]. Finally, the concept of weak emergence is closely associated with computation modeling or simulation, which is widely used in science. A macro level property is *weakly* emergent if it can be derived from the micro dynamics but only by finitely long simulation [12, 48].

Two types of single weak emergence are stable and unstable. In *stable weak* emergence, negative DC weakens positive UC to keep the system in equilibrium such that there is a balance between diversity, autonomy, and randomness through UC and unity, self-organization, and order through DC. For example, ants have different unique contexts (diversity) and make their own decision (autonomy) to explore every direction in a constantly changing environment (randomness). Yet they have a collective goal (unity), e.g. reaching the same destination, and move in a colony (self-organization) by following their own pheromone trails (order). In *unstable weak* emergence, positive DC amplifies positive UC, leading the system to unstable states. For instance, inflation keeps the price of goods and services increasing: high prices of goods and services increase the cost of living, high costs of living increase wages, and high wages increase high prices of goods and services.

Multiple weak emergence rests on the balance between positive and negative DC. For example, stock market has a balance between UC that makes the market unstable and DC that pulls the market back to equilibrium. When stocks are rising, investors tend to buy; the stocks rise further, thus the market becomes unstable. At some point, the stock market is highly unstable, and investors believe that the market is likely to fall, they stop transactions, taking the market back to a more stable state. *Strong* emergence is due to complex DC that changes the behavior rules of the underlying components to accommodate external influences. Strong emergence is considered non-deducible, even in principle, from laws of the micro level. Instead, this notion of emergence is most common

in philosophy and natural sciences. From the philosophy perspective, strong emergence is due to intricate interactions of components, limitations of the observer's knowledge [49], and the scale and level of abstraction under which the system is observed [16]. Limitations of the observer's knowledge imply that strong emergence requires the introduction of new fundamental laws to explain it [13, 18]. In natural sciences, typically in biology, strong emergence usually involves very large jumps in complexity [34] and some kind of evolutionary processes. *Evolutionary* emergence has the highest degree of complexity in the sense that components are capable of learning in order to adapt to new conditions and evolve [34]. Typical examples are biological systems. Life, in particular, is an evolutionary emergent phenomenon of genes, genetic code, and nucleic/amino acids.

2.3 Emergence Formalizations

The demand of understanding and engineering complex systems exhibiting emergent properties, and the lack of consensus on emergence definition have attracted immense interdisciplinary interest for formalizing emergence [22, 44, 52, 82]. Formalization enables comprehensive analysis of complex systems, and thus advancing the reasoning of the cause-and-effect of emergent properties. There are three main approaches of emergence formalization: variable-based, event-based, and grammar-based, depending on the kind of emergence analysis they employ: post-mortem or on-the-fly analysis. *Post-mortem* analysis refers to detecting and reasoning about emergence by observing system states. This analysis needs prior knowledge of emergence from experts. *On-the-fly* analysis, on the other hand, focuses on detecting and validating emergence when it happens, thus does not require knowledge of emergence to be defined in advance. Table 2.3 shows a comparison among the three formalization approaches.

Formalization	Prior Knowledge	Analysis
Variable-based [32, 57, 78]	required	post-mortem
Event-based [22]	required	post-mortem
Grammar-based [52]	not required	on-the-fly

Table 2.3: Emergence Formalizations

2.3.1 Variable-based

In *variable-based* methods, one variable is chosen to model the attribute space that describes the state of the observed system. This variable is then used to detect and measure emergent properties [64]. Usually, emergence is measured using probability and information theory [32, 57, 78]. For example, the change of the center of mass of a group of birds may indicate the formation of flocking behavior.

Many variable-based efforts [35, 45, 57, 88] deploy Shannon entropy [79], which measures the uncertainty and unpredictability of a system with respect to one attribute. The key idea is that emergence most likely occurs as the system self-organizes and exhibits some kind of patterns or structures, thus resulting in lower entropy. Mnif and Muller-Schloer [57] introduce emergence as the difference between the entropy at the beginning and at the end. A system is said to exhibit emergence if the entropy difference is positive, i.e. the entropy value decreases in the end. Despite simplicity, Shannon entropy only deals with a single attribute with discrete values. To address systems containing many attributes with continuous values, Fisch et al. [32] define multivariate divergence, "an unexpected or unpredictable change of the distribution underlying the observed samples", using Hellinger distance [32] as an emergence measure. This measurement suffers from expensive computation of density functions and high user intervention. Inspired by the idea that weak emergence is both dependent upon and autonomous from the micro level causal factors, Seth [78] proposes G-emergence as a measure of emergence based on two other non-linear time series quantities: G-causality and G-autonomy, which compute the dependence and

autonomy of a variable with respect to a set of other variables respectively. A macro variable M is *G-emergent* from a set of micro variables m if and only if M is G-caused and G-autonomous with respect to m. However, a set of variables must be defined and the computations of G-causality, G-autonomy, and G-emergence are expensive. One of the most significant drawbacks of variable-based emergence formalization is that it requires prior knowledge of emergence to define a variable manifesting the system behavior. This variable needs to model the whole system rather than pertain to a specific part or a group of parts.

2.3.2 Event-based

In *event-based* approaches [22], emergence is defined as complex events that can be reduced to a sequence of simple events. An event is a state transition occurring at a particular level of abstraction. A *simple* event results from the execution of a single state transition rule. A *complex* event is either a simple event or two complex events satisfying a set of constraints with respect to each other. A constraint could be a temporal, spatial, or component or variable constraint. First, a temporal constraint defines the temporal relationship between two events. Second, a spatial constraint defines the space within which an event should occur relative to another. Finally, component or variable constraints define the relationship between variables or components of the two events. Similar to variable-based approaches, event-based approaches need the formalism of event types and emergent behavior to be defined in advance, thus can be applied only for the post-mortem analysis of emergence.

2.3.3 Grammar-based

Kubik [52] avoids the requirement of prior knowledge by defining emergence using grammar systems. The grammar-based approach combines the idea of emergence relative to model [75], Bedau's notions of micro-macro relations [12], and agent-based modeling approach to emergence advocated by Holland [44], to move towards a more formal theory with well-defined meaning for farther study of emergence. The key idea is to determine a set of system states (L_ξ) that result from the interactions of system agents and cannot be produced by summing their individual states, thus formally describing systems properties that are more than the sum of its parts.

$$L_\xi = L_{whole} \setminus L_{sum} \tag{2.2}$$

where L_{whole} denotes the set of system states when the agents act as a whole, and L_{sum} denotes the sum of individual states of all agents when they act individually. The grammar-based approach does not make any assumptions about the knowledge of emergence, and moves much closer to a concept where emergence is observer-independent. Emergence arises out of the interactions of components and can be computationally determined in terms of system states without the presence of an observer. Furthermore, observer-independence is the core idea behind computational approaches to emergence [44].

However, Kubik's work has a number of limitations: (1) suffers from state-space explosion (L_{sum}), (2) cannot model agent types, (3) does not support mobile agents, (4) only deals with closed systems with fixed number of agents, and (5) needs further work for the summing operator. First, L_{sum} contains all permutations of individual states of agents, including those that never exist in practice due to constraints among agents. Such constraints are usually invariants that hold for the entire system life. For example, in a one-way single lane road, a car A will never take over another car B in front. In other

words, the system state in which car A is in front of car B is invalid. A large number of invalid, unreachable system states will lead to the state-space explosion problem, even with a small number of agents [84]. Additionally, the example used in Kubik's paper, The Game of Life, is a simple one, in which all agents are identical (have the same set of possible states, the same set of state transition rules), stationary (always stay at the same cell). Finally, L_{sum} is calculated using the superimpose operator that, according to the author [52], is chosen because there is no better choice. Therefore, further work for a convincing explanation of the superimpose operator or for a better way of summing is required.

We [84] addressed the first four limitations. To reduce the system state space, we ignore the set of invalid permutations of individual states in L_{sum} based on constraints among agents defined from the system specification, and propose a tighter notion of L_{whole}. For L_{whole}, instead of dealing with the whole set of states of real system to be studied, this work considers only a subset of it with respect to the system designer's interest. Ideally, if we knew all rules defining a system, we could completely understand and explain it. However, in practice, this is not always the case. In fact, a system is typically modeled as an abstract approximation of the real system with respect to mainly the system designer's interest, and other things such as computational power and simulation time constraint. This paper also considered a general grammar-based formalization for the system that supports agent types, mobile agents, and open systems (agents may enter and leave the system over time). For agent mobility, an agent may have attributes that are closely related with its location such as position, speed, moving direction, and so on.

2.4 Summary

Emergence is gaining more interest from researchers in many fields, from philosophy to science. Each perspective investigates emergent phenomena with different views and approaches. Philosophical studies explain the unpredictability of emergence to the limitations of knowledge of observers while scientific perspectives, including natural and social sciences, and computer science believe that emergence properties are observer-independent, i.e. a feature intrinsic to the system. Computer science perspective, in particular, emphasizes the key role of the interactions of components in the presence of emergent properties. This perspective also asserts the great importance of agent-based simulation regarding the system model to detect, validate, and reason about emergence. A quite complete taxonomy of emergence based on downward causation is presented. This taxonomy contributes to consolidate almost all other concepts of emergence in the literature. It also shows the mapping between perspectives and categories of emergence. Based on this mapping, we know what notion of emergence we should take out and what theories or techniques we should use to study emergence in a specific perspective.

Formalization is probably the most significant but difficult part in the study of emergence. Efforts are mainly variable-based, event-based, or grammar-based. While the first two have to describe emergence in advance, grammar-based method, on the other hand, does not. It exploits grammars to model the system to be studied and to expose emergence. The outcome of the approach is a set of emergent property states that is simply the difference when considering the interactions of components and when not. Eliminating the posteriority drawback makes grammar-based formalization promising for automatically detecting, and therefore, validating emergent properties.

Chapter 3

Grammar-based Set-theoretic Approach

Emergence formalization is an important step towards understanding and reasoning about system behavior. This chapter presents a grammar-based set-theoretic approach for formalizing system components and their interactions, and deriving emergent property states. The set of emergent property states is a source from which emergent properties could be deduced. This chapter also presents a technique for reducing the state space accounted for the calculation of emergent property states using degree of interaction between components. The application and validation of our approach are discussed when determining bird flocking and deadlock in multi-threaded programs.

3.1 Approach

The objective of our approach is to determine a set of emergent property states (L_ξ) from which emergent properties can be deduced. This is a multi-step process, starting with the system to be studied, and come out with a set of system states that potentially exhibit

emergent properties of interest (as shown in Figure 3.1). Given a problem to be studied,

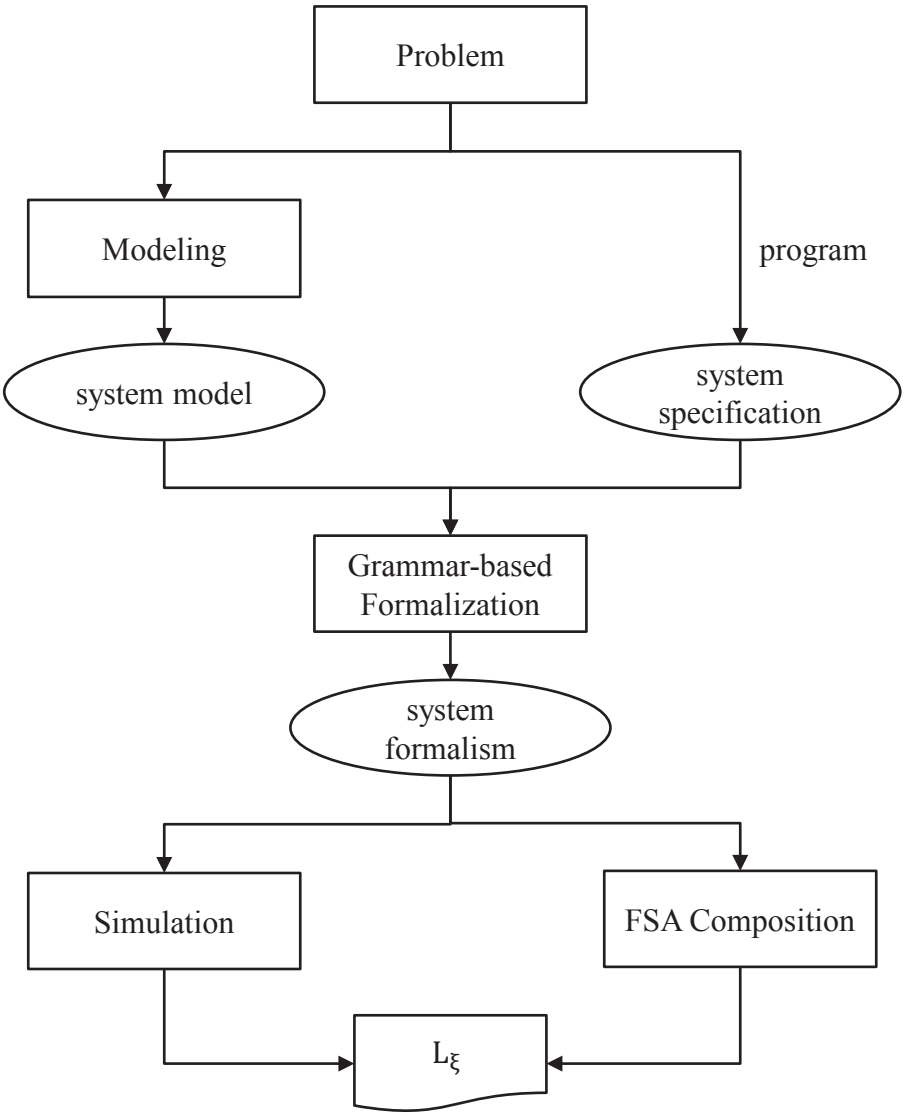

Figure 3.1: Grammar-based Set-theoretic Approach to Determine Emergent Property States

we first need to model it or start with its specification. The model is an approximate abstraction of the problem in relation to the modeler's interest and purpose. On the other hand, the specification is often a concrete program provided by a user. Whichever the case, either the model or the specification is input to the next step, grammar-based formalization, to formalize the system using context-free grammars. The output of this

step, a grammar-based formalism is used to determine the set of emergent property states. We consider two ways of deriving L_ξ: (1) simulation and (2) Finite State Automata (FSA) composition. The former is considered when the system behavior is deterministic in the way that the simulation has only one possibility to move forward in the next step. In addition, components are usually assumed to behave at the same time to transfer the system to a new state. In Section 3.4, the Boids model that imitates the flying of a group of birds is such a system. Given a system state and a set of rules for the flying of birds, the next system state can be deterministically derived. In contrast, FSA composition is used to calculate L_ξ when the system behavior is non-deterministic. FSAs model components that asynchronously interact with others. At a point of time, any of the components can do activities and trigger the system to change to a new state. A multi-threaded program (as seen in Chapter 4) allows all possible execution orders of interleaving threads. With this arbitrary scheduling, different simulations starting with the same configuration may give different outcomes. Therefore, simulation is not efficient and feasible to calculate a huge number of reachable states from the initial system state. Asynchronous composition of FSAs overcomes this problem as discussed in the next section. Whichever the case, we finally obtain a set of emergent property states L_ξ, which is the foundation for further work. For example, given the specification of a known emergent property, we can retrieve all system states that possess the concerned property. We can also trace back to the execution paths leading to this emergent property and gain insights into the causes that mainly contribute to the emergence of the property.

Our approach extends Kubik's grammar-based approach [52] to determine emergent property states that are due to the interactions of components but cannot be derived from summing states of individual components. In Kubik's work, the set of emergent property

states is defined as the difference:

$$L_{\xi} = L_{whole} \setminus L_{sum} \tag{3.1}$$

where L_{whole} describes all possible system states due to agent-to-agent and agent-environment interaction, and L_{sum} is the sum of all individual agents' states, without considering agent interaction. Given an initial system configuration, the system is simulated until it arrives in a state that has already appeared before, and L_{whole} is the set of all distinct states appeared. On the other hand, L_{sum} is derived by superimposing states of individual agents using a superimpose operator. L_{ξ} contains the set of system states that are in L_{whole} but not in L_{sum}. This broad perspective of emergence leads to state explosion when determining L_{sum} and L_{whole}. This is because all possible combinations of individual agent states are considered following a defined superimposition operator, without including system-defined rules.

We propose a new perspective that significantly reduces the state space for L_{ξ}. Firstly, it is important to highlight here that while Kubik refers to the difference $L_{whole} \setminus L_{sum}$ as emergence, we refer to this set as the emergent states set, because it is the set from which emergent properties can be deduced. Given this set, if we have knowledge of a certain emergent property, for example its definition or the characteristics specifying it, we can pick up all system states that possess the property and analyze its cause-and-effect. Emergent property states that possess some emergent property that we already know are called *known* emergent property states. Otherwise, they are called *unknown* emergent property states. Secondly, we observe that the size of L_{whole} is dependent of the number of interactions and state transition rules defined by a modeler, and a subset of interest (to the modeler) from all the rules in a given system. This would be the case, for example, when we consider different kinds of rules in modeling a flock of birds: the entire rule set, or some

rules of interest while ignoring others. As shown in Figure 3.2, L_{whole} can be redefined as:

$$L_{whole} = L^{I}_{whole} \cup L^{NI}_{whole} \tag{3.2}$$

where L^{I}_{whole} is bounded by the number of interaction rules that are of interest to the designer for the particular study, and L^{NI}_{whole} represents the set of all possible system states that are not of interest. The system designer's interest in turn depends on the system to be studied, knowledge of the system, and the objective of modeling. Instead of dealing with the whole set of states of real system to be studied, L^{I}_{whole} considers only a subset of it regarding the system designer's interest. Ideally, if we knew all rules defining a system, we could completely describe, understand, and explain it. However, in practice, this is usually not the case. In fact, a system is typically analyzed via its model, which is an abstract approximation of the real system. This model is relative to the system designer's interest, and other things such as computational power and simulation time constraint.

