Algebraic approaches to artificial chemistries

Penelope Selina Margaret Faulkner Rainford

PhD

University of York

Chemistry

December 2018

Abstract

We have developed a new systematic framework, MetaChemisty for the description of artificial chemistries (AChems). It encompasses existing systems. It has the flexibility and complexity to allow for new features and new systems. A joint description language will allow comparisons to be drawn between systems. This will allow us to write metrics and benchmarks for artificial chemistries. It also enables us to combine existing systems in different ways to give a wealth of more complex and varied systems. We will be able to build novel chemistries quicker through reuse of code and features between chemistries allowing new chemistries to start from a more complex base line.

We have also developed an algebraic artificial chemistry, Jordan Algebra Artificial Chemistry (JA AChem). This chemistry is based on existing algebra which is leverage to ensure features such as isomers and isotopes are possible in our system. The existence of isotopes leads naturally to the existence of elements for this chemistry. It is a chemistry with both constructive and destructive reactions making it a good candidate for further study as an open-ended system.

We analyze the effect of changing probabilistic processes in JA AChem by modifying the probability spawning functions that control them. We also look at the algebraic properties of these probability spawning functions. We have described Swarm Chemistry, Sayama (2009), in the MetaChem showing it is at least more expressive than the previous framework for artificial chemistries, Dittrich et al. (2001).

We use the framework to combine two artificial chemistries using a simple environment link structure to produce eight new modular AChems with a modular approach. This link structure requires minimal addition to existing code for artificial chemistry systems and no modification to most modules.

Contents

List of Tables

List of Figures

“It’s a dangerous business, Frodo, going out of your door,” he used to say. ”You step into the Road, and if you don’t keep your feet, there is no knowing where you might be swept off to.”

- *J. R. R. Tolkien*

Acknowledgments

I would like to briefly thank both Susan Stepney and Angelika Sebald for being fantastic supervisors and following my crazy idea. I would also like to thank Leo Caves for his support as my advisor during my studentship. Finally a huge amount of credit goes to James Rainford for keeping me sane and fed all the way through.

Declaration

This thesis has not previously been accepted in substance for any degree and is not being concurrently submitted in candidature for any degree other than Doctor of Philosophy of the University of York. This thesis is the result of my own investigations, except where otherwise stated. Other sources are acknowledged by explicit references. The Jordan Algebra Artificial Chemistry chapter contains content from Faulkner et al. (2016). The Tuning Artificial Chemistries with Probability Spawning Functions chapter contains content from Faulkner et al. (2017). Finally the Nested Chemistry chapter contains content from Rainford et al. (2018).

I hereby give consent for my thesis, if accepted, to be made available for photocopying and for inter-library loan, and for the title and summary to be made available to outside organisations.

Chapter 1

Introduction

1.1 Artificial life and transitions

Artificial Life research looks into systems that model, simulate and are inspired by life. This spans from work in labs attempting to create artificial cells and modify existing life for use in computation to agent based models using bio-inspired algorithms to perform computational tasks. Here we focus on software based systems designed to be inspired by life and help to illuminate life.

There are many things about life and the complex interactions between living organisms that are hard for us to determine through observation. One of the key sources of this problem is often the "common ancestor". All life on earth started in the same place. All animals have a common ancestor. This means if we want to know anything about how "life" behaves or comes about then from observation of the world around us we can only draw conclusions for "carbon based life on earth". Considering the size of the universe this might be a very limiting restriction on "life". Can life develop without gravity? We don't really know if gravity is important for the development of complex life. So far the best we can do on this question would be to see if existing life can survive in zero gravity. This does not answer the question though. Humans can survive in zero gravity, but obviously the bipedal human would not evolve in zero gravity.

This is where artificial life with life forms entirely *in silico* can help us. We could simulate a simple environment with life at the point where we think gravity starts to become important (eg flat worms) and compare evolutionary potential of models with and without gravity. The key part of this technique is that we can control different aspects of the environment and speed up the timescale. Putting flat worms in space and waiting for them to evolve would take a very long time.

1.1.1 Different scale perspectives on artificial life

As well as the difference in time scales we also have very different scales of interest in the development of life. Artificial life in social networks, how we learn, the development of intelligence and language: these are all involved with or inspired by very high level lifeforms. But there is also work at the very low levels, looking at the movement of single cell organisms, evolution in terms of mutations in genomes, and abiogenesis: the emergence of life from non-living matter.

This forms a cohesive whole as the field strives in different directions trying to connect the movement of the cell to the behaviour of the organism and the evolution of the brain to the development of different types of social learning. Changes as we move, for example, from single cell to multi-cell organisms are called *major transitions* and are the target of much of artificial chemistry research. At the lowest level of artificial life we have the ultimate bottom-up approach to answering these questions through artificial chemistries.

1.2 Artificial Chemistry

In the early 2000s Dittrich et al. (2001) declared that an artificial chemistry could be described by the triplet (S, R, A). In this form we have three parts to an artificial chemistry: a set of particles, rules for reactions, and the algorithm. Expanded we have the particles of our chemistry described as a set. The rules of our reactions are the second aspect meaning we must have a discrete rule set describing all reactions possible in the system. Finally all other aspects of our system as clumped in A as part of the algorithm. This is a large number of things including global variables, timing, logging and rule application.

Another interpretation of an artificial chemistry may be more vague but also more inclusive of all artificial chemistry systems. An artificial chemistry is a system with minimal components designed to use or explore the higher order emergent properties of their interactions. This allows for chemistries with purely kinetic interactions and others that do not distinguish between R and A parts of the previous definition. It also acknowledges that some systems are distinguished as different due to the the differences in A rather than S or R. This definition can also be seen as the broadest possible definition encompassing any system in the universe, but then again this may be bias caused by living in a universe so heavily populated by at least one chemistry.

1.2.1 Different artificial chemistries

Artificial Chemistry systems have many differences. Almost all systems have unique and different particle sets, S and rules R. There is more overlap in other areas to do with timing and motion in the systems. To talk about the differences in artificial chemistries in these terms we could talk about implicit rules vs explicit rules and the size of particle sets.

However a better overview of chemistries is to discuss the difference in intent and features of chemistries.

Features, like spatiality and movement of particles, appear in some chemistries but not others. In particular some particles move independently of the reactions in the system or based on a “flow” pushing all particles in a direction. In some cases obstacles or particular reactions might also effect this movement. We also have some systems with resources and kinetics, and others that do not even have conservation of the number of particles in the system. We have systems with discrete vs continuous time. A key difference between systems can be the existence or not of destruction in the system: do structures once built ever break? These kinds of features can all exist somewhat separate to the nature of the particles and rules of the interactions in the system, but have a big effect on what a system can do.

What a system can do is to different extents, in artificial chemistries, defined by the designer. Some artificial chemistries are designed with the concept of producing a system capable of building a cell-like structure, or a membrane. These systems can then do these things. Other systems are designed to be capable of evolution and individual self-replication, and are then examined for larger scale organisation and the maintenance of that broader object.

1.2.2 Issues with (S, R, A) description

In the (S, R, A) description, almost all the features described above are in A. This one variable contains huge amounts of very important parts of an AChem system but is bundled as “everything else”. Also, aspects of a system that may indicate intent or over-design towards a goal can be lost in A. If a system has to use an extra reaction to make membranes possible, where do we find this in (S, R, A)? If a system is probabilistic based on a property of the environment, is that in R for the reaction or A the algorithm? It is impossible to rigorously define differences or similarities between systems with such a crude language of only three words.

The (S, R, A) description was a good tool for its time, when artificial chemistries were often still small toy systems or even thought experiments. It was even sufficient for the many “proof of concept” chemistries that demonstrated that an artificial system could produce cell-like objects capable of self-maintenance and self-replication. However, as artificial chemistries move into new realms of being used for computation and as the basis for open-ended evolution systems, we need to be able to analyse them more rigorously and make comparisons. We need to develop a new, more complicated but also more descriptive, framework for artificial chemistries.

1.3 Rigorous Design

1.3.1 Challenges of designing a framework

Designing a framework for such a diverse a set of systems is a distinct challenge. It cannot be done just by attempting to break up A into more distinct and descriptive objects. This would possibly be able to cover existing artificial chemistries, but could limit the creation of new ones as new features may not be included. To avoid that issue we would still need an A-like object capable of being "everything else".