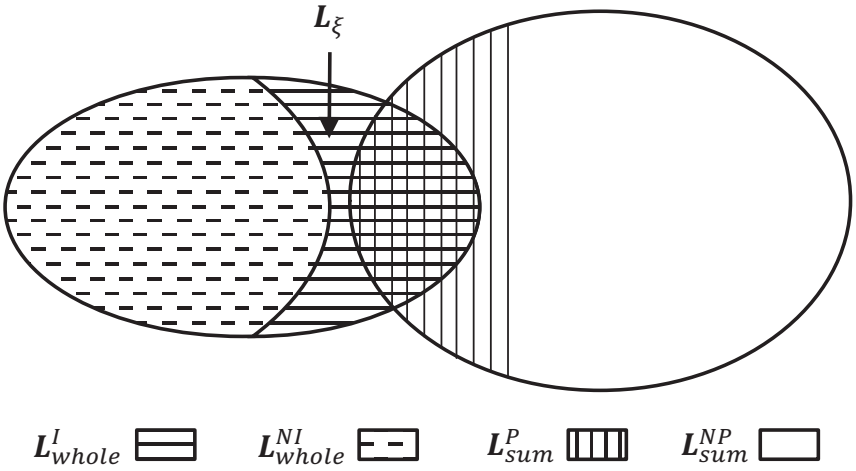

Figure 3.2: Set of Emergent Property States

The size of L_{sum} increases exponentially with the number of agents as all possible combinations of agent state are considered. Although it is mathematically possible to derive L_{sum}, not all of these combinations can happen in practice. Some of them will never

occur because of constraints among agents. An example is combinations in which two agents are located at the same position. Therefore, L_{sum} can be split into two disjoint sets as follows:

$$L_{sum} = L^{P}_{sum} \cup L^{NP}_{sum} \tag{3.3}$$

where L^{P}_{sum} is a subset of feasible combinations of agents' behavior that can happen in practice, and L^{NP}_{sum} is the set of combinations of agents' behavior that will not exist in practice.

Given the above, the set of emergent property states is reduced to:

$$L_{\xi} = L^{I}_{whole} \setminus L^{P}_{sum} \tag{3.4}$$

Emergent property states, with respect to a model of interest, are due to non-trivial interactions of agents, and cannot be derived by summing individual agents' states. As shown in Figure 3.2, there are states in L^{I}_{whole} that can be found in L^{P}_{sum}, in other words,

$$L^{I}_{whole} \cap L^{P}_{sum} \neq \emptyset \tag{3.5}$$

Intuitively, these states are resultant from agent computations that do not require interactions, or from the interactions of agents that have no effect on agent behavior. For example, in the flock of birds model detailed in Section 3.4, two individuals can be very far apart and as such interaction rules between them have no effect.

Contrary to Kubik's approach, which regards emergent properties as system states, we do not consider that L_{ξ} contains the emergent properties but only states that together, following particular criteria, can form an emergent property. In the next step, we propose to determine whether L_{ξ} contains emergent properties that have been seen before, or that are beneficial or harmful to the system. Towards this, we propose to use emergence criteria derived from the system expert, as well as information obtained from L^{NI}_{whole}.

3.2 Grammar-based System Formalization

As discussed above, Kubik's work has several limitations: (1) suffers from state-space explosion, (2) cannot model agent types, (3) does not support mobile agents, and (4) only deals with closed systems with fixed number of agents. L_{sum} is a mechanical combination of individual states of agents, regardless of constraints among agents, thus resulting in invalid permutations of agents' states that will never happen in practice. A large number of invalid, unreachable system states will lead to the state-space explosion problem, even with a small number of agents as shown in Section 3.4. Moreover, the example used in Kubik's paper, The Game of Life, is simple in the sense that all agents have the same type and are stationary.

The first limitation is addressed by reducing L_{whole} and L_{sum} to L^I_{whole} and L^P_{sum} respectively as presented in the previous section. Further reduction of state space is discussed in Section 3.5. In this section, we propose a more general grammar-based formalization of the system to address the three remaining limitations. Accordingly, we enhance the grammar-based approach proposed by Kubik with three main extensions: (1) introduce agent type (A_{ij} denotes an agent of type i and instance j), (2) introduce mobile agents by defining mobility as attributes of agents $P_i = P_{i\_mobile} \cup P_{i\_others}$ where P_i denotes the set of attributes of agents of type i, and (3) model open systems whereby agents can enter and leave system over time. These kinds of systems are ubiquitous in practice such as traffic networks [55], social networks [40], and flock of birds [74], among others. Table 3.1 shows the list of notation used in the formalization.

A multi-agent system consisting of an environment and a set of agents is formalized as an extended cooperating array grammar system where context-free grammars represent agents, and a two-dimensional array of symbols represents the environment. Each grammar

	Notation	Description
System	$S(t)$	state of system at time t
Environment	V_E	set of possible cell states
	V_e	set of possible states of cell e
	$S_E(t)$	state of environment at time t
	$s_e(t)$	state of cell e at time t
Agent Type	m	number of agent types
	n_i	number of agents of type i $(1 \leq i \leq m)$
	V_{A_i}	set of possible states for agents of type i
Agent	n	number of agents
	A_{ij}	agent of type i $(1 \leq i \leq m)$ and instance j $(1 \leq j \leq n_i)$
	V_A	set of possible agent states for all agent types
	P_i	set of attributes for agents of type i
	R_i	set of behavior rules for agents of type i
	$s_{ij}(t)$	state of agent A_{ij} at time t
	$L_(A_{ij})$	set of system states representing the behavior of agent A_{ij}
Emergence	$\oplus$	superimpose operator
	L^I_{whole}	set of system states of interest due to agent interactions
	L^P_{sum}	set of possible system states by summing agents' states
	L_ξ	set of emergent property states

Table 3.1: Glossary of Notations

has its own rewriting rules defining how the grammar cooperates with the other grammars and with the array, i.e. rewrite the symbols on the array. A system of m agent types and a total of n agents $A_{11}, \ldots, A_{1n_1}, \ldots, A_{mn_m}$ interacting in a 2D grid environment (E) of c cells is defined as follows:

$$GBS = (V_A, V_E, A_{11}, \ldots, A_{1n_1}, \ldots, A_{mn_m}, S(0)) \tag{3.6}$$

where V_A denotes the set of possible agent states for all agent types, V_E denotes the set of possible cell states, A_{ij} denotes an agent of type i $(1 \leq i \leq m)$ and instance j $(1 \leq j \leq n_i)$, and $S(0)$ denotes the initial system state. The system state at time t $(S(t))$, is composed of state of the environment $(S_E(t))$ and states of agents $(s_{ij}(t))$ at time t. Hence,

$$S(t) = S_E(t) \bigcup_{\forall i \forall j} s_{ij}(t) \tag{3.7}$$

3.2.1 Environment

The environment is part of the system that lies outside the agents and can be regarded as a platform for agent interactions. The environment is shared by all agents, and plays a *passive* role in the sense that its state is changed by interactions with agents. For example, ants communicate with each other to find the shortest path for food by leaving pheromone trails for other ants to follow. In this situation, the environment consists of the ground and the trails indicating shortest paths between the nest and sources of food. The interactions of ants change the environment, i.e. the trails are changed over time.

The environment can also play an *active* role in a multi-agent system. The environment can have its own properties and rules and it can provide feedbacks to agents. Changes of the environment manifest or visualize the connections between agents, and impact the behavior of agents. For example, a road traffic network consists of agents, including pedestrians and vehicles, and the environment composed of roads and traffic lights. The traffic lights have their own rules which specify how pedestrians and vehicles move on the roads. An environment playing an active role in a system could be regarded as an autonomous agent. For simplicity, we only consider passive environments and assume that environments have no behavior rules and change only as a result of agent actions.

In the scope of this thesis, the environment E is modeled as a 2D grid[1] that is subdivided into c units called cells (e). Changes of the environment are therefore changes of the states of the cells. For example, a cell turns from "occupied" to "free" when the agent that occupies the cell moves to another cell. V_e denotes the set of possible states of cell e. Similarly, V_E denotes the set of possible cell states. For the environment (E),

[1] E can be easily extended to model other topologies.

$$V_E = \bigcup_{e=1}^{c} V_e \tag{3.8}$$

In addition, the environment state is made up of the states of all cells in the environment. The states of cell e and the environment E at time t are $s_e(t) \in V_e$ and $S_E(t) \in V_E^c$ respectively.

3.2.2 Agents

In a multi-agent system, agents are autonomous entities. Agents are characterized by a set of attributes, e.g. the location of an agent in a spatial environment or the distance an agent travels in a time step. Values of these attributes at a point of time present the state of an agent at that time. The state of an agent is changed because of the behavior of the agent. Agents act and interact with other agents and the environment according to their own rule set. Generally, a behavior rule is a function from agent state to agent state as follows:

$$rule(condition) : s(t) \rightarrow s(t+1) \tag{3.9}$$

If the condition is fulfilled at time t, the agent will apply the rule to transform its current state at time t ($s(t)$) to state at the successive time step ($s(t+1)$). The condition of rules includes the states of the neighbors of the agent and the state of the agent itself. Neighbors of an agent are usually close to the agent in terms of proximity. Defining neighborhood, i.e. proximity, is problem-specific. Different models of the same problem may have different ways of specifying neighborhood. For example, cellular automata have the two most common types of neighborhoods: *von Neumann* neighborhood and the *Moore* neighborhood. The former consists of the four orthogonally adjacent cells. The latter includes eight neighbors inhabiting the cells that are horizontally, vertically, and diagonally

adjacent to the cell whose state is to be calculated.

Looking at agents from the point of view of aggregative statistics, there are two main factors related to the presence of emergence: population size of agents and number of types of agents. On the one hand, the population size of agents may be a requirement for emergence. Emergent phenomena sometimes need a specific minimal number of agents to happen. For example, a glider in The Game of Life requires five agents to form. Furthermore, a large population of agent results in more interactions taking place between agents, thus increasing the chance of emergence. On the other hand, high heterogeneity of agents can also increase the chance of emergence. This is because different types of agents with different attributes and behavior rules would increase the complexity of agent interaction.

n agents are classified into m different types in which all agents of the same type have the same attribute set and behavior rule set. A_{ij} denotes an agent of type i ($1 \leq i \leq m$) and instance j ($1 \leq j \leq n_i$), where n_i is the number of agent instances of type i, and $n_1 + n_2 + ... + n_m = n$. An agent of type i is characterized by three factors: a set of attributes for agents of type i (P_i), a set of behavior rules for agents of type i (R_i), and an initial state ($s_{ij}(0)$).

P_i consists of two main subsets: P_{i_mobile} modeling the mobility of agents of type i and P_{i_others} modeling other attributes. Velocity and location are examples of mobility attributes. We are interested in mobility because it is typically an important characteristic of systems exhibiting emergence. Particularly, mobility is the prerequisite of the formation of emergent patterns. In many systems, emergence often appears in form of patterns or structures such as flocks of birds, schools of fish, colonies of ants, gliders and spaceships in The Game of Life, and lanes of pedestrians crossing a road. It is important to distinguish agent attributes from their values. An attribute is a characteristic that describes an agent, while an attribute value is a value the attribute can take. An attribute can take a number

of values, i.e. value set. For instance, attribute color has several values, such as red, yellow, and green.

Agent behavior (L) is characterized by behavior rules (R) that define how agents act and interact with other agents and the environment. For simplicity, we assume that no evolutionary processes are involved in the system, i.e. behavior rules do not change over time. R_i denotes the set of behavior rules of agents of type i. Similar to P_i, which consists of P_{i_mobile} and P_{i_others}, R_i consists of R_{i_mobile} that impacts agent mobility, i.e. changes values of attributes of P_{i_mobile}, and R_{i_others} for the rest.

An agent changes its state because it is triggered by a rule that affects the agent itself or one of its neighbors changes state. The state of A_{ij} at time t, denoted by $s_{ij}(t)$, is defined by values of its attributes at time t. The initial state of A_{ij} ($s_{ij}(0)$) is specified by the values of all of its attributes at the beginning. Only the initial state $s_{ij}(0)$ is specific to the agent, while P_i and R_i are common to all agents of type i.

V_{A_i} denotes the set of possible agent states for agents of type i. For the agents (A),

$$V_A = \bigcup_{i=1}^{m} V_{A_i} \tag{3.10}$$

where V_{A_i} denotes the set of possible states for agents of type i.

Agent of type i ($1 \leq i \leq m$) and instance j ($1 \leq j \leq n_i$), A_{ij}, is defined as follows:

$$A_{ij} = (P_i, R_i, s_{ij}(0)) \tag{3.11}$$

where P_i denotes the set of attributes for agents of type i, R_i denotes the set of behavior rules for agents of type i, and $s_{ij}(0)$ denotes the initial state of the agent. P_i is defined as:

$P_i = P_{i\_mobile} \bigcup P_{i\_others}$ where

$P_{i\_mobile} = \{x \mid x \textit{ is an attribute that models mobility}\}$

$P_{i\_others} = P_i \setminus P_{i\_mobile}$

Agents change their states according to behavior rules that are regarded as functions from a set of agent states to the set itself. Some of these rules that affect the mobility of agents, for example change location or speed, are defined as mobile rules.

$R_i : V_{A_i} \rightarrow V_{A_i}$

$R_i = R_{i\_mobile} \bigcup R_{i\_others}$

A_{ij} has an initial state $s_{ij}(0) \in V_{A_i}$. Systems displaying emergent properties are usually sensitive to initial configurations. Slight changes in initial conditions may result in no emergence any more [44]. In other words, emergence is considered with respect to some specific initial system states. For example, in The Game of Life, there are some system configurations that never change over time, i.e. the system configuration is always identical to its initial status, thus having no emergent patterns. Therefore, starting conditions should at least enable state transitions that bring about changes in the state of the system.

3.3 Emergent Property States

Our approach computes two sets of system states corresponding to the two levels of abstraction defined above, namely, the macro level when regarding the system as a whole and the micro level when regarding the system as an aggregation of its individual agents. The difference between the two sets is a set of emergent property states that are due to interactions of agents, but cannot be deduced from individual agents. As discusses above, the set of emergent system states is defined as:

$$L_\xi = L^I_{whole} \setminus L^P_{sum}$$

L^P_{sum} can be determined by adding agent constraints among others but is not straightforward. We leave this as future work and for simplicity we use L_{sum} for the rest of this

section. In addition, in the next section, we show that we no longer need L^P_{sum} and L_{sum}, and calculate L_ξ using only L^I_{whole}.

Taking the interactions of agents (denoted as $GROUP$) into account, the system behavior of interest (L^I_{whole}) returns a set of words (w) that represents the set of system states reachable from the initial system state. Complex adaptive and non-linear systems with strange attractors where system states vary continuously are not considered in this study. L^I_{whole} with respect to the initial state is therefore the set of all distinct states obtained as follows:

$$L^I_{whole} = \{w \in V^{c+n} \mid S(0) \Rightarrow^*_{GROUP} w\} \tag{3.12}$$

Algorithm 1 presents the pseudo-code for the calculation of L^I_{whole}. Given an initial state $S(0)$ (line 3), the system is simulated until it reaches a state that has already appeared before (line 7). We stop the simulation when the system state repeats itself. L^I_{whole} comprises all distinct states $S(t)$ obtained (line 18).

The sum of agents' behaviors (L_{sum}) is defined as the set of words resulting from superimposing behaviors of individual agents.

$$L_{sum} = \oplus(L(A_{11}), \ldots, L(A_{1n_1}), \ldots, L(A_{mn_m})) \tag{3.13}$$

where $L(A_{ij})$ denotes the behavior of agent A_{ij} and $\oplus$ is the superimpose operator defined in [52]. The superimpose operator $\oplus$ does a sum of the individual agents' behaviors when they do not interact with each other. Let $w_1 = a_1a_2 \ldots a_x, w_2 = b_1b_2 \ldots b_y$ be words of symbols $a_i, b_j, 1 \leq i \leq x, 1 \leq j \leq y$ over an alphabet $(V_A \cup V_E)^+$, and ϵ denotes the empty symbol. Hence, the superimposition of the word w_1 on the word w_2 is a function $\oplus : V^* \times V^* \rightarrow V^*$ that results in $w_{supimp} = c_1c_2 \ldots c_z$ of symbols $c_k, 1 \leq k \leq z$, defined as follows:

1. $z = max(x, y)$;

Algorithm 1 Pseudo-code for L^I_{whole} Calculation

```
1: procedure calculate_L^I_whole
2:     t:= 0;                                   //initialized clock
3:     set S(0);                                //set initial system state
4:     L^I_whole:= ∅;
5:     add S(0) to L^I_whole;
6:     repeat:= false;
7:     while repeat false do
8:         t:= t + 1;
9:         ...                                  //simulate next step
10:                                             //use a for loop to compare states
11:        for i = 0 to t − 1 do
12:            if S(t) equal S(i) then
13:                repeat = true;
14:                exit for loop;
15:            end if
16:        end for
17:        if repeat false then
18:            add S(t) to L^I_whole ;
19:        end if
20:    end while
21:    return L^I_whole;
22: end procedure
```

2. if $a_i \in V_A$ then $c_k = a_i$;

3. if $a_i = \epsilon$ then $c_k = b_j$;

4. if $b_j = \epsilon$ then $c_k = a_i$;

5. if $a_i \in V_E$ and $b_i \in V_E$ then $c_k = a_i$;

6. if $a_i \in V_E$ and $b_j \in V_A$ then $c_k = b_j$.