We instead want a framework that considers the dynamic nature of these systems. They are not just a set of features and particles and rules. They are how these are all executed. We therefore need a framework that can follow the execution of the system as well as the static description of it. This needs to be able to encompass existing artificial chemistries and not limit the design of new ones. With such a broad definition of what an artificial chemistry is, this is a very difficult challenge.

1.3.2 Why algebraic structures?

We want to describe a large set of systems and not limit future systems. We also want a framework that lends itself to comparison between systems. For this we want systems to be able to have identical components within their very different settings, or at least have equivalent components. This means we need a concept of equivalence. We also need a rigorous definition that will encourage identical description of identical components. This level of restriction over large possibilities is done well in most cases by algebraic structures. There are also a large number of options available to us for this, as the world of algebraic structures has been well explored. This means that the correct structure could naturally lend us pre-existing morphism definitions, for instance, that could be used for comparison between systems.

1.3.3 Using but not being constrained by algebra

The one drawback to an algebraic approach could be over constraint due to the difficulty of describing a vague idea in a specific manner. Algebra can provide considerable structure to a framework, but we must be careful not to allow this to lead to us feeling required to describe every possibility in minute detail in algebra, or to provide constraints on everything such as the kinds of interactions possible for our systems. This is not required for an algebraic description it is only a risk thereof. We can easily avoid this through awareness of the issue. Much like artificial life is inspired by but not constrained by nature, an artificial chemistry framework can be described by but not constrained by algebra.

1.4 Structure of thesis

We will start by reviewing current literature and artificial chemistries with particular consideration for what we would like from an artificial chemistry framework. We will outline the goals of our work. We will then describe and formally define a new framework for artificial chemistry (MetaChem). We will describe an algebraically inspired artificial chemistry in MetaChem, Jordan Algebra Artificial Chemistry (JA AChem) and its elements and reactions. We will investigate the effect of variation in the probabilistic nature of JA AChem using probability spawning functions. We will review an existing artificial chemistry (Swarm Chemistry) described in our MetaChem framework. We will use the properties of our framework to combine both the described artificial chemistries in a hierarchical system and analyze the effect of this on the components of both systems. We will discuss how this allows us to build multilevel artificial chemistries towards transitions.

Chapter 2

Literature Review

2.1 Artificial Life

Artificial life is a broad field dealing with many living and life-like systems. There are wet lab systems building artificial protocells (Walde, 2010) and studying evolution in *E.coli* (Elena and Lenski, 2003). There are hardware systems such as robotic fish shoals (Liu and Hu, 2010) and self-maintaining robot swarms (Turgut et al., 2008). There are software systems at many levels from social learning models (Acerbi and Nolfi, 2007), evolution models (Ofria and Wilke, 2004; Andrews and Stepney, 2014; Knibbe et al., 2007) and artificial chemistries (Hutton, 2002; Ono and Ikegami, 2001; Liu, 2018; Clark et al., 2017; Sayama, 2009; Dittrich et al., 2001). These all exist for many different reasons, some practical for computation and problem solving like evolution systems (Nguyen et al., 2012; Branke et al., 2000; Yamamoto, 2010), others for hypothesis testing to try to prove theories about origins of life or the development of society and learning (Knibbe et al., 2007; Elena and Lenski, 2003; Carter et al., 2018). Some systems are just the first steps in exploring the broader space of potential life.

2.1.1 Origins of Life

The origins of life debate within chemical and biological sciences was long dominated by the information or replication first vs metabolism first argument (Pross, 2004; Faulconbridge et al., 2009; Anet, 2004). More recently it has become commonly accepted that this differentiation is incorrect. Information has no purpose without a metabolism to maintain and use it, otherwise we just have a photocopy of random strings, this is not life. Similarly metabolism without information is just a processing system, a plumbing system that can process resources and maintain itself to some extent, but this is not on its own “alive”. Most conceptualizations of ”life” as needing to pass itself on-wards to a future system or a copy of itself. For this a concept of self is needed normally associated with the information in a life form.

With this change in focus many models have been built to try to emulate early life and its formation from non-life, to test theory in a more controlled and less random environment, (Higgs and Lehman, 2015; Ruiz-Mirazo et al., 2017). These models are artificial life systems, and many such systems built with ALife in mind are trying to achieve the same thing, although they often step beyond simulation of possible earth-like conditions for the beginning of life into exolife (possible life on other planets) (Szostak, 2017; Michalski et al., 2018) and strict ALife, completely artificial systems,(Yampolskiy, 2016; Froese et al., 2014; Nitash et al., 2017). Within these systems we begin to talk in a new way about top-down and bottom-up approaches, which we hope will meet in the middle (Hickinbotham et al., 2016; Allen et al., 2005).

Artificial systems may seem at first to have very little to do with the origins of life on earth. However if we can find the conceptual limitations of the development of a system with both metabolism and information this may help focus more practical experimenters onto more promising routes for how life started, in particular protocells (Ackley, 2018). It also presents a timescale advantage: life began on this planet almost certainly over a very long time period as systems slowly changed and moved to cross the line into life. In an artificial system we can guide change and speed up the generational clock such that as soon as a state has been evaluated we can change and move to the next possible state and not have to wait for "natural" processes to complete.

This also applies for systems looking at the development of life which take the existence of "living" material as a given, and attempt to build to more complicated organisms and behaviours. Particularly of interest to ALife research has been the cause and identification of transitions in systems, often called major transitions (Bedau et al., 2000). These are points where it becomes possible to differentiate "higher level entities from cooperative organizations of lower-level entities" (Stewart, 1997). It is these processes that allow us to move from chemistry to biology, from particles up to cells and then on to multi-celled organisms. This question at the level of chemistry has started to be addressed in practice only recently (Sayama, 2018a), despite being an important artificial life question for almost 20 years. This is mostly due to scale restriction on artificial chemistries up to now, and the use of symbolic and "stick and ball" chemistries that make broader behaviours harder to view. This is discussed further in section 2.2.

2.1.2 Open ended evolution

Here we define open ended evolution as "processes that do not stop and have no specific objective" taken from (Banzhaf et al., 2016). Open ended evolution is an area of artificial life which reaches for systems that, like life on earth, display an ability for unending novelty at all levels. Recently these novelties have been categorised at three levels: variation, innovation and emergence (Banzhaf et al., 2016). This is a propety that has proved challenging just to define, let alone achieve (Maley, 1999; Adams et al., 2017). It has been more possible to say what some of the necessary conditions are, though we are still not sure on what would be sufficient(Soros, 2018). Despite their elusive nature, such systems appeal strongly to the artificial life community as a chance at a true "origin of artificial life" story. If it were possible to build a truly open-ended system capable of evolution that we could run on a large enough scale, then we would have the chance to see an interpretation of the first development of an artificial life system from infancy to unplanned adulthood of complex artificial life forms. The first steps in identifying the major transitions of an ALife system are the first steps towards identifying open-ended evolution and have been taken on in artificial chemistries (Sayama, 2018a,c).

2.2 Artificial Chemistries

Artificial Chemistries have for the last two decades hovered on the periphery of artificial life. It is only now with the increase in computational power that we are starting to see artificial chemistry systems that can do more than just be proofs of concept systems or even toy systems.

Up until now artificial chemistry systems have been best described in the (S, R, A) framework given in Dittrich et al. (2001). This framework does a good job of classifying those early artificial chemistry systems. It does this through distinctions based on the S and R aspects of the systems.

We have explicit particle sets, S, such that every particle is separately defined, or implicit particle sets, where the bounds of the particle set is defined.

There is similar distinction between the rule sets, R. Some rule sets are lists that state the exact particles in each reaction and in the result. Others have general rules and describe the types of particles and based on the starting particle what the result will be.

The A aspect of the framework is given a few categories: stocastic molecular collisions, continuous differential or discrete difference equations, metadynamics, mixed approaches, and symbolic analysis of the equations. These all describe reaction dynamics in well-mixed tank models. They do not even begin to describe systems with spatial dynamics or can only capture an approximation of automata systems which actually replicate their own particles using clock cycles.

This framework has been applied to artificial chemistries since its creation, but has not kept up with the increasing complexity of these systems. Here we discuss artificial chemistries with a different categorisation based on the purpose of the chemistries.