The superimposition is done over all permutations of n behaviors of agents. It is important to note that ordering is important in the process of calculating the superimposition. The

expression below shows the superimposition of three languages L_1, L_2, and L_3.

$$\begin{aligned}\oplus(L_1, L_2, L_3) = L_1 \oplus (L_2 \oplus (L_3)) \cup L_1 \oplus (L_3 \oplus (L_2)) \\ \cup L_2 \oplus (L_1 \oplus (L_3)) \cup L_2 \oplus (L_3 \oplus (L_1)) \\ \cup L_3 \oplus (L_1 \oplus (L_2)) \cup L_3 \oplus (L_2 \oplus (L_1))\end{aligned}$$

Defining the sum of the individual agents' behaviors is difficult. Ideally, the result should contain exactly all designed system states that can be derived from the system specification. Unfortunately, this is only true if agents are independent from the others in the system. Given the initial system state, we obtain n states where each consists of one agent. Considering only the agent in the system, the agent's behavior returns a set of words (w) that represents the set of system states reachable from the corresponding system state. $L(A_{ij})$ is defined as follows:

$$L(A_{ij}) = \{w \in V^{c+1} \mid (S_E(0) \cup s_{ij}(0)) \Rightarrow^* w\} \tag{3.14}$$

Agent symbols have priority over environmental symbols. Any non-empty symbol has priority over the empty symbol ϵ. For example, consider $V_A = \{a_1, a_2, a_3\}, V_E = \{o, f\}$, $L_1 = \{fa_1of\}, L_2 = \{oa_2ff\}, L_3 = \{foffa_3\}$. The superimposition of the agents' behaviors will be the language $L_{sum} = \{fa_1ofa_3, oa_2ffa_3, fa_1ffa_3, fa_2ffa_3\}$.

In summary, the superimpose operator is a mechanical solution to sum agents' states. However, one of the major disadvantages of the superimpose operator is that it suffers from state-space explosion problem. This is because superimposition takes into account all permutations of agents' states, including those that are invalid, i.e. conflict with constraints among agents. As shown in Section 3.4, L_{sum} grows to be very large even with a small number of agents. Section 3.5 presents an alternative to deducing L_ξ without involving L_{sum}. The key idea is to measure degree of agent interaction in order to derive L_ξ from L^I_{whole}.

To validate and evaluate the scalability of the proposed approach, we consider its application to two examples: Boids model that simulates the flocking motion of a group of birds in the next section, and deadlock in multi-threaded programs in the next chapter. In the former, we illustrate how to formalize the interactions of birds using our grammar-based formalization. Given the formalism of a group of birds, a set of emergent property states is derived. This set consists of states that possess flocking patterns with respect to a certain criteria of flocking and the others. The proposed approach is validated up to 1,024 birds in a grid of 128 x 6,128 with the execution time less than five hours. This finding supports scalability of the approach. The latter, deadlock emergence in multi-threaded programs, considers deadlock states due to the interleaving among threads in a concurrent program. Threads are formalized as grammar-based agents, and emergent property states, including deadlock ones, are then derived. The proposed approach is validated to be applicable to precisely detect all potential deadlock states when varying the number of threads and thread types.

3.4 Example: Bird Flocking Emergence

In this section, we apply the proposed formalism to formalize birds' interactions in terms of grammar-based agents. The input to our approach is the Boids model and the output is a set of emergent property states. The derived set of emergent property states allows users to identify flocking patterns in accordance to their pre-defined criteria of flocking behavior. In addition, the set has some states that might exhibit other surprising properties that go beyond the flocking criteria. The experimental results (discussed in Section 3.4.4) show that our approach is valid to detect flocking up to 1,024 birds in a 128 x 128 grid.

3.4.1 The Boids Model

The Boids model [74] captures the motion of bird flocking and is a seminal example for studying emergence [19]. At the macro level, a group of birds tends to move in a V-like formation, which has aerodynamic advantages, obstacle avoidance, and predator protection, regardless of the initial positions of the birds. At the micro level, three simple rules define how each bird flies: (1) separation - steer to avoid crowding neighbors, (2) alignment - steer towards the average heading of neighbors, and (3) cohesion - steer towards the average position of neighbors.

To demonstrate our proposed approach, we extended the Boids model to include two types of birds, ducks and geese. For ease of discussion, we model a multi-agent system with ten birds with equal numbers of ducks and geese interacting in an environment represented as a 2D grid of 8×8 cells. Each cell is *occupied* by a bird or *free*, and two birds cannot be located at the same cell at the same time. Birds have two attributes position and velocity which model birds' mobility. The *position* of a bird is location of the cell occupied by that bird. The *velocity* is a vector specifying moving direction and speed. Ducks can fly zero, one, or two cells per time step in one of eight directions: north, north-east, east, south-east, south, south-west, west, and north-west. Similarly, geese can fly at the maximum speed of three cells per time step. The vector representation for velocity of ducks is shown in Table 3.2. Birds behave according to three rules: (1) separation: avoid collision with nearby birds, (2) alignment: fly as fast as nearby birds of the same type, and (3) cohesion: stay close to nearby birds of the same type.

Direction	Speed		
	0	1	2
North	(0,0)	(0,1)	(0,2)
North-East	(0,0)	(1,1)	(1,2), (2,1), (2,2)
East	(0,0)	(1,0)	(2,0)
South-East	(0,0)	(1,-1)	(1,-2) (2,-1), (2,-2)
South	(0,0)	(0,-1)	(0,-2)
South-West	(0,0)	(-1,-1)	(-1,-2), (-2,-1), (-2,-2)
West	(0,0)	(-1,0)	(-2,0)
North-West	(0,0)	(-1,1)	(-1,2), (-2,1), (-2,2)

Table 3.2: Vector Representation for Velocity of Ducks

3.4.2 System Formalism

The Boids model is formalized as:

$$GBS_{boid} = (V_A, V_E, A_{11}, \ldots, A_{15}, A_{21}, \ldots, A_{25}, S(0))$$

where A_{1j} denotes a duck instance j $(1 \leq j \leq 5)$, and A_{2j} denotes a goose instance j $(1 \leq j \leq 5)$, $V_A = V_{A_1} \cup V_{A_2}$ denotes the set of possible states for the ducks (V_{A_1}) and the geese (V_{A_2}), V_E denotes the set of possible cell states. $S(t) \in (V_A \cup V_E)^+$ denotes system state at time t, and $S(0)$ denotes the initial system state at time zero.

For cell e, $V_e = \{o, f\}$ where o means occupied and f means free, and $s_e(t) \in V_e$. For the entire environment E, $V_E = \bigcup_{e=1}^{64} V_e = \{o, f\}$, and $S_E(t) \in V_E^{64}$, where $64 = 8 \times 8$ is the number of cells.

A duck instance $A_{1j}(1 \leq j \leq 5)$ is defined as follows:

$$A_{1j} = (P_1, R_1, s_{1j}(0))$$

where

$P_1 = P_{1\_mobile} \cup P_{1\_others}$

$P_{1\_mobile} = \{position(g_{1j}), velocity(v_{1j})\}$

$P_{1\_others} = \emptyset$

$V_{A_1} = \{(x, y)|1 \leq x \leq 8, 1 \leq y \leq 8\} \times \{(\alpha, \beta)| - 2 \leq \alpha \leq 2, -2 \leq \beta \leq 2\}$

$R_1 = R_{1\_mobile} \cup R_{1\_others}, R_{1\_others} = \emptyset$

$s_{1j}(t) \in V_{A_1}$

R_{1_mobile} defines the update of the position $g_{1j}(t)$ and the velocity $v_{1j}(t)$ of duck A_{1j} over time. We limit the speed of ducks to two cells per time step so that they cannot fly arbitrarily fast. Consequently, absolute values of the horizontal component (α) and the vertical component (β) of the velocity vector are bounded to two cells. Let $sign(\alpha)$ and $sign(\beta)$ return signs of α and β, i.e. 1 for positive and -1 for negative, respectively. Both position and velocity of birds are represented as 2D vectors; the update is therefore simply vector additions.

$(\alpha, \beta) = v_{1j}(t) + separation(A_{1j}) + align(A_{1j}) + cohesion(A_{1j})$

$v_{1j}(t + 1) = (sign(\alpha)min(|\alpha|, 2), sign(\beta)min(|\beta|, 2))$

$g_{1j}(t + 1) = g_{1j}(t) + v_{1j}(t + 1)$

Separation: If duck a is close to another duck or goose b, i.e. within ϵ cells, then a flies away from b.

$$separation(a) = \sum_{distance(a,b) \leq \epsilon} a.position - b.position$$

Algorithm 2 presents the pseudo-code for the calculation of the separation vector of a duck. The separation vector is the sum of the differences between the position of duck a and the position of other close boids (within ϵ cells), regardless of agent type.

Alignment: Duck a changes its velocity by $\lambda\%$ towards the average velocity of its neighboring ducks.

$$align(a) = (\frac{\sum_{duck(b)}^{k} b.velocity}{\frac{neighbor(a,b)}{k}} - a.velocity) \times \frac{1}{\lambda}$$

Algorithm 2 Pseudo-code for Separation Rule

```
1: procedure separation(duck a)
2:     vector c := 0;
3:     for each boid b do
4:         if |a.position - b.position| ≤ ϵ then
5:             c := c - (b.position - a.position);
6:         end if
7:     end for
8:     return c;
9: end procedure
```

Algorithm 3 presents the pseudo-code for the calculation of the alignment vector of a duck.

Algorithm 3 Pseudo-code for Alignment Rule

```
1: procedure alignment(duck a)
2:     vector c := 0;
3:     integer k :=0;
4:     for each neighbor duck b do
5:         k := k + 1;
6:         c := c + b.velocity;
7:     end for
8:     c := c / k;
9:     return (c - a.velocity) / λ;
10: end procedure
```

In this pseudo-code, we only consider ducks and do not care about geese (line 4) since birds are supposed to fly in groups of birds of the same type. In other words, ducks tend fly together with ducks and geese tend fly together with geese. Duck a gradually aligns its velocity towards the average velocity of its neighboring ducks by $\lambda\%$ (line 9).

Cohesion: Duck a moves by $\gamma\%$ towards the center of its neighboring ducks.

$$cohesion(a) = (\frac{\sum_{\substack{duck(b) \\ neighbor(a,b)}}^{k} b.position}{k} - a.position) \times \frac{1}{\gamma}$$

Algorithm 4 presents the pseudo-code for the calculation of the cohesion vector of a duck. Similar to alignment, duck a gradually coheres towards the center of its neighboring ducks

Algorithm 4 Pseudo-code for Cohesion Rule

```
1: procedure cohesion(duck a)
2:     vector c := 0;
3:     integer k :=0;
4:     for each neighbor duck b do
5:         k := k + 1;
6:         c := c + b.position;
7:     end for
8:     c := c / k;
9:     return (c - a.position) / γ;
10: end procedure
```

by $\gamma\%$ (line 9). The model for geese follows in a similar manner except that their maximum speed is three cells per time step.

3.4.3 Simulation for Calculating Flocking Emergence States

Our proposed approach returns a set of emergent property states. To verify that these states contain emergent properties, we show how the well-known flocking of birds emergence is derived from L_ξ. The initial system state is given as the state at time $t = 0$ in Figure 3.3. For ease of visualization, we distinguish ducks from geese using a star ($*$) symbol and bolded cell. $< j, (\alpha, \beta) >$ denotes a bird instance j, with velocity (α, β).

Figure 3.3 shows a simulation of the system when $\epsilon = 2$, $\lambda = 10$, and $\beta = 8$. We observe that the birds keep flying in the same pattern from $t = 4$. Moreover, the system gets back to the system state $S(4)$ at time $t = 12$: $S(12) = S(4)$. Hence, $L^I_{whole} = \{S(0), S(1), \ldots, S(11)\}$.

L_{sum} is calculated using the superimpose operator as $L_{sum} = \oplus(L(A_{11}), \ldots, L(A_{25}))$. By definition and following our discussion above, L_{sum} tends to be very large, even for small problem sizes. As such, we consider for illustration two geese A_{23} and A_{25}. For simplicity, the superimposition of two birds, $L(A_{23})$ and $L(A_{25})$, is shown in Figure 3.4.

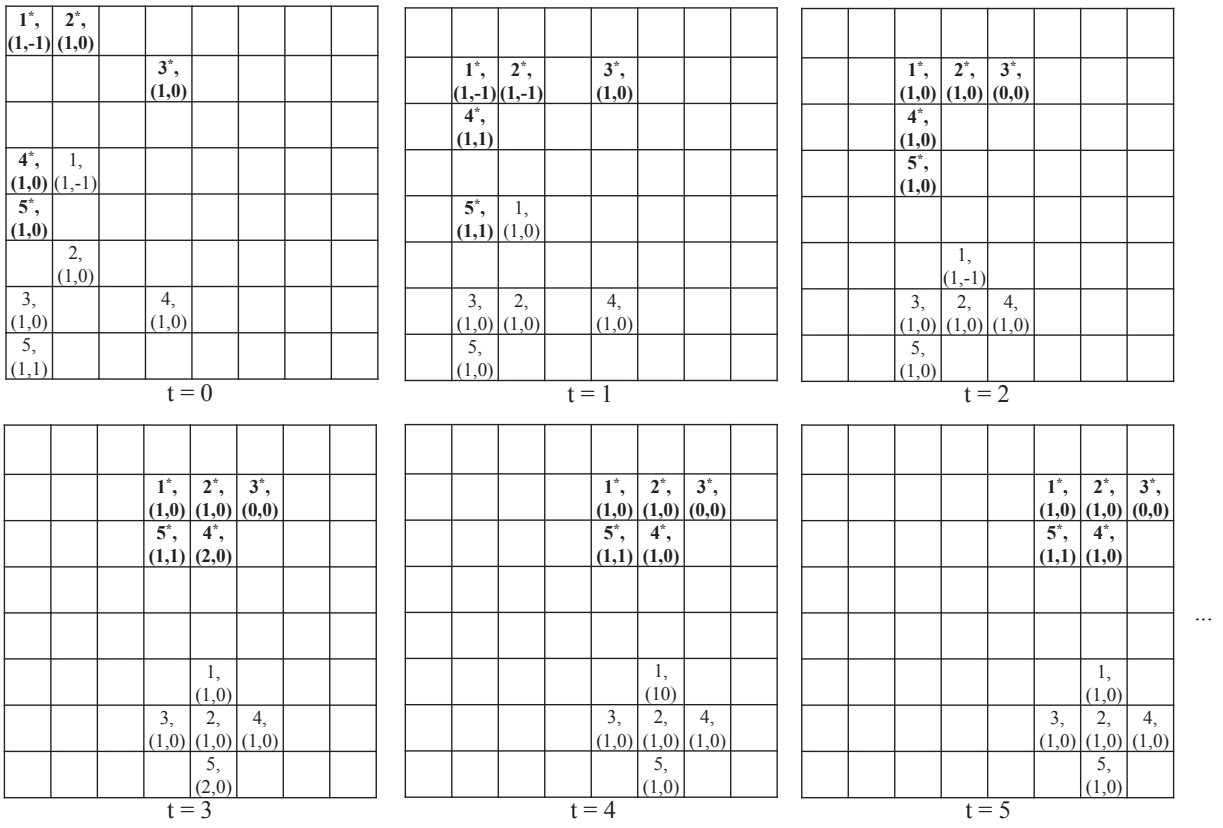

Figure 3.3: Snapshot of Emergent Property States

Formally,

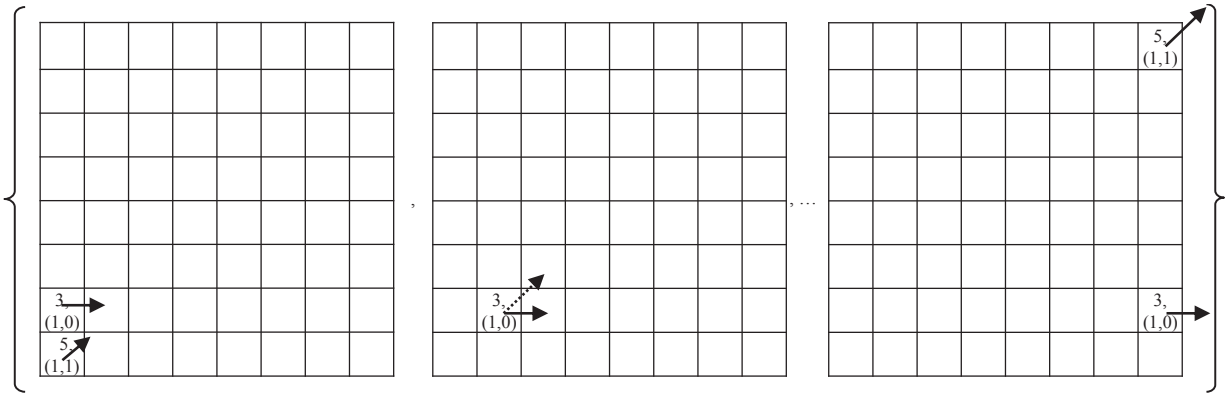

Figure 3.4: Example of $L(A_{23}) \oplus (L(A_{25}))$

$$
\begin{aligned}
&\oplus (L(A_{23}), L(A_{25})) \\
&= L(A_{23}) \oplus (L(A_{25})) \cup L(A_{25}) \oplus (L(A_{23})) \\
&= \{((2,(1,1),(1,1))fff\ldots(2,(1,2),(1,0))fff\ldots), \\
&\quad (fff\ldots(2,(2,2),(1,0))fff\ldots), \\
&\quad (fff\ldots(2,(8,2),(1,0))fff\ldots(2,(8,8),(1,1))\}
\end{aligned}
$$

$L_(A_{23})$ and $L_(A_{25})$ are behaviors of agent A_{23} and A_{25} respectively. f represents an empty cell and a tuple $< i,(x,y),(\alpha,\beta) >$ represents the state of a bird of type i at cell (x,y) with velocity (α,β). For example, $< 2,(1,2),(1,0) >$ represents a goose locating at cell $(1,2)$ with velocity $(1,0)$. The important point to note is that the x-axis is horizontal and oriented from left to right, and the y-axis is vertical and oriented from bottom to top. In the example of ten birds, L_{sum} contains one common state, S(0), with L^I_{whole}. Therefore, $L_\xi = \{S(1), S(2), \ldots, S(11)\}$.