2.2.1 Subsymbolic artificial chemistries

Subsymbolic chemistries are one of the types of chemistry that has emerged in recent years. These are systems whose particles have their own properties which are used to define their interactions by more than just a label, (Faulkner et al., 2018). This category of artificial chemistries in terms of the (S, R, A) format have particles with internal structure in S, implicitly defined rules in R based on those internal structures, and no particular requirements on A.

This category of AChems includes intentionally subsymbolic chemistries such as RBN-World (Faulconbridge et al., 2009, 2010; Faulconbridge, 2011) and its variants such as Spiky RBN-World (Krastev et al., 2016, 2017). This category also includes automata chemistries based on instructions which form programs, whose interactions are defined by the operation of those programs. Examples include Stringmol (Hickinbotham et al., 2016) and Avida (Ofria and Wilke, 2004).

There are other artificial chemistries that have internal structure to their particles and those with implicit reaction rules such as Swarm Chemistry (Sayama, 2009) where particles have their own parameters that dictate their reactions to other particles. This could also be considered as a subsymbolic chemistry, but without physical linking this is very different from the other chemistries mentioned.

2.2.2 AChems for behaviours

The previous grouping of AChems were described by their low level properties. Here we wish to move away from that and look more at discussing AChems in terms of intent and the utility for achieving that intent of a structure like (S, R, A).

Many AChems exist that are designed with a clear aim for a system capable of a specific behaviour. Common choices are self-maintenance of cell-like organisations and self-replication of particles or cell-like organisations. These vary between systems in which the behaviour has been found, and some where the system has been evolved or manipulated into producing the behaviour. The second of these is useful as a proof of concept that a system can exist with that behaviour within an artificial chemistry. However such systems often have very little broader applicability. It is harder to expand their capabilities as they are often lack destructive reactions and are not truly open ended systems.

However these proofs of concepts have shown the very broad behaviours artificial chemistries are capable of. For example Squirm3 is a basis for self-replicating molecules (Hutton, 2002, 2003), enzymes (Hutton, 2007) and parasites (Lucht, 2012). This has been simulated also in Soula (2016).

2.2.3 AChems for modelling

Artificial Chemistries are not limited to just abstract systems for origin of life or evolution. They also feature as simulations of natural chemistry. In some cases we have artificial life systems which simulate chemistry but we also have the fact that simulations of chemistry are at times artificial chemistries.

A particular grouping of these are stochastic simulation chemistry (Gillespie, 1977). A more recent addition in this area (Andrews and Bray, 2004) is very much a system we could class as an artificial chemistry that just happens to be attempting to closely mimic natural chemistry. These systems are important to remember when we look at designing our own framework, as any artificial chemistry framework should also encompass natural chemistry and its simulation. On an abstract and conceptual level these are the same class of system so should be included.

2.2.4 AChems for computation

A small number of Artificial chemistries have been designed to perform computation. Included in this set is the artificial chemical reaction optimisation algorithm for global optimisation (Alatas, 2011). This algorithm meets all the requirements to be an artificial chemistry in its own right even if as a system it is designed to converge on a solution rather than display new behaviours.

We likewise can mention systems like AVIDA which have proved capable of computation; it evolves its organism to compute logic functions to receive resources (Ofria and Wilke, 2004; Goldsby et al., 2012). These systems are generally inefficient at computation compared to modern computer processors but similar to all evolution based computation it has greater power on some problems as it is better at exploiting unexpected solutions.

2.2.5 AChems for Open Ended Evolution

There are many systems searching for open ended evolution. Swarm Chemistry looks for spatially organised entities as higher-order entities for transitions (Sayama, 2018c). We also have simple systems such as “Hash Chemistry” (Sayama, 2018a), which shows that we can create an infinite possibility space by leveraging higher-order entities for a “cardinality leap”.

Even the more designed artificial chemistries are being used to explore the mechanisms behind open ended evolution (Young and Neshatian, 2013), though with acknowledgement that this is a property more likely to be discovered than designed.

The automata chemistries (Hickinbotham et al., 2010; Ofria and Wilke, 2004) also look for open ended evolution. We can classify the features of these systems into physical and biological. The lower level of the system becomes the physics and the higher level of the system the biology (Hickinbotham et al., 2016).