So far, we have successfully determined the set of emergent property states, L_ξ, which consists of system states that are due to the interactions of birds, but cannot be derived from the behavior of individual birds. However, we can go further by making these states meaningful with respect to users' perspective. In particular, the set of emergent property states allow the users to identify emergent properties possessed in these states. These properties are what the users are interested in and seek in relation to their knowledge of emergent properties. More interestingly, users probably recognize additional surprising properties that they do not intentionally look for. For example, if the users consider that a group of birds form a flock if at least four birds of the same type fly *together*, with each bird having at least one immediate neighbor of the same type, then ten emergent property states from $S(2)$ to $S(11)$ have flocking emergent property. In addition, flocking patterns in S(3), ..., S(10), and S(11) are identical, regardless of different positions of the birds

forming the flock. In other words, we have two types of flocking emergence that the users have already known: one type in S(2) and one type in S(3), ..., S(10), and S(11). One interesting observation is that the second flocking type is more popular and stable than the first one. This makes sense because the second flocking type visually seems to be superior to the first in terms of aerodynamic efficiency, obstacle avoidance ability, and predator protection power as shown in Figure 3.3.

The remaining emergent property state, $S(1)$, is unknown with respect to the criteria of flocking described above. In fact, it does not have any group of four birds of the same type flying together. However, we observe that there are two groups of three birds, and the birds already tend to fly at the same speed (of one cell per time step). Furthermore, there are two additional birds of the same type near (distance of one cell) to each group. Therefore, each group is close to a form of flocking since at the next step, the two birds likely join their corresponding group of three to form a flock of five birds. As a result, the unknown emergent property state, $S(1)$, provides some clues that help predict the flocking behavior of the group of birds as well as the process leading to flock formation. In other words, unknown emergent property states give users the benefit of understanding and predicting the system behavior as well as identifying emergent properties, although they do not explicitly exhibit emergent properties regarding the users' knowledge.

On the other hand, if a flock is defined as a group of at least three birds of the same type flying together, $S(1)$ becomes a known emergent property state, hence all emergent property states are known. Consequently, there are three types of known flocking behavior: in $S(1)$, in $S(2)$, and in S(3), ..., S(10), and S(11). We see that different criteria of describing flocking result in different views of emergence. What users perceive may slightly vary, depending on their knowledge of emergence. However, whatever the case, the set of emergent property states is the same, i.e. independent from the users. This set plays

the key role in allowing the users to investigate to determine emergent properties of their interest.

We also validate the proposed approach for a specific case of five ducks and give geese in a grid of 16 x 16. According to the experimental results, $L^I_{whole} = \{S(0), S(1), \ldots, S(46)\}$ and $L_\xi = \{S(1), S(2), \ldots, S(46)\}$. If we define a flock as a group of at least three birds of the same type flying together, all states of L_ξ excepts S(9), S(10), and S(11) possess flocking patterns. These exceptions are unknown emergent property states. In these states, there is no flocking because birds have to separate, i.e. break their flock formed previously, to avoid collision with others. However, the birds eventually gather and form flock again as soon as there is no longer collision. In contrast to the example of 8 x 8 grid (see Figure 3.3) in which the unknown emergent property state, S(1), allows to predict the future flying behavior of the birds, the unknown emergent property states in this example of 16 x 16 grid provide a better understanding of how birds form a flock, avoid collision, and then form a flock again. Another interesting finding is that S(22) starts to have two flocks of three birds instead of one flock of three birds as the previous states. In other words, after a number of steps, another type of birds finally builds up a flock.

On the other hand, if a flock is regarded as a group of at least 4 birds of the same type flying together, starting from S(23), emergent property states exhibit flocking behavior. The other states of L_ξ are unknown emergent property states. Similar to the case of 8 x 8 grid, S(22) enables the prediction of the formation of flocking. Other states, S(1), S(2), ..., S(20) probably contain something interesting for further investigation. Finally, if we define a flock as a group of five birds, the set of known emergent property states would be $\{S(30), S(31), \ldots, S(46)\}$. It is clear to observe the gradual development of flocking from three birds, four birds, to five birds. The formation of a flock of five birds is a result of a chain of improvements of the casual relationships among birds rather than a sudden

occurrence.

3.4.4 Evaluation

In this section, we present a theoretical and experimental analysis of our approach. This analysis provides a feel of the scalability of the grammar-based emergence formalization. In particular, we show that the proposed approach scales well with the number of birds and the environment size; and the execution time is less than five hours for 1,024 birds in a 128 x 128 grid.

Since we derived the sets of states for L^I_{whole}, L_{sum}, and L_ξ, we measure the complexity in terms of the number of states. We implemented our approach using a Java program, and our experimental analysis quantifies the state size, by varying the number of birds.

Theoretical Analysis

The complexity of deriving L_ξ (O(L_ξ)) consists of two parts: complexity of L^I_{whole} (O(L^I_{whole})) and complexity of L_{sum} (O(L_{sum})).

$$O(L_\xi) = O(L^I_{whole}) + O(L_{sum}) \tag{3.15}$$

Because detecting emergence is to differentiate system states that appear when taking into account interactions of agents, but not when regarding them separately, it is reasonable to use the number of system states as a complexity measure. Key complexity factors include: the environment size (2D grid of size x by y), the number of agent types (m), the number of agents (n), and the number of possible states an agent can take (s). We derive $O(L^I_{whole})$ and $O(L_{sum})$ in the *worst case*. Let $n = n' + n''$ where n' is the number of mobile agents, and n'' is the number of stationary agents.

$\boldsymbol{O(L^I_{whole})}$: Given a position, an agent can take one of s states. Moreover, stationary agents are fixed in n'' positions. There are $\binom{xy-n''}{n'}$ possibilities for allocating n' mobile

agents into the remaining $xy - n''$ positions. Hence,

$$O(L^I_{whole}) = O(\binom{xy - n''}{n'} s^n) \tag{3.16}$$

$\boldsymbol{O(L_{sum})}$: Without considering the interactions of agents, a mobile agent, in the worst case, can arbitrarily move to any position (cell) in the environment. Hence, the upper bound complexity of superimposing individual behaviors for all agents is:

$$O(L_{sum}) = O((xy)^{n'} s^n) \tag{3.17}$$

For example in The Game of Life, agents are stationary, i.e. $n' = 0$ and $n'' = n$, and $O(L^I_{whole}) = O(L_{sum}) = O(s^n)$). If all agents are mobile, i.e. $n' = n$ and $n'' = 0$, then $O(L_{sum}) = O((xy)^n s^n)$, which is much larger than $O(L^I_{whole}) = O(\binom{xy}{n} s^n)$. This is because the summing operation involves all combinations of individual agents' behaviors, including those that could never happen in practice, as discussed before.

To reduce $O(L_{sum})$, we consider the fact that some results from the superimposition cannot be possible due to system wide rules, so we eliminate these from the calculation. For example, in a traffic junction model, car A following car B in a one-lane road cannot move ahead of B at any point in time. Adding system constraints that can be obtained from the system specification to the superimposition process to reduce L_{sum} to L^P_{sum} is part of our future work. The more known constraints we add, the smaller L_{sum} is.

Experimental Simulation Results

We implemented the Boids model as a Java simulator to further understand the relationship among L^I_{whole}, L_{sum}, and L_ξ. We also analyzed how interactions of agents affect the size of L_ξ with respect to the size of L^I_{whole}. As L_{sum} suffers from state-space explosion, we varied the number of birds from four to ten, with equal numbers of ducks and geese. A difference between ducks and geese is maximum speed, which is two cells per time step for ducks and

three cells per time step for geese. Position and velocity of birds are initialized randomly. Both ducks and geese follow three behavior rules defined above. Given an initial system state, at some point of time t, the system will arrive in a state that has already happened at time $t' < t$ because the number of possible system states is finite, even if it can be very large. Due to the assumption that agents' behavior rules do not change over time, the simulation loops afterwards. As a result, the size of L^I_{whole} is the number of distinct system states obtained from the beginning until time t. L_{sum}, on the other hand, is computed over all possible combinations of isolated agents' behaviors with respect to the initial system state. The initial system state belongs to both L^I_{whole} and L_{sum}.

For every experiment, we ran the simulation ten times and took the average number of states as shown in Table 3.3. The experiments are run using a 3.4GHz machine with 8GB RAM. As expected, the size of L_{sum} is large and increases exponentially with the

number of birds	**number of states**			$\frac{\mathbf{L_\xi}}{\mathbf{L^I_{whole}}}$	**execution time (s)**
	$\mathbf{L^I_{whole}}$	$\mathbf{L_{sum}}$	$\mathbf{L_\xi}$		
4	13	767	6	0.46	0.3
6	18	70,118	12	0.67	3.7
8	13	509,103	9	0.69	446.2
10	26	13,314,066	23	0.88	3,092.1

Table 3.3: Size of L^I_{whole}, L_{sum}, and L_ξ

number of birds. For instance, L_{sum} grows by 90 times when the number of birds changes from four to six. Another interesting observation is that L_ξ / L^I_{whole} tends to increase with the number of birds. In other words, more interactions (interdependences) of birds lead to more emergent property states that cannot be derived by summing the independent agents' behaviors. Furthermore, the number of common elements between L^I_{whole} and L_{sum} is small. Consequently, computation is wasted on calculating L^{NP}_{sum} states that are not feasible in practice. However, when the impact of neighboring agents on an agent is not strong, and then L^{NP}_{sum} tends to be small. For example, agents do not interact frequently because they

are far from each other, such as when the number of agents is much smaller than the number of cells in the environment. Finally, the execution time increases sharply when the number of birds grows slowly. For example, the simulation takes barely 0.3 seconds for 4 birds while that is more than 45 minutes for 10 birds. This finding is reasonable in the sense that L_{sum} increases exponentially in the number of birds, the significant difference in execution time is mainly due to the calculation of L_{sum}.

3.5 Reduction of State Space

Section 3.4 showed that compared to Kubik's work, even though we reduced the state space for L_ξ by eliminating the system states out of the system designer's interest, our proposed approach still significantly suffered from state explosion. From Equation 3.4, L_ξ is determined by checking whether L^I_{whole} is in L^P_{sum}. L^P_{sum} is derived from L_{sum}, which is in turn determined using a superimpose set operation that extrapolates all possible permutations of agents' behavior. The outcome of this extrapolation includes those permutations that do not exist in practice, and thus L_{sum} can be very large even for a small number of agents.

In this section, we further reduce the state space for L_ξ by excluding L_{sum}. We propose a new method for splitting L^I_{whole} into L_ξ and L_o, which is the intersection between L^I_{whole} and L^P_{sum} as shown in Figure 3.5.

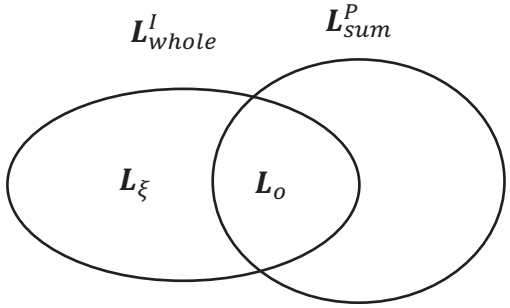

Figure 3.5: Emergent and Non-emergent Property States

3.5.1 Degree of Interaction as an Emergence Measure

The key idea is to use degree of interaction of agents as a measure of emergence. It is largely accepted that interaction is a key prerequisite of emergence [19, 44, 52]. Intuitively, the stronger interactions of agents are, the higher chance of emergence. Emergence happens when interactions of agents are "strong", i.e. their degree is larger than a given threshold. L_o is due to interactions that are not strong, and L_ξ is due to strong interactions. There are three possible categories of non-strong interaction, namely, no interaction (individual agent behavior), interaction that cancels out and has no overall effect on the system, and weak interaction. The first two have degree of zero, while the last category has degree of less than or equal to the pre-determined threshold.

We propose to use system state change as a measure of degree of agent interaction. This is because interactions trigger state changes. Intuitively, stronger interactions tend to make more significant changes on the state of the system. This idea is based on the *outcome* of an interaction rather than the process of the interaction. Note that, in the scope of this thesis, we only consider systems in which interactions finally induce some sort of state changes. Systems whereby interactions trigger agents to go through a decision process, but not result in any state changes are not considered.

Interaction arises from *neighboring rules* that define how an agent interacts with other agents, i.e. has at least one neighbor. For example, a bird changes its position because of coordination (collision avoidance, alignment, and cohesion) with other neighboring birds. Besides neighboring rules, an agent also has *individual rules* that govern the agent behavior when the agent acts individually without the neighbors. For example, a bird may not change its speed for a period during which it is alone in the environment. As a result, the difference between two system states $S(t)$ and $S(0)$ at time t and zero re-

spectively, $D(S(t), S(0))$, is due to individual rules, $D_I(S(t), S(0))$, and neighboring rules, $D_N(S(t), S(0))$.

$$D(S(t), S(0)) = D_I(S(t), S(0)) + D_N(S(t), S(0)) \tag{3.18}$$

$D_N(S(t), S(0))$ represents degree of interaction between agents. Because emergent properties arise from interaction, and agent interaction, in turn, arises from neighboring rules, $D_N(S(t), S(0))$ can be regarded as a measure or a criterion for determining emergence. Consider system state $S(t)$ in L^I_{whole}:

$$S(t) \in \begin{cases} L_o & \text{if } D_N(S(t), S(0)) = 0, \text{ no or cancel out interaction} \\ & \text{if } 0 < D_N(S(t), S(0)) \leq \delta, \text{ weak interaction} \\ L_\xi & \text{otherwise} \end{cases}$$

where δ $(0 < \delta < 1)$ is a pre-defined threshold that denotes degree of interaction. The larger δ we set, the smaller number of emergent property states we obtain. Algorithm 5 presents the pseudo-code for deriving L_ξ based on degree of agent interaction. L_o and L_ξ are initialized to empty sets (line 2 and line 3 respectively). The difference between every state in L^I_{whole} and the initial state is measured (line 5) and compared with the pre-determined threshold δ. If the difference is less than or equal to δ, the corresponding state is added to L_o. Otherwise, it is added to L_ξ, i.e. an emergent property state. The difference between $S(t)$ and $S(0)$ is accumulated from the difference between $s_{ij}(t)$ and $s_{ij}(0)$ for individual agents (line 9). The *for* loop terminates as soon as the accumulated difference is larger than δ (line 12).

$\mathbf{D_N(S(t), S(0))}$ Calculation

System state is composed of the states of all constituent agents and the states of the environment. Therefore, we define:

$$D_I(S(t), S(0)) = \frac{1}{n} \sum_{\forall i \forall j} d_I(s_{ij}(t), s_{ij}(0)) \tag{3.19}$$

Algorithm 5 Pseudo-code for L_ξ Calculation

```
1: procedure calculate_L_ξ
2:     L_o = ∅;
3:     L_ξ = ∅;
4:     set δ;                                   //set weak emergence threshold
5:     for each state S(t) ∈ L^I_whole do
6:         D(t,0) := 0;
7:         for each entity A in system do
8:             call d_A(t,0);                   //entity state difference time t and zero
9:             add d_A(t,0) to D(t,0);
10:            if D(t,0) > δ then               //L_ξ state
11:                add S(t) to L_ξ;
12:                exit for loop;
13:            end if
14:        end for
15:        if D(t,0) = 0 then
16:            add S(t) to L_o;                 //no or cancel out interaction
17:        else if D(t,0) ≤ δ then
18:            add S(t) to L_o;                 //weak interaction
19:        end if
20:    end for
21:    return L_ξ;
22: end procedure
```

$$D_N(S(t), S(0)) = \frac{1}{n} \sum_{\forall i \forall j} d_N(s_{ij}(t), s_{ij}(0)) + D_N(S_E(t), S_E(0)) \tag{3.20}$$

where n denotes the number of agents, $d_I(s_{ij}(t), s_{ij}(0))$ and $d_N(s_{ij}(t), s_{ij}(0))$ denote the difference between A_{ij}'s states in $S(t)$ and $S(0)$ due to individual rules and neighboring rules respectively, and $D_N(S_E(t), S_E(0))$ denotes the difference between states of the environment at time t and zero. Let

$$d(s_{ij}(t), s_{ij}(0)) = d_I(s_{ij}(t), s_{ij}(0)) + d_N(s_{ij}(t), s_{ij}(0)) \tag{3.21}$$

be the difference between A_{ij}'s states in $S(t)$ and $S(0)$ in total, i.e. due to both individual and neighboring rules.