2.2.6 Problem with (S,R,A) format for comparison of systems

Over time systems have become more complex in all cases. Systems for modelling have to take into account the importance of separation and barriers in life systems. For computation we have the need for multiple layers of “physics” in the form of assigning entities their own simulated processor or clock cycles. For open-ended evolution we need the concept of higher order entities. All of these things are lost in the (S, R, A) definition.

We need greater complexity to capture greater complexity.

2.3 Applications of Algebraic structures to real world and computational problems

Algebra is often seen as pure mathematics, a subject that furthers only mathematics and has no real direct application to the rest of the world. However algebra and algebraic structures have many applications.

We can find algebra in fluid dynamics (Vreman, 2004), and quantum physics (Haag and Kastler, 1964). It is in virology (Keef and Twarock, 2009) and genetics (Shannon, 1940). We find various algebras in computation including graph theory (Ehrig et al., 2010), and category theory (Barr and Wells, 1990). Fields that have advanced in recent years are pushed by their applications.

In this work we are particularly interested in two kinds of algebra: non-associative algebras and semi-rings. Non-associative algebra has been associated with genetics for a long time (Etherington, 1941), and many other applications exist (Sabinin et al., 2006). These algebras have an innate ordering to them, or time installed, by not being able to rearrange their operation. This makes them perfect for describing the results of irreversible operations.

Semi-rings are also widely used in varied applications. These include constraint satisfaction and optimisation (Bistarelli et al., 1997), and many fields within mathematics and informational sciences (Głazek, 2002). These applications then apply onwards to many other fields and subjects.

Chapter 3

Research Hypothesis

3.1 Expressing Artificial Chemistries in a single logical framework

We will develop a framework which will allow us to describe artificial chemistries. This will make it quicker and easier to talk about these systems when we can do so directly as compare like for like. This give us a basis for categorisation and joint analysis. This could allow us to develop metrics and benchmarks for artificial chemistry systems that are not possible using only the existing (S, R, A) framework. This system will be logically consistent so that while versatile to expressing artificial chemistries it will also provide the basis for rigorous analysis.

3.2 Rational Design

3.2.1 Framework

Our framework will be described as an algebraic structure. It will also be modularised so we can recombine our systems and compare them. This will be done in a bottom up approach in which we will incorporate both information and algorithm into a single description of the system. We will also isolate the particles of our system from our environmental variables. This will allow for focuses attention on our particles and for general analysis algorithms for particle networks.

3.2.2 Artificial Chemistry

In order to test our framework we will also need an artificial chemistry to test it on. We will keep with our philosophy of rational design and will build the chemistry based on a mathematically derived set and operation. We will describe this system using our framework.

3.3 Combining existing artificial chemistries to increase complexity

We will use the modularised framework to combine both our algebraic artificial chemistry and a pre-existing artificial chemistry. This will be chosen to be as different as possible from our own system to highlight the diverse capabilities of the framework to describe artificial chemistries. The combination chemistry will be designed to be have a greater richness than either individual chemistry. Whether this means a greater complexity to the individual particles, interactions or additional features such as kinetics or greater degrees of freedom in their behaviour.

Chapter 4

MetaChemisty

We introduce MetaChemistry, a representation language for Artificial Chemistries. We consider the motivation for modularisation and standardisation in representation of artificial chemistries. We describe a mathematical formalism for MetaChemistry as a static graph based system. We introduce StringCatChem (String Concatenation Chemistry) a simple artificial chemistry which we use to illustrate the concepts of different levels of description possible with MetaChemistry. We also use StringCatChem to illustrate our mathematical formalism.

In artificial chemistry we struggle to talk about our systems in terms of the whole field. It is hard to make comparisons between systems. This is because many of our systems have similar goals but very different components and algorithms. This makes building even a basic classification system such as in Dittrich et al. (2001) very challenging.

There are concepts that we build on in artificial chemistries. We work on the basis of small components interacting to generate our systems. We are in general interested in the emergent properties and behaviours of these systems. To differentiate an artificial chemistry from an individual based model we add requirements for a level of simplicity and tractability in our particles and their interactions. The intention is that these systems work over large collections of individuals over long time periods, though most are currently limited by computational capability.

From this we can identify many common elements of artificial chemistry systems. We can use these as the basis for a bottom up approach to modularisation of artificial chemistry systems in general. Small, simple individuals and their interactions are our primary focus. We call these individuals *particles.* These are a constant through out artificial chemistry systems. Systems also have other variables, properties and values. Much like in real chemistry we separate the chemistry of the beaker from our consideration of its contents. We separate these other values and properties into the *environment.* We also have multiple *containers* in our systems that allow us to isolate particles and move them. This splits the dynamic parts

Table 4.1: Common parts of Artificial Chemistry Systems

	Primary Focus	Auxiliaries
Objects	Particles	Variables
Containers	Tanks	Environment

of our system as shown in Table 4.1.

This handles the "things" in our systems. However there are far more commonalities in the algorithms than there might seem. Control flows related to time and generations occur in most systems. Some systems perform an update across all objects in the system at once. Others continuously update objects at random. If we can identify the modularised control that produces these timing systems designers could switch between them. These sorts of structural combinations would then allow designers to focus on the new AChem specific features of their design. They could do this by using existing elements to implement less unique aspects of their systems.

By dividing our "things" we can then define our control flow in relation to those divisions. We have functions that modify our particles similar to reactions and interactions in chemistry. We have functions that record observations of our system. We can modify our environment such as by changing the temperature of the system. We move particles around our system. Finally we make decisions about which of these things we should do next.

4.1 Modularisation: Components of an Artificial Chemistry System

We conceptualise these idea into the structure provided in Figure 4.1, which we use to build a graph-based formalism. We have the overarching concepts of the System, made up of the elements formalised as graph nodes (Containers, Control), and as graph edges (Information Flow, Control Flow).

Control items become static nodes in the graph, defined at the start and remaining unchanged. These control nodes are connected by Control Flow edges, which together give the system's algorithm.

Containers are also static nodes in the graph (that is, the location and connectivity of these nodes is unchanging), but they have a mapping to the dynamic "things" in our system: particles and environmental values. Information flow edges allows control to influence the connected containers' states. Information flows in either direction: read or pulled from containers' state to control, and pushed from control to update containers.

These general concepts are captured in the set of graph nodes in Table 4.2 and edges in Table 4.3. We can then create graphs that provided a view of our systems once modularised. This can be done as an overview and then we can expand nodes to give us greater detail.