From Equation 3.20, to calculate $D_N(S(t), S(0))$, we need $d_N(s_{ij}(t), s_{ij}(0))$ which requires $d(s_{ij}(t), s_{ij}(0))$ and $d_I(s_{ij}(t), s_{ij}(0))$ as from Equation 3.21. We propose a method

to measure $d(s_{ij}(t), s_{ij}(0))$ and $d_I(s_{ij}(t), s_{ij}(0))$. The calculation of $D_N(S_E(t), S_E(0))$ will be discussed after that. Because the state of an agent is characterized by the values of its attributes, which may have different units of measurement and different ranges, we need to normalize agent attributes to the same range [0,1] with no units. For example, the speed of a car may vary from 0 km/h to 100 km/h, while its travel time is measured in seconds. Interval [0,1] of normalized attributes ensures that agent state difference $d_N(s_{ij}(t), s_{ij}(0))$ and system state difference $D_N(S(t), S(0))$ due to neighboring rules are also in interval [0,1]. An attribute p is normalized using the following scaling formula:

$$p' = \frac{p - p_{min}}{p_{max} - p_{min}} \tag{3.22}$$

where p_{min} and p_{max} are the minimum and maximum values, respectively, of the attribute p. In addition, we assume that all agent attributes can be measured in a numerical form. Non-numerical attributes could be translated in to a numerical form. For example, consider color attribute, we can translate its values green and blue into 0.1 and 0.2 respectively. This translation is problem-specific, and we leave this issue for future work. Thus, $d(s_{ij}(t), s_{ij}(0)$ is defined as:

$$d(s_{ij}(t), s_{ij}(0)) = \frac{\sum_{p \in P_i} |p'(t) - p'(0)|}{|P_i|} \tag{3.23}$$

where P_i denotes the set of attributes for agents of type i, and $p'(t)$ denotes value of normalized attribute p at time t. Algorithm 6 presents the pseudo-code for the calculation of $d(s_{ij}(t), s_{ij}(0))$. This difference counts the differences of the values of attributes at time t and zero. Agent attributes are normalized in line 4.

$d_I(s_{ij}(t), s_{ij}(0))$ is due to individual rules, and defined as:

$$d_I(s_{ij}(t), s_{ij}(0)) = d(s, s_{ij}(0)) \tag{3.24}$$

where s is an agent state reached from $s_{ij}(0)$ after t steps considering only individual rules

Algorithm 6 Pseudo-code for $d(s_{ij}(t), s_{ij}(0))$ Calculation

```
1: procedure calculate_d(s_ij(t), s_ij(0))
2:     d(s_ij(t), s_ij(0)) := 0;
3:     for each attribute p ∈ P_i do
4:         add |(p(t) − p(0)|/(p_max − p_min) to d(s_ij(t), s_ij(0));
5:     end for
6:     d(s_ij(t), s_ij(0)) := d(s_ij(t), s_ij(0))/|P_i|;
7:     return d(s_ij(t), s_ij(0));
8: end procedure
```

for A_{ij}, i.e. A_{ij} is alone in the system.

$$s_{ij}(0) \xrightarrow{\text{individual rules}}{}^{t} s \tag{3.25}$$

If $s_{ij}(0) \xrightarrow{\text{individual rules}}{}^{t} s_{ij}(t)$, then then $d(s_{ij}(t), s_{ij}(0)) = d_I(s_{ij}(t), s_{ij}(0))$ and $d_N(s_{ij}(t), s_{ij}(0)) = 0$.

For example, consider the Boids model. A bird has two attributes: position, which is the location of the cell occupied by the bird, denoted by a 2D vector $(x(t), y(t))$, and velocity, denoted by a 2D vector $(\alpha(t), \beta(t))$. We assume that a bird does not change its velocity when flying alone, and visits certain cells along its path with corresponding to its velocity. Let $\sqrt{c}$ be the width/height of the environment, where c is the size, i.e. number of cells, of the square environment 2D grid. $s_{ij}(0) \xrightarrow{\text{individual rules}}{}^{t} s_{ij}(t)$ if and only if:

1. $x(t) - x(0) = 0 \ mod(\sqrt{c})$ and $y(t) - y(0) = 0 \ mod(\sqrt{c})$;

2. $(\alpha(t), \beta(t)) = (\alpha(0), \beta(0))$.

where *mod* is the modulo operation.

$\mathbf{D_N(S_E(t), S_E(0))}$ Calculation

The difference between states of the environment at time t and time zero is defined as followed:

$$D_N(S_E(t), S_E(0)) = min(D_{N,A_{ij}}(S_E(t), S_E(0))) \tag{3.26}$$

where $D_{N,A_{ij}}(S_E(t), S_E(0))$ denotes the difference of states of the environment at time t and time zero due to the behavior of agent A_{ij} only. Considering A_{ij} alone in the system, $D_{N,A_{ij}}(S_E(t), S_E(0))$ can be computed in a manner similar to Equation 3.23 with respect to all attributes of the environment.

The strength of emergence is measured using the degree of interaction of agents, and stronger interaction leads to a higher likelihood of emergence. Degree of interaction can be measured as the change of system state because interactions trigger state changes. Our degree of interaction is derived based on a number of entity attributes. As discuss in the next Section, it is not always straightforward to select these attributes. As a result, degree of interaction of agents at time t, $D_N(S(t), S(0))$, is measured as the state difference between $S(t)$ and $S(0)$. This difference consists of the difference between agents' states due to neighboring rules at time t and time zero, and the difference between environment states at time t and time zero (as shown in Equation 3.20). The difference between the two states of an agent is derived from the difference of values of the attributes of the agent at the two respective time points. Agent attributes are normalized so that we can combine attributes of different measurement units and magnitudes.

3.5.2 Evaluation

Similar to Section 3.4.4, in this section we evaluate the state space reduction technique, in terms of both theoretical complexity and experimental results.

Theoretical Analysis

The complexity of deriving L_ξ ($O(L_\xi)$) consists of two main parts: (1) complexity of calculating L^I_{whole} ($O(L^I_{whole})$), and (2) complexity of classifying L^I_{whole} into L_o and L_ξ

using the interaction degree ($O(classification)$).

$$O(L_\xi) = O(L^I_{whole}) + O(classification) \tag{3.27}$$

Clearly, O(L^I_{whole}) can be evaluated using Equation 3.16. Therefore, the main aim of this section is to define $O(classification)$. In contrast to previous approach, which involves L_{sum}, by considering the interaction degree as an emergence measure, we eliminate L_{sum}, and traverse all states in L^I_{whole} to check whether they are emergent or not. Given a state in L^I_{whole}, we compute the difference between this state and the initial state in relation to the behavior rules. The difference between these two system states is summed over the difference of individual birds contained in these states. Finally, the difference between two states of a particular bird is composed of the difference of its attributes in the two bird states. As a result, the complexity of checking whether a system state is emergent, is *linear* in the number of attributes. Thus the complexity of doing this checking for all states in L^I_{whole} is linear in the cardinality of L^I_{whole} and the average number of attributes of birds. Let $\bar{P}$ be the average number of bird attributes. We have

$$O(classification) = O(|L^I_{whole}|\bar{P}) \tag{3.28}$$

Compare $O(classification)$ with $O(L_{sum}) = O((xy)^{n'}s^n)$ (as shown in Equation 3.17). As reasoned in Section 3.4.4, $O(L^I_{whole})$ is typically much smaller than $O(L_{sum})$. Hence, if $\bar{P}$ is small, $O(classification)$ would be also much smaller than $O(L_{sum})$, i.e. the proposed technique for reducing the state space to be searched is superior to the original approach, which involves L_{sum}. For example, in our extended Boids model, $\bar{P}$ is only two; the calculation of L_ξ should be theoretically reduced remarkably. The experimental results presented below will support this statement.

Experimental Simulation Results

Due to state-space explosion, simulation experiments in the previous section are limited

to ten birds. Even with such a small number of birds, L_{sum} already exceeds ten millions. In this section, by eliminating L_{sum}, our experiments scale up to a larger number of birds (1,024 birds) and a larger environment (128 x 128 grid). We set δ to 0.1 to consider that emergence occurs when degree of interaction of birds is larger than 0.1. Other values of δ are examined later. Position of birds in the grid is the only attribute concerned in calculating degree of interaction as position is the result of applying all behavior rules, thus changes in position demonstrate how strong birds' interactions are. The simulator is run using a 2.4GHz machine with 3GB RAM. Table 3.4 shows experimental results for different numbers of birds with grids of 16 x 16, 32 x 32, 64 x 64, and 128 x 128. We keep the population of ducks and geese equal. The purpose of the experiments is to analyze the relationships among L^I_{whole}, L_o, and L_ξ. We are also interested in evaluating the scalability of the new method of determining emergence.

The first observation is that when the number of birds increases doubly, L^I_{whole} grows as expected, but soon drops. For example, as can be seen in the table, for 128 x 128 grid, L^I_{whole} decreases sharply from 7,497 to 4,072 when the number of birds increases from 128 to 1,024. Similarly, L^I_{whole} changes noticeably from 3,803 to 1,536 when bird population varies from 128 to 1,024 in 64 x 64 grid. This tendency is probably because that a larger number of birds encourage more interactions of birds, thus making the birds' movement more structured. Consequently, the more self-organized movement makes the simulation repeat faster.

Second, for all four sizes of the environment, system states due to no/cancel out interaction and weak interaction are small, in particular maximum of 38, and tend to decrease with the number of birds. This further clarifies the close relationship between bird population and degree of bird interaction that more birds lead to more interactions, hence smaller L_o. Furthermore, the number of system states due to no/cancel out interaction tends to

number of birds	L^I_{whole}	L_o		L_ξ	execution time (s)
		no/cancel out	weak		
16 x 16 grid					
4	18	2	1	15	small
8	44	1	2	41	small
16	65	1	1	63	small
32	127	1	1	125	0.2
64	220	1	1	218	0.6
32 x 32 grid					
4	47	3	6	38	small
8	72	2	5	65	small
16	107	2	4	101	small
32	287	1	2	284	0.3
64	454	1	2	451	1.3
128	437	1	2	434	6.4
256	389	1	2	386	23.9
64 x 64 grid					
4	50	7	12	31	small
8	83	4	7	72	small
16	211	1	11	199	0.1
32	271	1	8	262	0.3
64	2,573	1	6	2,566	9.3
128	3,803	1	6	3,796	66.2
256	2,340	1	5	2,334	184.5
512	1,785	1	5	1,779	780.4
1,024	1,536	1	5	1,530	5,392.6
128 x 128 grid					
4	133	17	21	95	small
8	157	3	19	135	small
16	643	5	16	622	0.2
32	1,158	2	19	1,137	1.3
64	3,037	1	15	3,021	12.8
128	7,497	1	12	7,484	151.7
256	6,871	1	13	6,857	941.8
512	5,038	1	11	5,026	5,394.3
1,024	4,072	1	11	4,060	32,266.8

Table 3.4: Varying Number of Birds and Environment Size with δ of 0.1

reach one. This only state is actually the initial state. In other words, there is no system state caused by interactions that cancel out when the bird population is large.

Third, L_o is small compared to L_ξ, especially when the number of birds is large. For example, $\frac{L_o}{L_\xi}$ is less than 1% when bird population is 64, 256, 512, and 1,024 in grid of 16 x 16, 32 x 32, 64 x 64, and 128 x 128 respectively. There are two possible reasons for this. The first reason is that $\delta = 0.1$ is a small value. If δ is set to a larger value, we likely retrieve less emergent property states but with a higher degree of interactions. However, there is a risk that some appealing emergent properties may reside in the other part that we do not consider. The second reason is that more interactions of birds when the group becomes more crowded amplify degree of bird interaction, thus making the degree larger than δ for most system states.

In addition, by examining the execution times, we can see that the new proposed method of identifying emergent property states is much more efficient than the original method. In fact, eliminating L_{sum} enables experiments of much larger bird populations and environment sizes (1,024 birds and 128 x 128 grid in the new method compared to 10 birds and 8 x 8 grid in the original method). In particular, for 1,024 birds and 128 x 128 grid, the experiment took about nine hours to run through 4,072 system states and classify them into non-emergent and emergent property states based on degree of interaction. Compared to the original approach of calculation of L_ξ, which took about one hour for only ten birds and 8 x 8 grid as shown in Table 3.3, we can see the high efficiency of the proposed technique for reduction of state space.

To understand how strong interactions of agents are, and the relationship between L_ξ and δ, we do experiments for different numbers of birds, different values of δ in a 16 x 16 grid as shown in Table 3.5. The first obvious observation is that L_ξ mainly occurs when degree of interaction lies in two ranges [0.2, 0.3] and [0.3, 0.4]. For example, consider 64 birds, these two ranges accounts for about 97% of emergent property states. In addition, [0.3, 0.4] seems to have more emergent property states than [0.2, 0.3] when bird population

number of birds	$\mathbf{L^I_{whole}}$	$\mathbf{L_\xi}$					
		0.0	**0.1**	**0.2**	**0.3**	**0.4**	**0.5**
4	18	3	3	6	5	1	0
8	44	3	4	21	13	3	0
16	65	2	2	22	33	6	0
32	127	2	2	20	93	10	0
64	220	2	1	31	183	3	0

Table 3.5: Size of L^I_{whole} and L_ξ for Different Numbers of Birds, Different δ with 16 x 16 Grid

increases. For instance, the ratio of L_ξ in [0.3, 0.4] and [0.2, 0.3] grows from 0.8 to 5.9 when the number of birds increases from 4 to 64 respectively. One explanation is that birds must conform to some pre-determined behavior rules to interact with others, thus changes of the system state caused by their interactions tend to follow some distribution in relation to the behavior rules. Based on the reasoning above, our hypothesis is that *the behavior rules of birds is observed to follow a particular distribution of degree of birds' interaction but this has to be further investigated.*

3.6 Summary

In this section, we formalized emergence using an extended cooperating array grammar system so as to verify the existence and the size of emergent property states in multi-agent systems. In weak emergence, the set of emergent property states for a given system (L_ξ) is derived by taking the difference between the set of observed system states due to agent interactions (L_{whole}), and the set of system states obtained by combining the states of individual agents (L_{sum}). However, this broad definition suffers from state explosion.

We introduced a tighter definition of weak emergence to reduce the size of the system state space. In studying a system, users focus only on modeling agent interactions of interest ($L^I_{whole} < L_{whole}$). Though all combinations of L_{sum} are mathematically possible,

a large number of these system states may not be feasible because of system constraints. Thus, L_{sum} can be reduced to L^P_{sum}, i.e. set of possible combinations of agents' states.

We extended Kubik's approach to a grammar-based formalism of emergence that models agents of different types, mobile and static agents, as well as open systems with agents arriving and departing over time. Theoretical analysis reveals a number of observations, e.g. the complexity of L_{sum} increases exponentially with the number of agents. These observations were also verified experimentally using a Boids model with two types of birds. In addition, we showed how a known emergent property such as bird flocking can be extracted from the emergent property states. In terms of system state-space size, preliminary experimental results show that our simulator can handle L_{sum} of about 10^8 states. L_{sum} can be further reduced by identifying system-specific constraints that eliminate impossible states when summing individual agents' states.

To reduce of the state space to be searched, we proposed an alternative to determining L_ξ without the calculation of L_{sum}. The key idea is to divide L^I_{whole} into two subsets: L_ξ - the set of emergent property states and L_o - the set of system states in which interactions of agents are weak, cancel out, or do not happen at all, using degree of interaction between agents. This degree is measured as the difference between a system state under examination and the initial state. The stronger agent interaction is, the more the system state changes with respect to the initial state. The Boids model with various bird populations and grid sizes were experimented. The results showed that the proposed reduction technique based on the interaction degree is efficient. The simulation for 1,024 birds in 64 x 64 grid took roughly the same amount of time (around one hour) compared to the original approach (involving L_{sum}) for only ten birds in 8 x 8 grid. Lastly, we put forward a hypothesis that behavior rules defines a particular distribution of interaction degree.

Chapter 4

Example: Deadlock Emergence in Concurrent Programs

One of the most challenging limitations of our work is that the proposed approach is relative to the model of the system to be examined. Given a complex system, we have to abstract the system, get rid of unnecessary details, and then come out with a model. This model is the best representation of the system in relation to the designer's interest. However, the designer may not take into account behavior rules that contribute to emergence. In other words, there may be emergent properties that occur in the system but not in the model of the system. Consequently, our approach returns only a proportion of the whole emergent property states. This proportion could be small compared to the part that is not modeled. In this section, we try to lighten the limitation above by considering the situation when the input to our approach is a concrete specification rather than a model of the system. We analyze deadlock emergence in concurrent programs as an example.

The focus is to validate the proposed approach by deriving the set of emergent property states, including deadlock states, and evaluate the scalability of the approach in context

of multi-threaded programs. It is shown that our approach precisely detect all potential deadlock states. Besides the deadlock states, the approach also detects other system states that are due to interleaving among threads. These states are a useful source to determine other properties of the system such as exceptions and bugs that facilitate the program development. Although the proposed approach encounters state-space explosion because it has to examine all possibilities of threads' interleaving, its scalability could be significantly improved with further investigations.

4.1 Multi-threaded Programs as Problem Specification

Rather than modeling the problem to have a model as the input to our approach, for example the Boids model in Section 3.4, we are provided with a system specification. In particular, the specification is a multi-thread program coming from a user. A multi-threaded program is largely different from the Boids model in several aspects as shown in Table 4.1.

Criteria	**Boids Model**	**Multi-threaded Programs**
Representation	system's model	system's specification
Number of component types	less than number of boids	equal number of threads
System behavior	deterministic	non-deterministic
Component interactions	component-component	component-environment
Emergent properties	flocking properties	deadlock, livelock, etc.

Table 4.1: The Boids Model vs. Multi-threaded Programs

In contrast to the Boids model, which is a model of a group of birds, a multi-threaded program is regarded as a concrete specification of the system. A multi-threaded program consists of multiple threads interacting with each other. Each thread has local variables

and statements that define how the thread acts and interacts with the other threads of the program. Different threads typically have different local variables or statements, thus belonging to different types of thread. Therefore, the number of thread types approximates to the number of threads. This is contrary to the Boids model in which several birds are often grouped into a type. Threads are assumed to have the same priority and interleave, allowing their execution in all possible orders relative to each other. This arbitrary execution leads to a large number of possible program states and execution paths. The interleaving model of computation makes the multi-threaded program *non-deterministic*. In other words, different simulations starting with the exact same initial configuration may result in different results. It is important to note that the Boids model and The Game of Life are *fully deterministic* systems. In fact, both deterministic and non-deterministic systems could exhibit emergent properties, as long as there are interactions between components.

The interactions of threads are manifested in changes of the shared environment, which is visible to all threads. Typically, the environment is a set of shared variables. Shared variables can be static variables (Java), global variables (C), and locks in the context of concurrent programming, to name a few. These interactions are sort of *indirect interaction* or *component-environment* since effects caused by the interactions are presented by a third party, i.e. the environment, instead of the two interacting threads. The interactions in the Boids model, on the other hand, are *direct* or *component-component*. A boid must initially inquire to figure out the status of its neighboring boids, and then changes its state accordingly. The changes made by the boid in turn directly influence the behavior of its neighbors.

Deadlock is an *undesired emergent property* in concurrent systems. A deadlock is a permanent blocking of a set of processes that either compete for resources (data resources or communication messages) with each other. Figure 4.1 depicts a deadlock situation

with two processes sharing two resources. Process 1 is in possession of resource 1 and

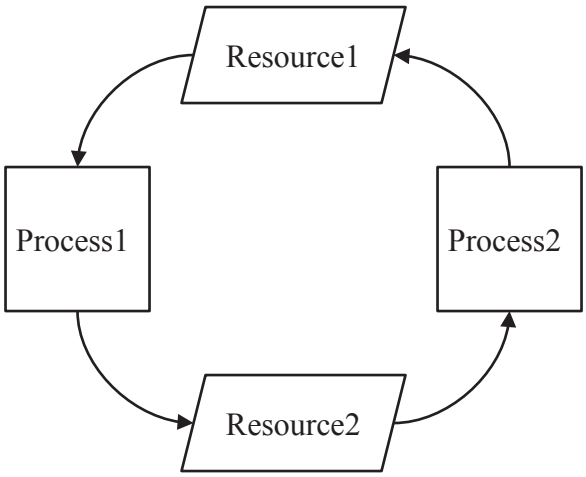

Figure 4.1: Deadlock with Two Processes and Two Shared Resources

requires additional resource 2, process 2 requires additional resource 1 and is in possession of resource 2; neither process can continue.

Although deadlock arises from the interactions of processes, it cannot be derived from the behavior of individual processes, and is often unpredictable until run time. Deadlock makes a system less credible and harder to predict. However, currently existing techniques can only address a proportion of the whole deadlock population. Model checking, in particular, only detects certain deadlocks in accordance to a given deadlock specification. There is no guarantee that other types of deadlock that have not been known do not exist.

4.2 Grammar-based Formalism of Multi-threaded Programs

In this section, we formalize a simple multi-threaded program using our grammar-based approach. Figure 4.2 provides a Java code segment that potentially leads to deadlock. The code segment consists of two threads sharing two variables. Thread 1 and thread 2

```
final Object resource1 = "resource1";
final Object resource2 = "resource2";
Thread t1 = new Thread() {
   public void run() {
      // Lock resource 1
      synchronized (resource1) {
         synchronized (resource2) {
         }
      }
   }
};
Thread t2 = new Thread() {
   public void run() {
      // Lock resource 2
      synchronized (resource2) {
         synchronized (resource1) {
         }
      }
   }
};
t1.start(); t2.start();
```

Figure 4.2: Two Threads Sharing Two Variables

share two variables: $resource1$ and $resource2$. These two resources are *mutual exclusive* in the sense that each resource can be used by only one thread at a time. Note that these two resources are a representation of the lock concept in concurrent programming. Thread 1 needs to hold resource 1, then resource 2, and finally releases both resources. Similarly, thread 2 requires resource 2, then resource 1, and finally releases both resources. The competition between two threads for the shared resources could result in deadlock in which thread 1 possesses resource 1 and wait infinitely for resource 2 which is possessed by thread 2. This situation could happen in practice when a thread, after successfully requesting its first needed resource, pauses for a bit, for example because of the statement

Thread.sleep(50) in Java, simulating some file I/O, and the scheduler gives the other thread a chance to run.

Because emergence is due to the interactions of the threads, and these interactions are manifested in changes of the environment, we simplify threads to consider only local attributes and statements that affect the environment in one way or another. The program above is formalized as:

$$GBS_{thread} = (V_A, V_E, A_{11}, A_{21}, S(0))$$

where A_{11} denotes thread 1, A_{21} denotes thread 2, and V_A, V_E, and $S(0)$ have the same meaning as defined in Table 3.1. The environment is an array of two elements $E = [e_1, e_2]$, where $e_i \in \{0, 1, 2\}$, where $e_i = 0$ denotes resource i $(i = 1, 2)$ is free, $e_i = 1$ denotes resource i $(i = 1, 2)$ is hold by thread 1, and $e_i = 2$ denotes resource i $(i = 1, 2)$ is hold by thread 2. The state of the environment is characterized by the values of the shared variables, i.e. e_1 and e_2. Thread 1 is defined as follows:

$$A_{11} = (P_1, R_1, s_{11}(0))$$

where P_1 is the set of properties (local variables), R_1 is the set of behavior rules, and $s_{11}(0)$ is the initial state of thread 1. Clearly, P_1 is empty since thread 1 has no local variables. Thread 1 has three behavior rules: requests resource 1, additionally requests resource 2, and finally releases the two resources. Hence, $R_1 = \{e_1 = 1, e_2 = 1, e_1 = e_2 = 0\}$. The state of a thread is characterized by the values of variables local to that thread. Because thread 1 has no local variables, its states, in particular its initial state, is always empty. Similarly, we can define the grammar-based formalism for thread 2. The state of the program comprises states of all threads and the state of the environment representing the shared variables. Two threads have an empty state, hence the state of the whole program

is equivalent to the state of the environment. Initially, $S(0) = (0, 0)$ as both resources are free at the beginning.

4.3 Asynchronous Composition of FSAs of Threads

Given a multi-threaded program, our purpose is to determine L^I_{whole}, and then derive L_ξ. L^I_{whole} is a set of all reachable states with respect to the initial state. A *reachable* state refers to a state that can be reached from the initial state after a finite number of transitions. Note that a reachable state may or may not involve interactions among threads. L_ξ is a subset of L^I_{whole} that consists of reachable states with interactions (via the environment - shared resources) of at least two threads. A subset of L_ξ is composed of deadlock states. A deadlock state is a reachable state that is not terminating and cannot move out, i.e. has no successors, because of interactions of threads. Figure 4.3 shows a diagram of four state spaces, from the outer to the inner: all states due to interleaving of threads, L^I_{whole} (reachable states, with and without interactions), L_ξ (reachable states with interactions), and deadlock states.

Compared to the Boids model discussed in Section 3.4, in multi-threaded programs, it is not feasible to use simulation to obtain L^I_{whole}. In the Boids model, we assume that all boids update their state synchronously. As a result, given a state, the state of the system at the next time step is deterministic. However, a multi-threaded program is non-deterministic. The calculation of L^I_{whole} has to take into account all possibilities of scheduling among threads that could be very large. A promising approach for handling the non-deterministic scheduling in multi-threaded programs is to model threads as Finite State Automata (FSAs) and use asynchronous composition to integrate them together as a whole. A thread can be modeled as a FSA (q, q_0, l, δ, q_f), where

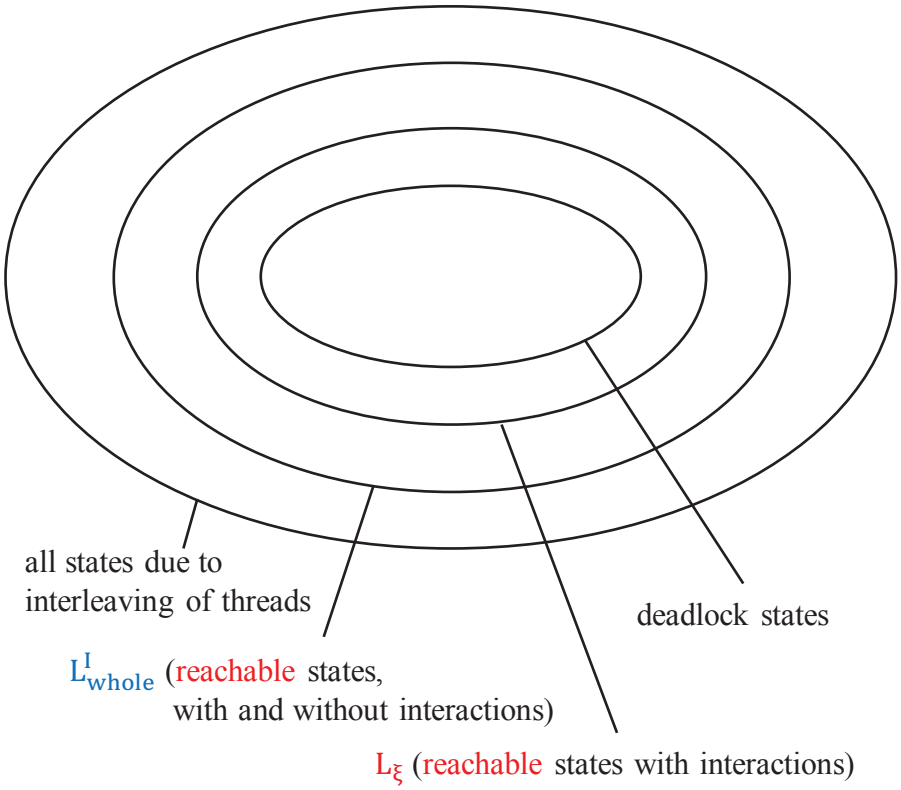

Figure 4.3: State Diagram of Deadlock Emergence

.

- q is a finite, non-empty set of states of the thread.
- q_0 is a set of distinguished initial states of the thread with $q_0 \subseteq q$.
- l is a finite set of labels of transitions that change the thread from one state to another.
- δ is a set of transitions with $\delta \subseteq (q \times l \times q)$.
- q_f is a set of final states of the thread with $q_f \subseteq q$.

Usually, the cardinality of q_0 is one, i.e. the thread has only one initial state, and l includes statements of the thread. The asynchronous composition of two FSAs A and B is a FSA $A||B = (q, q_0, l, \delta, q_f)$, where

- q is the Cartesian product $A.q \times B.q$

- q_0 is $\{(a_0, b_0) \in q \mid a_0 \in A.q_0 \wedge b_0 \in B.q_0\}$
- l is the union $A.l \cup B.l$
- δ is $\{((a_1, b), l, (a_2, b)) \in q \times l \times q \mid (a_1, l, a_2) \in A.\delta \wedge b \in B.q \wedge \neg me(l, b)\} \cup$
  $\{((a, b_1), l, (a, b_2)) \in q \times l \times q \mid a \in A.q \wedge (b_1, l, b_2) \in B.\delta \wedge \neg me(l, a)\}$
- q_f is $\{(a_f, b_f) \in q \mid a_f \in A.q_f \wedge b_f \in B.q_f\}$

$A.q$ and $B.q$ denote the set of states of FSA A and B respectively. $A.q_o$, $B.q_o$, $A.l$, $B.l$, $A.\delta$, $B.\delta$, $A.q_f$, and $B.q_f$ have the similar meanings. $me(l, s) = true$ iff l contains any mutually exclusive variable that is being hold by s. The corresponding FSAs for thread 1 and thread 2 are presented in Figure 4.4. The asynchronous composition of these two FSAs is presented in Figure 4.5.

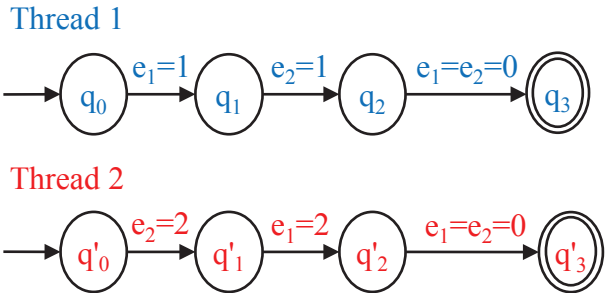

Figure 4.4: FSAs of Thread 1 and Thread 2

Consider states that are crossed in Figure 4.5. These states are not reachable from the initial state because of the constraints among threads. For example, state (q_2, q'_1) in which thread 1 is holding resources 1 and 2, and thread 2 is holding resource 2, cannot be reached from its predecessors, neither (q_2, q'_0) nor (q_1, q'_1). In fact, this state is invalid since resource 2 cannot be shared by both threads at the same time. L^I_{whole}, therefore, comprises

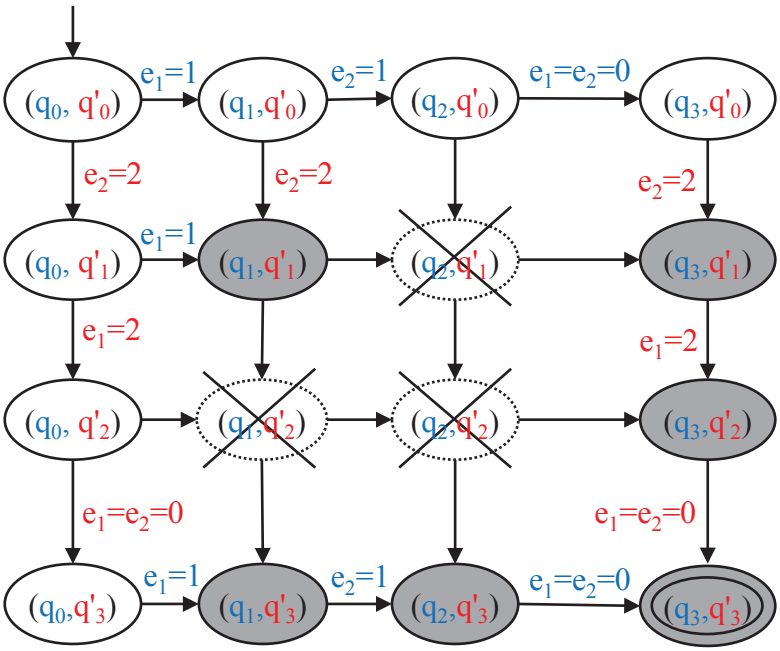

Figure 4.5: Asynchronous Composition of FSAs of Thread 1 and Thread 2

all other states, i.e. non-crossed states, as shown in the figure.

$$L^I_{whole} = \{(q_0, q'_0), (q_1, q'_0), (q_2, q'_0), (q_3, q'_0),$$
$$(q_0, q'_1), (q_1, q'_1), (q_3, q'_1),$$
$$(q_0, q'_2), (q_3, q'_2),$$
$$(q_0, q'_3), (q_1, q'_3), (q_2, q'_3), (q_3, q'_3)\}$$

Consider state $S \in L^I_{whole}$, that are not filled with grey in the figure. We can see that $D_N(S(0), S) = 0$, where $S(0)$ is the initial state (q_0, q'_0), because S can be reached from S(0) by applying individual behavior rules of only one thread. For example, starting from (q_0, q'_0), the system can move to state (q_0, q'_3) if thread 2 is scheduled to execute in the following sequence: requests resource 2, requests resource 1, and releases both resources 1 and 2. In this situation, only thread 2 acts and there is no interaction between thread 1 and thread 2. Consequently, (q_0, q'_3) is not an emergent property

state. Applying the similar argument for the other states not filled with grey, we have $L_0 = \{(q_0, q'_0), (q_1, q'_0), (q_2, q'_0), (q_3, q'_0), (q_0, q'_1), (q_0, q'_2), (q_0, q'_3)\}$. The remaining states that are filled with grey involve interactions of threads, thus are likely to be emergent property states. Whether a state of these states is emergent or not depends on the pre-determined value of δ, which represents degree of weak interaction. We first show that these states involve interactions of threads, and discuss δ after that. We pick up (q_1, q'_1), which is two time steps from the initial state, as an example. The state of the program consists of the states of two threads and the state of the environment. The threads have an empty state since they do not have any local variables. The environment includes two shared variables: resource 1 (e_1) and resource 2 (e_2).

$$\begin{aligned} D_N((q_0, q'_0), (q_1, q'_1)) &= D_N((0,0),(1,2)) \\ &= min(D_{N,A_{11}}((0,0),(1,2)), D_{N,A_{21}}((0,0),(1,2))) \\ &= min(1,1) = 1 \end{aligned}$$

Note that, the difference between the two states is only counted for the change of the state of the environment. Similarly, other states S filled with grey have $D_N(S(0), S) > 0$. Hence, if we set δ close to zero, then the set of emergent property states includes all states filled with grey $L_\xi = \{(q_1, q'_1), (q_3, q'_1), (q_3, q'_2), (q_1, q'_3), (q_2, q'_3), (q_3, q'_3)\}$.

In particular, we obtain a known emergent property state that exhibits deadlock. (q_1, q'_1) is a deadlock state since it is a non-terminating (final) state and has no successor. The deadlock happens when thread 1 holds resource 1 and waits infinitely for resource 2 while thread 2 holds resource 2 and is infinitely waiting for resource 1. The two threads interact with each other through changing the state of the environment that involves the states of the two resources.

To analyze the application of the proposed approach to deadlock emergence, we do

experiments for different numbers of threads of two main types. A thread of type 1 requests resource 1, then resource 2, and finally releases two resources. Conversely, a thread of type 2 requests resource 2, then resource 1, and finally releases two resources. We consider case studies presenting different situations in which threads of the two types compete for two shared resources. This competition potentially leads to deadlock. The case study of two threads, thread 1 and thread 2, is what we discussed above. For three threads, we add one more thread, thread 3 of type 1. Therefore, deadlock may occur between thread 1 and thread 2, or between thread 3 and thread 2. Similarly, for four threads, we add the forth thread, thread 4 of type 2. Deadlock in this case, becomes more complicated, that can be due to thread 1 and thread 2, thread 1 and thread 4, thread 3 and thread 2, or thread 3 and thread 4. Similarly, we can have up to twelve threads. Whichever the case, each thread has no local variables, three transition rules, and consequently four states. These four states compose of the initial state, the final (stopping) state, and two intermediate states in between. Table 4.2 shows the numbers of states obtained and simulation time using asynchronous composition of FSAs. The simulator is run using a 2.4GHz machine with 3GB RAM.

number of threads	**all states due to interleaving**	$\mathbf{L^I_{whole}}$	$\mathbf{L_\xi}$	**Deadlock Emergence**	**execution time (s)**
2	16	13	6	1	small
3	64	36	26	4	small
4	256	96	83	16	small
5	1,024	240	224	48	0.1
6	4,096	592	573	144	0.3
7	16,384	1,408	1,386	384	0.8
8	65,536	3,328	3,303	1,024	3.3
9	262,144	7,680	7,652	2,560	18.0
10	1,048,576	17,664	17,633	6,400	79.6
11	4,194,304	39,936	39,902	15,360	404.7
12	16,777,216	90,112	90,075	36,864	1,463.0

Table 4.2: Varying Number of Threads

As we can see from Table 4.2, the number of all states due to interleaving increases exponentially in the number of threads. This number is the product of the number of states of individual threads (16 = 4 x 4, 64 = 4 x 4 x 4, 256 = 4 x 4 x 4 x 4, and so on). This is understandable because each thread can run at any point of time (we assumed that all threads have the same priority). Clearly, we are considering the worst cases in which all possibilities of threads' interleaving are taken into account. In practice, the state space of the problem to be traversed is probably much smaller because there could be constraints among threads. For example, threads' interleaving is not arbitrary. Instead, there is an order among them in the execution flow of the program. In addition, we might think of some techniques to reduce this state space to enable programs with larger numbers of threads. One direction is to do research on techniques that are employed to reduce state space in model checking.

Another observation is that the number of unreachable states grows rapidly with the number of threads. For example, there are three unreachable states for two threads, while that number is 28 for three threads, 160 for four threads, and 16,687,104 for twelve threads. One explanation is that when the number of interacting threads increases, the number of states that are invalid due to constraints of the manipulating order of the shared resources grows up shapely. As a result, the state space to be searched is significantly reduced.

Furthermore, the ratio $\frac{L_\xi}{L^I_{whole}}$ tends to increase when the number of threads increases. Reasonably, this is because more threads lead to more interactions, hence more emergent property states in comparison to the whole population of reachable states. Finally, deadlock emergence increases exponentially with the number of threads. For example, the number of deadlock states changes shapely from 1 to 36,864 when we increase the number of threads from two to twelve. More specifically, for the case study of three threads, we detected two deadlock states between thread 1 and thread 2, and two deadlock states between thread 3

and thread 2. Similarly, for four threads, we detected four deadlock states between thread 1 and thread 2, thread 1 and thread 4, thread 3 and thread 2, and thread 3 and thread 4. These deadlocks states are due to the competition among threads for the shared resources. The resources play the role of a third party, the environment that reflects the interactions of the threads. These experimental results are generated using our simulator and verified by hands as well. We also intend to analyze our approach for larger problem sizes, i.e. larger numbers of threads.

4.4 Comparison with Modeling Checking

Model checking is known as the most successful technique for automated verification [9]. Given a system model, typically in the form of a state transition graph, and a formal temporal specification of the property of interest, modeling checking verifies whether the system model satisfies the property or not. If the property does not hold in the system model, counterexamples are returned.

After decades of model checking research with a huge number of efforts, model checking is quite mature and has several variants, mainly including *explicit-state model checking* and *symbolic model checking* [30]. The former is based on *explicit state search* where progress is made one state at a time, while the latter is based on *symbolic search strategy* [56] that examines sets of states in each step. More importantly, explicit-state model checking often deals with asynchronous model of execution in which the execution of independent transitions can be interleaved in all possible orders. This kind of execution model is very popular in software, in particular in concurrent programs. In contrast, symbolic model checking aims to synchronous model of execution in which all components are supposed to progress simultaneously, for example in many hardware systems. Furthermore, explicit-state model

checking uses linear temporal logic as in SPIN [46] and Java Pathfinder [87] model checkers, while symbolic model checking is based on Computation Tree Logic as in NuSMV [1] model checker. Table 4.3 summarizes a comparison between the two main paradigms of model checking. Because explicit-state model checking usually assumes asynchronous execution model, and our multi-threaded program is also asynchronous, i.e. supports arbitrary scheduling order among threads, we compare our approach with the explicit-state model checking only.

Criteria	**Explicit-state Model Checking**	**Symbolic Model Checking**
Search Strategy	explicit - one state/step	symbolic - sets of states/step
Execution Model	asynchronous (software)	synchronous (hardware)
Temporal Logic	Linear Temporal Logic (LTL)	Computation Tree Logic (CTL)
Tools	SPIN, Java Pathfinder	NuSMV

Table 4.3: Explicit-state Model Checking vs. Symbolic Model Checking

In explicit-state model checking, a system is typically modeled as a Kripke structure. A Kripke structure is basically a graph whose nodes represent the reachable states of the system and whose edges represent state transitions. Kripke structures that model the system and the negation of the LTL formula specifying the property to be studied respectively are both converted into Buchi automata as the next phase of the verification procedure. A Buchi automaton is a type of automata that extends a finite automaton to infinite inputs. It accepts an infinite input sequence if and only if there exists a run of the automaton that visits (at least) one of the final states infinitely often. For simplicity, we refer model checking to explicit-state model checking afterwards.

A comparison between model checking and our approach regarding software verification is presented in Table 4.4. Our approach is, to some extent, more general than model checking in the sense that we determine a set of all emergent properties due to concurrency rather than verifying a specific property as model checking aims to. Clearly, this set

Criteria	Model Checking	Our Approach
Objective	verify a specific property, e.g. deadlock	determine emergent property states due to concurrency
Knowledge of Property	a formal specification of the property to be verified	no prior knowledge of emergent properties
State Space Size	state-space explosion	state-space explosion

Table 4.4: Model Checking vs. Proposed Approach

includes states that exhibit properties examined in model checking, such as deadlock. In addition, while our approach does not require prior knowledge of emergent properties, model checking needs a formal specification of the property it is looking at. Finally, although the proposed approach and model checking both suffer from state-space explosion, sources of the problem are different. In our approach, the complexity mainly comes from two phases:

- Transform threads into FSAs: *linear complexity* in the number of threads, and the number of local variables and statements.
- Asynchronously compose FSAs into a single one: *exponential complexity* in the number of concurrent threads.

On the other hand, the state-space explosion in model checking is due to four main phases:

- Model a program as a Kripke structure (graph): *exponential complexity* in the number of concurrent threads, threads' behavior (statements), and the number of variables specifying the state of the system.
- Transform the Kripke graph to a Buchi automaton: *linear complexity* in the number of nodes of the Kripke graph.
- Generate a Buchi automaton for the negated LTL formula: *exponential complexity* in the size of the LTL formula.

- Generate the product of the two Buchi automata: *linear complexity* in the size of the two automata.

We can see that in both model checking and our approach, the number of concurrent threads and the number of threads' characteristics, i.e. local variables and statements, play the key role in the state-space explosion. On the other hand, our approach only considers variables and statements with respect to the interactions of threads, i.e. are related to the shared variables. Consequently, the state-space explosion could become trivial compared to model checking. Table 4.5 provides the number of states the approaches, model checking and our approach, have to traverse to detect emergent property states and deadlock states respectively. As shown in Table 4.5, vertically, the number of states visited in model

number of threads	**Model Checking**		**Our Approach**	
	visited states	**run time (s)**	**visited states**	**run time (s)**
2	34	small	16	small
3	254	small	64	small
4	1,675	1.0	256	small
5	9,681	4.0	1,024	0.1
6	52,407	19.0	4,096	0.3
7	263,362	82.0	16,384	0.8
8	1,272,025	375.0	65,536	3.3

Table 4.5: State Space Examined and Run Time in Model Checking and Our Approach

checking goes up exponentially with the number of threads. This is because the tree to be searched becomes wider, i.e. has more branches, when the number of threads increases. Our approach also encounters the space explosion, but at a slower pace, particularly about half of that in model checking. Horizontally, our approach is superior to model checking in terms of the number of states to be visited. One explanation is that the grammar-based formalization eliminates unimportant details and only considers parts that relate to the interleaving among threads. Model checking, however, considers every statement of the program as a trigger of state transition, thus resulting in a tree with a much higher depth

of search. For example, the statement *public void run()* is ignored in our approach, but is regarded as a transition trigger in model checking.

In addition, we observe that our approach takes much less time to run than model checking because the number of visited states in our approach is much smaller than that in model checking. For example, for eight threads, our approach (execution time of 3.3 seconds) is more than 100 times faster than model checking (execution time of around 375 seconds). However, the execution time of the two approaches increases rapidly with the number of threads. In particular, the execution time of model checking seems to grow up by a multiplier of four when the number of threads increases by one. As a result, state-space explosion of analyzing deadlock in large concurrent problems is still a big issue for both model checking and our approach; and further work is required.

4.5 Summary

With the advances of computer technology, highly concurrent systems are being developed. The verification of such systems is a challenging task, as their state space grows exponentially with the number of processes. In addition, existing studies only verify a few of known properties due to interleaving among concurrent processes, such as some known types of deadlock. Other types of deadlock that have not been seen before are simply ignored. In this section, we explore the application of the proposed space reduction mechanism to concurrent program verification. Given a multi-threaded program, the set of system states containing both known and unknown emergent property states, such as deadlock, can be determined. Threads are modeled as Finite State Automata (FSAs) and asynchronous composition of all FSAs returns L^I_{whole} - a set of program states reachable from the initial state. From this set, we derive L_ξ and observe an interesting emergent property state:

a deadlock state. This approach compliments model checking where the main goal is to verify predetermined behavioral properties of a given system.

Chapter 5

Conclusion and Future Work

5.1 Thesis Summary

Emergent properties are a distinguishing feature of complex systems, and become pervasive when system complexity increases in terms of system size, component types, and the interactions of components. Due to its unpredictable nature, emergence makes a system less credible and more difficult to design, analyze, and control. However, emergence is not wholly undesirable; instead it can be beneficial to the system exhibiting it. A formal definition, identification, and understanding the cause-and-effect of emergence are a key to developing and engineering more complex, but more robust systems. Unfortunately, there is still a lack of consensus on emergence definition. Even worse, current studies usually assume prior knowledge of emergence or refer it to a closely related concept in a particular domain. Another remaining challenge is state-space explosion. A complex system tends to go through a large number of states, and some of these states manifest emergent properties.

In this thesis, we define emergence that as system states that result from interactions of components, but cannot be derived by summing the state of individual components. We

focus on addressing complex systems with mobile agents of different types. The reduction of search space is implemented in both our new perspective of emergence and the technique used for deriving emergent property states. The two major contributions of this thesis are: (1) a set-theoretic approach for determining a set of all system states from which emergent properties can be deduced, (2) a technique for reducing the state-space explosion problem.

5.1.1 Set-theoretic Approach to Determine Emergent Property States

We proposed a formal definition of emergence and a computational set-theoretic approach to determine it. Given a system, emergence is defined as a set of system states that arise from the interactions of the components of the system, but cannot be derived by summing the state of individual components together. The system is modeled as a multi-agent system in which agents play role of the components, and agents have different types and can move to enter and leave the system over time. As a result, the set of emergent property states is the difference between two sets: L^I_{whole} - the set of all system states reachable from the initial state due to interactions of agents, and L^P_{sum} - the set of possible system states resultant from mechanically superimposing state of individual agents.

To demonstrate our approach, we extended the Boids model to include two types of birds, ducks and geese. Our approach returns a set of emergent property states in which some states of this set exhibit a well-known flocking behavior. In addition, the experimental results showed that the size of the superimposition is large and increases exponentially with the number of birds. Another interesting observation is that more interactions of birds lead to more emergent property states. This emphasizes the key role of interaction in the presence of emergence.

In addition, considering a concurrent program as a concrete specification of the problem to be studied, we got rid of the limitation that abstraction makes emergence relative to the system model rather than to the system itself. A case study showed that given a multi-threaded program, our approach can be applied to derive the set of program states from which emergent properties, for example deadlock, can be deduced. By modeling the threads of the program as Finite State Automata (FSAs), L^I_{whole}, which is a set of all states reachable from the initial state, can be specified by composing the FSAs. Consequently, L_ξ is deduced from L^I_{whole} that consists of reachable states due to interactions of threads. These states potentially possess interesting properties due to concurrency, i.e. the interleaving execution among threads. In particular, we applied the proposed approach to multi-threaded programs to detect all deadlock states for different numbers of threads sharing two resources. Compared to model checking, our approach is more general in the sense that we can detect all types of deadlock rather than a particular type with respect to the given deadlock specification. Furthermore, our approach is superior to model checking in terms of the number of states to be visited.

5.1.2 Reduction of Search Space

The state space to be searched was gradually reduced in two steps: the definition of emergence and the elimination of L_{sum}. On the one hand, the multi-agent system abstracts aspects of the system of interest, and ignores details that are not of the designer's interest, thus constructing a smaller state space. On the other hand, relied on the observation that L_{sum} is the key source of the state-space explosion problem, but it does not contribute much to the derivation of L_ξ, especially when agents interact frequently, we proposed an alternative to determining emergent property states. The idea behind this method is to use degree of interaction of agents as an emergence criterion, thus eliminating the

unnecessary calculation of L_{sum}. In general, a property is emergent if agent interaction producing that property is strong. In other words, compared to the situation where agents behave independently, agents with interactions impose a remarkable effect on the system state. Intuitively, more interactions lead to more changes of the state of the system. By associating agent interaction with the state of the system, interaction degree is defined as the difference between system states. This idea enables a measurable and computational manner of studying emergence. The experimental results showed that the state space reduction technique enables simulating a large number of agents, 1,024 birds, compared to a simulation of 10 agents when not using this technique.

5.2 Future Directions

This thesis provides a first step towards advancing our understanding of emergence. Although emergence occurs in many domains at different levels, research into emergence is still in its infancy and poses several challenges that require further work to be done.

5.2.1 Consensus on Emergence

Complex systems research lacks of a consistent definition of emergent properties. It is important to recall that emergence is a set of system states, while emergent properties are system characteristics exhibited in these states. We proposed a new perspective of emergence, but a definition of emergent properties is still controversial. Different studies have adopted different views and methods dealing with emergent properties. Indeed, the study of emergent properties is an interdisciplinary field that ranges from surprise in contemporary philosophy to the whole-parts relationships with agent-based modeling and simulation in the field of computer science. Its occurrence in many disciplines in various

forms makes emergent properties difficult to be captured in a single, explicit, and formal definition. To enable a scientifically feasible study of emergent properties, we believe that emergent properties should contain the following characteristics:

- observer-independent, i.e. viewed as part of the system;
- originate from the interactions of components;
- only predictable through simulation techniques.

Scientific research on emergent properties should avoid dealing with subjective observation and should move towards a notion where they are considered intrinsic to the system, i.e. a feature of the system, rather than dependent on the observer. Emergent properties arise from interactions of the constituent components. The components are most likely loosely coupled and autonomous without any central control or global visibility. However, multiple components interact with others in a non-trivial way that should be analyzed using means of modeling and simulation.

Another issue with the definition of emergent properties is that it is somewhat difficult to distinguish among emergent properties, behavior, phenomena, rules, and structures. This difficulty is a proof of Holland's view: "Emergence will submit weakly to concise definition" [44]. Besides the frequently discussed definitions of emergent properties/behavior/phenomena in the literature, we can also have an idea on emergent rules and emergent structures. Intuitively, emergent rules (or algorithms) are simply behavior rules defined for individual components of the system whereby emergent properties inevitably appear. Similarly, emergent structure is the structure of an emergent property. Usually, emergent structures are the outcome of self-organization, which arises from feedback loops of influences between components.

5.2.2 State-space Explosion

State-space explosion is a challenging problem in the study of emergence. In the context of grammar-based formalization, this problem may come from two sources: L_{sum} and L_{whole}. As shown, L_{sum} increases exponentially with the number of components [84]. Eliminating invalid permutations of individual states based on the constraints between components does not guarantee a computationally practical size of L_{sum}. Moreover, determining these constraints from the system specification is non-trivial. L_{whole} contains all distinct system states reachable the initial state, for example obtained by simulation until the system state repeats. However, a complex system probably goes through a large number of distinct states, even infinite state space due to a large number of components or non-linear interactions of them. As a result, detecting emergence may suffer from expensive computational cost. Further work for reducing the state space, both L_{sum} and L_{whole}, is required. On the other hand, from the perspective of implementation, the way of representing system state also considerably affects the potential of our approach. As our approach is both compute and memory intensive, more efficient state representations, such as bit state hashing and state vector compression used in model checking, should be taken into account.

5.2.3 Emergence Reasoning

While there have been many efforts in detecting emergence, reasoning about its cause-and-effect is still in its infancy. The visible effects of emergence at the macro level can be far from explainable from the causal relationships between the micro level interactions. In other words, the state of a complex system can change drastically after a few transitions, and cannot be easily traced back to the initial system configuration and transition rules defining the behavior of individual components. This issue is even more challenging in

living systems and hybrid systems than in non-living systems. emphLiving systems are known to have evolutionary properties that may not be traced to component interactions. *Hybrid* systems consist of human (living) that interacts/inter-operates with non-living components. Such types of systems often present several amazing phenomena that seem to have no connection with the underlying components.

Based on the fact that emergence arises from component interaction, a formal method for reasoning of emergence should be bottom-up. This approach is opposite to the classical methods whereby systems are considered in a top-down manner and reduced to the constituent components. In the end, we should have some idea of what emergence exactly is, where it comes from, and how it develops, among others. In particular, it is important to know which state transitions or interactions significantly contribute to the occurrence of emergence.

5.2.4 Emergence Validation

The stochastic nature of emergence demands a new treatment of validating engineered systems. Unlike simulation validation, which confirms that a simulation meets expected behavior, emergent properties validation checks whether the properties are desirable or undesirable, and reflect some design errors or represent additional features in the discipline. Given a detected emergent property, it is important to determine whether the property is desirable or undesirable. The exploitation of desired emergent properties can benefit the development and performance of the system, making it more available, scalable, and robust. Undesired emergence or "misbehavior" [58], on the other hand, is a more important concern in the literature because it can significantly violate the performance of engineered systems. As a result, solutions that guarantee that there are no undesired properties at runtime are needed. If these cannot be achieved, it is important to detect the undesired

properties as soon as possible during the system execution to minimize their side effects.

Moreover, the system designer should validate that an emergent property is due to some design errors or is a new feature of the system. In the former case, the system designer needs to fix the errors with a new design. The latter case should be analyzed to understand its sources such that it can be further exploited to make the system more robust. Somewhere in between are emergent properties that the designer intends but cannot be able to specify in the design explicitly. Such an example is swarm behavior in birds, fish, bees, robotics, etc. in which entities attract and avoid their neighbors in a coherent way to achieve a common goal. Validating systems exhibiting this sort of emergent properties is a challenge.

Finally, new techniques are needed to extract *known* emergent properties, and to identify new (or *unknown*) emergent properties from the set of emergent property states. For example, if emergent properties are known, our formalism facilitates post-mortem emergence analysis to determine the causes of emergence.

Despite the importance of emergence validation, to the best of our knowledge, no comprehensive studies address the validation of emergence. This area of research is still active.

References

[1] F. Giunchiglia A. Cimatti, E. M. Clarke and M. Roveri. Nusmv: A new symbolic model checker. *International Journal on Software Tools for Technology Transfer*, March 2000.

[2] R. Abbott. Emergence explained: Abstractions getting epiphenomena to do real work. *Complexity*, 12(1):13–26, 2006.

[3] L. A. Adamic and B. A. Huberman. Power-law distribution of the World Wide Web. *Science*, 287:2115, 2000.

[4] R. Albert, H. Jeong, and A. L. Barabasi. Diameter of the World Wide Web. *Nature*, 401:130–131, 1999.

[5] R. C. Arkin. *Behavior-Based Robotics*. Cambridge, MA: MIT Press, 1998.

[6] M. A. Aziz-Alaoui and C. Bertelle. *From System Complexity to Emergent Properties*. Springer-Verlag: Berlin, 2009.

[7] N. A. Baas. Emergence, hierarchies, and hyperstructures. *Artificial life III*, pages 515–537, 1994.

[8] N. A. Baas and C. Emmeche. On emergence and explanation. *Intellectica*, pages 67–83, 1997.

[9] C. Baier and J. Katoen. *Principles of Model Checking (Representation and Mind Series)*. The MIT Press, 2008.

[10] Y. Bar-Yam. *Dynamics of Complex Systems*. Perseus Books, Cambridge, MA, USA, 1997.

[11] Y. Bar-Yam. A mathematical theory of strong emergence using multiscale variety. *Complexity*, 9(6):15–24, 2004.

[12] M. A. Bedau. Philosophical perspectives: Mind, causation and world. 1997.

[13] M. A. Bedau. Downward causation and the autonomy of weak emergence. *Principia 3*, 3:5–50, 2003.

[14] M. A. Bedau and P. Humphreys. *Emergence Contemporary Readings in Philosophy and Science*. MIT Press, 2008.

[15] E. Bonabeau. Agent-based modeling: Methods and techniques for simulating human systems. *Proc. of the National Academy of Sciences of the United States of America*, 99(10):7280–7287, 2002.

[16] E. Bonabeau and J. Dessalles. Detection and emergence. *Intellectica*, 25(2), 1997.

[17] J. A. Casazza and F. Delea. *Understanding Electric Power Systems: An Overview of the Technology and the Marketplace*. Wiley, New York, 2003.

[18] D. J. Chalmers. Strong and weak emergence. In *The Re-emergence of Emergence*. Oxford University Press, 2006.

[19] W. K. V. Chan. Interaction metric of emergent behaviors in agent-based simulation. In S. Jain, R. R. Creasey, J. Himmelspach, K. P. White, and M. Fu, editors, *Proc. of Winter Simulation Conference*, pages 357–36, Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc., December 2010.

[20] C. C. Chen. *Complex Event Types for Agent-based Simulation*. PhD thesis, University College London, 2009.

[21] C. C. Chen, S. B. Nagl, and C. D. Clack. Complexity and emergence in engineering systems. *Complex Systems in Knowledge-based Environments: Theory, Models and Applications*, 168:99–128, 2009.

[22] C. C. Chen, S. B. Nagl, and C. D. Clack. A formalism for multi-level emergent behaviours in designed component-based systems and agent-based simulations. In *From System Complexity to Emergent Properties*, pages 101–114. Springer-Verlag, 2009.

[23] S. H. Chen and C. H. Yeh. On the emergent properties of artficial stock markets: The efficient market hypothesis and the rational expectations hypothesis. *Journal of Economic Behavior and Organization*, 49:217239, 2002.

[24] L. Chi. Translating social capital to the online world: Insights from two experimental studies. *Journal of Organizational Computing and Electronic Commerce*, 19:214–236, 2009.

[25] James P. Crutchfield. Is anything ever new? considering emergence. In *IN*, pages 479–497. Addison-Wesley, 1994.

[26] P. Dallard, A. J. Fitzpatrick, A. Flint, S. Le Bourva, A. Low, R. M. Ridsdill Smith, and M. Wilford. The London Millennium Footbridge. *Structural Engineer*, 79(22):17–35, November 2001.

[27] V. Darley. Emergent phenomena and complexity. *Artificial Life IV*, pages 411–416, 1994.

[28] J. Deguet, L. Magnin, and Y. Demazeau. Elements about the emergence issue: A survey of emergence definitions. *ComPlexUs*, 3:24–31, 2006.

[29] G. B. Dyson. *Darwin Among the Machines: The Evolution of Global Intelligence*. Perseus Books Group, 1998.

[30] C. Eisner and D. Peled. Comparing symbolic and explicit model checking of a software system. In Dragan Bonaki and Stefan Leue, editors, *Model Checking Software*, volume 2318 of *Lecture Notes in Computer Science*, pages 230–239. Springer Berlin Heidelberg, 2002.

[31] S. Ferreira, M. Faezipour, and H. W. Corley. Defining and addressing the risk of undesirable emergent properties. In *Systems Conference (SysCon), 2013 IEEE International*, pages 836–840, 2013.

[32] D. Fisch, M. Janicke, B. Sick, and C. Muller-Schloer. Quantitative emergence - a refined approach based on divergence measures. In *Proc. of 4th IEEE International Conference on Self-adaptive and Self-organizing Systems*, pages 94–103, 2010.

[33] S. Floyd and V. Jacobson. The synchronization of periodic routing messages. *IEEE/ACM Transactions on Networking*, 2(2):122–136, April 1994.

[34] J. Fromm. Types and forms of emergence. http://arxiv.org/abs/nlin.AO/0506028, 2007.

[35] J. M. E. Gabbai, H. Yin, W. A. Wright, and N. M. Allinson. Self-organization, emergence and multi-agent systems. In *Proc. of International Conference on Neural Networks and Brain*, pages 1858–1863, 2005.

[36] M. Gardner. Mathematical games - the fantastic combinations of John Conway's new Solitaire game life. *Scientific American*, 223:120–123, 1970.

[37] V. D. Gligor. Security of emergent properties in ad-hoc networks (transcript of discussion). In *Bruce Christianson, Bruno Crispo, James A. Malcolm, and Michael Roe, editors, Security Protocols Workshop, volume 3957 of Lecture Notes in Computer Science*, page 256266. 2004.

[38] R. Gore and P. F. Reynolds. An exploration-based taxonomy for emergent behavior analysis in simulations. In *Proc. of Simulation Conference*, pages 1232–1240, Washington, DC, 2007.

[39] A. Gorenstein. Emergent behavior as a function of information. http://ftp.cs.rochester.edu/u/brown/242/assts/termprojs/emerge.pdf, 2009.

[40] P. Haglich, C. Rouff, and L. Pullum. Detecting emergence in social networks. In *Proc. of IEEE Second International Conference on Social Computing*, pages 693–696, Minneapolis, MN, USA, 2010.

[41] B. Heath, R. Hill, and F. Ciarallo. A survey of agent-based modeling practices (January 1998 to July 2008). *Journal of Artificial Societies and Social Simulation*, 12(4):9+, 2009.

[42] F. Heylighen. Complexity and self-organization. *Encyclopedia of Library and Information Sciences*, 2012.

[43] H. M. Hinton. Under-specification, composition and emergent properties. In *Proc. of Workshop on New Security Paradigms*, pages 83–93, New York, NY, USA, 1997.

[44] J. H. Holland. *Emergence: From Chaos To Order*. Addison Wesley, 1997.

[45] R. Holzer, H. De Meer, and C. Bettstetter. On autonomy and emergence in self-organizing systems. In *Proc. of 3rd International Workshop on Self-organizing Systems*, pages 157–169, 2008.

[46] G. J. Holzmann. The model checker spin. *IEEE Transactions on Software Engineering - Special Issue on Formal Methods in Software Practice*, 23(5):279–295, May 1997.

[47] P. Hovda. Quantifying weak emergence. *Journal of Minds and Machine*, 18(4):461–473, 2008.

[48] H. Hyotyniemi. Emergence and complex systems - towards a new science of industrial automation. In *Proc. of 4th International Conference on Intelligent Processing and Manufacturing of Materials*, May 2003.

[49] C. W. Johnson. What are emergent properties and how do they affect the engineering of complex systems? *Reliability Engineering and System Safety*, 12:1475–1481, 2006.

[50] J. Johnson. Multidimensional events in multilevel systems. In *The Dynamics of Complex Urban Systems: An Interdisciplinary Approach*, pages 311–334. Physica-Verlag, 2008.

[51] W. L. Koh and S. Zhou. Modeling and simulation of pedestrian behaviors in crowded places. *Journal ACM Transactions on Modeling and Computer Simulation*, 21(3):20:1–20:23, 2011.

[52] A. Kubik. Toward a formalization of emergence. *Artificial Life IX*, 9(1):41–65, 2003.

[53] J. Ladyman, J. Lambert, and K. Wiesner. What is a complex system? *European Journal of Philosophy of Science*, 2012.

[54] Z. Li, C. H. Sim, and M.Y.H. Low. A survey of emergent behaviour and impacts in agent-based systems. In *Proc. of IEEE International Conference on Industrial Economics*, pages 1295–1300, August 2006.

[55] E. D. Manley and T. Cheng. Understanding road congestion as an emergent property of traffic networks. In *Proc. of International Multi-Conference on Complexity, Informatics and Cybernetics*, volume 1. International Institute of Informatics and Systemics, 2010.

[56] K. McMillan. *Symbolic Model Checking*. Kluwer Academic Publishers, 1993.

[57] M. Mnif and C. Muller-Schloer. Quantitative emergence. In *Proc. of IEEE Mountain Workshop on Adaptive and Learning Systems*, pages 78–84, Logan, UT, USA, 2006.

[58] J. C. Mogul. Emergent (mis)behavior vs. complex software systems. In *Proc. of 1st ACM SIGOPS/EuroSys European Conference on Computer Systems*, pages 293–304, New York, NY, USA, 2006.

[59] T. Moncion, P. Amar, and G. Hutzler. Automatic characterization of emergent phenomena in complex systems. *Journal of Biological Physics and Chemistry*, 10:16–23, 2010.

[60] M. Muramatsu and T. Nagatani. Jamming transition in two-dimensional pedestrian traffic. *Physica A*, 275:281–291, 2000.

[61] K. Nagel and M. Schreckenberg. A cellular automaton model for freeway traffic. *Journal de Physique I*, 2(12):2221–2229, December 1992.

[62] G. NicolisandC. Nicolis. *Foundation of Complex Systems: Nonlinear Dynamics, Statistical Physics, Information and Prediction*. World Scientific, 2007.

[63] D. O. Norman and M. L. Kuras. Engineering complex systems. http://www.mitre.org/work/tech_papers/tech_papers_04/norman_engineering/norman_engineering.pdf, 2004.

[64] I. Norros, B. J. Prabhu, and H. Reittu. Flash crowd in a file sharing system based on random encounters. In *Proc. of Workshop on Interdisciplinary Systems Approach in Performance Evaluation and Design of Computer & Communications Systems*, New York, USA, 2006.

[65] T. O'Conner. Emergent properties. *American Philosophical Quarterly*, 31(2):91–104, April 1994.

[66] N. O'Neill. Google now indexes 620 million Facebook groups. http://www.allfacebook.com/google-now-indexes-620-million-facebook-groups-2010-02. accessed 10-October-2008.

[67] J. S. Osmundson, T. V. Huynh, and G. O. Langford. Emergent behavior in systems of systems. Technical report, Naval Postgraduate School, 2008.

[68] H. V. D. Parunak and R. S. VanderBok. Managing emergent behavior in distributed control systems. In *Proc. of Instrument Society of America Tech*, 1997.

[69] D. T. Pele and A. M. Tepus. Information entropy and efficient market hypothesis. In *Proc. of International Conference on Applied Economics*, 2011.

[70] L. L. Pullum and X. Cui. Techniques and issues in agent-based modeling validation. Technical report, Oak Ridge National Laboratory, 2012.

[71] K. K. Ramakrishnan and H. Yang. The Ethernet capture effect: Analysis and solution. In *Proc. of 19th Conference on Local Computer Networks*, pages 228–240, Minneapolis, MN, USA, October 1994.

[72] M. Randles, H. Zhu, and A. Taleb-Bendiab. A formal approach to the engineering of emergence and its recurrence. In *Proc. of 2nd International Workshop on Engineering Emergence in Decentralized Autonomic Systems*, 2007.

[73] P. Rendell. Turing universality of the game of life. In *Collision-Based Computing*, pages 513–539. Springer-Verlag, 2002.

[74] C. W. Reynolds. Flocks, herds and schools: A distributed behavioral model. In *Proc. of 14th Annual Conference on Computer Graphics and Interactive Techniques*, pages 25–34, New York, NY, USA, 1987.

[75] E. Ronald, M. Sipper, and M. Capcarrere. Design, observation, surprise! a test of emergence. *Artificial Life V*, 5(3):225–239, June 1999.

[76] C. Rouff, A. Vanderbilt, M. Hinchey, W. Truszkowski, and J. Rash. Properties of a formal method for prediction of emergent behaviors in swarm-based systems. In *Proc. of 2nd IEEE International Conference on Software Engineering and Formal Methods*, Beijing, China, September 2004.

[77] A. Ryan. Emergence is coupled to scope, not level. *Complexity - Complex Systems Engineering*, 13(2), November 2007.

[78] A. K. Seth. Measuring emergence via nonlinear Granger causality. *Artificial Life XI*, 324(1):545–552, 2008.

[79] C. E. Shannon. A mathematical theory of communication. *Bell System Technical Journal*, 27(3):379–423, 2000.

[80] S. Stepney, F. A. C. Polack, and H. R.Turner. Engineering emergence. In *Proc. of 11th IEEE International Conference on Engineering of Complex Computer Systems*, 2006.

[81] I. Stewart. The ultimate in anty-particles. *Scientific American*, pages 104–107, 1994.

[82] C. Szabo and Y. M. Teo. An objective-based approach for semantic validation of emergence in component-based simulation models. In *Proc. of 26th ACM/IEEE/SCS Workshop on Principles of Advanced and Distributed Simulation*, pages 155–162, ZhangJiaJie, China, Jul 15-19 2012.

[83] C. Szabo and Y. M. Teo. Post-mortem analysis of emergent behavior in complex simulation models. In *Proc. of ACM SIGSIM Conference on Principles of Advanced Discrete Simulation*, pages 241–251, Montreal, Canada, May 19-22 2013.

[84] Y. M. Teo, B. L. Luong, and C. Szabo. Formalization of emergence in multi-agent systems. In *ACM SIGSIM Conference on Principles of Advanced Discrete Simulation*, pages 231–240, 2013.

[85] V. Trianni and S. Nolfi. Engineering the evolution of self-organising behaviours in swarm robotics: A case study. *Artificial Life*, 17(3):183–202, 2011.

[86] L. Vinerbi, A. Bondavalli, and P. Lollini. Emergence: a new source of failures in complex systems. In *Proc. of 3rd International Conference on Dependability*, pages 133–138, Venice, Italy, 2010.

[87] C. Pasareanu W. Visser and S. Khurshid. Test input generation with java pathfinder. In *Proc. of International Symposium on Software Testing and Analysis*, July 2004.

[88] T. De Wolf, G. Samaey, T. Holvoet, and D. Roose. De-centralized autonomic computing: Analysing self-organizing emergent behavior using advanced numerical methods. In *Proc. of 2nd International Conference on Autonomic Computing*, pages 52–63, 2005.

[89] S. Wolfram. Cellular automata as models of complexity. *Nature*, 311:419–424, 1984.

[90] C. Xiang, R. Kennedy, and S. Cabaniss G. Madey. Verification and validation of agent-based scientific simulation models. *Proc. of Agent-Directed Simulation Symposium. The Society for Modeling and Simulation International*, pages 47–55.