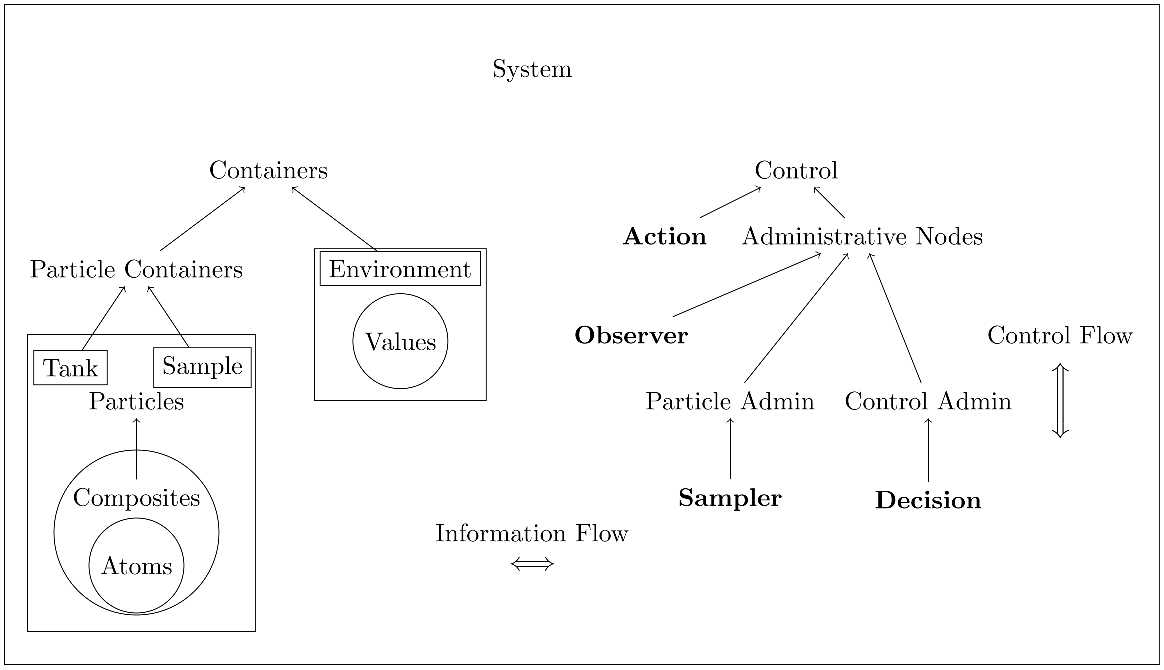


Figure 4.1: Conceptual structure of modularisation of Artificial Chemistries

4.1.1 Particles

The most fundamental parts of our systems are the *particles*. These and their emergent properties are the focus of our studies. These can be split at times into two subsets: *atoms* and *composites*. *Atoms* are the most basic particles they can not be divided or broken down into smaller parts. Any internal structure of the atoms is unbreakable.

Example: Atoms can take many forms: characters in a string chemistry, instructions in an automata chemistry or symbols that do not feed into a one to many rule associated with them in a symbolic chemistry.

Composite particles are made up of combinations of atoms. In symbolic systems the atoms making up a composite particle may be hidden or unknown.

Example: Composites would be strings in string chemistries, programs in automata chemistries or symbols that result from many to one rules in symbolic chemistries.

Some systems may only have atoms and due to lack of physical bonding rules may not seem to form composites. Other systems may be symbolic and assume that all particles are complex and the symbols represent composites.

Element	Description
Containers	
$\boxed{\textbf{T}}$	**Tank** containing particles.
$\boxed{\textbf{S}}$	**Sample** containing an editable subset of particles.
$\boxed{\textbf{V}}$	**Environment** containing non-particle variables and information in the system.
Control	
◇ s	Information administration node moves particles between containers by **sampling** them
◇ o	**Observes** particles and produces summary statistics (saved in the environment), also an information administration node.
△ d	Control administration node makes weighted **decisions** on control flow based on the state of the particles and the environment
ⓐ	Performs **actions** on particles based on state of particles and environment

Table 4.2: Legend of graph nodes in MetaChem

Element	Description
Information Flow	
□- - - -	Reads information from container into control node.
□- - -▶	Pull moves information out of a container into the control nodes local storage during it's operation.
□◀- - -	Writes information from control node into a container.
Control Flow	
——▶	Arrow between **Sampling**, **observer** and **action** nodes to indicate control flow in system.

Table 4.3: Legend of edges used in MetaChem

4.1.2 Container Nodes

Container nodes split in to two sets: Particle nodes and environment nodes. Particle nodes are mappings which take the node and the state of the system and returns the set of particles in that container at that state. When the system is in a particular state the set of mappings of all the containers forms a partitioning of all the particles in the system. There are two types of particle nodes: samples and tanks. Tanks are protected containers. Particles in tanks can be moved in and out but not changed. This is so that any changes must be made over samples so we have to decide what we will be changing before we can effect it.

Example: A beaker being used for an experiment and a pipet or petri dish.

Environment nodes work similarly to particles nodes but contain the other objects and information in our system. The system can have multiple environments to make reference to the things in the environment easier. For instance one might want to store a time record separately to summary statistics or log information. These are all still accessed via a mapping from the node and state of the system to the dynamic information and objects.

Example: Temperature readings, Bunsen burner, stirrer or observation log.

4.1.3 Action Nodes

Actions are where we actually modify particles through movement, linking, decomposition or any other change. These can only modify particles in a *sample*. This means we need to designate those particles we are changing before change occurs. This protects the *tanks*.

Example: Concatenate string, form chemical bond or split program

4.1.4 Admin Nodes

We start with our Information Admin nodes: *sampler* and *observation*. *Sampler* nodes move particles between particle containers (*tanks* and *samples*).

Example: Extracting a sample for testing with a pipet or choosing a neighbour to combine with the current particle.

Observer nodes of the particles and/or environment do not change any internal properties of particles or move them between containers. They can only see particles and can then modify the *environment*.

Example: Taking notes in a log book, updating time in a discrete time system or updating the number of particles in the system.

Our Control Admin nodes, *Decision* nodes, are used to change control flow. This is the only place control flow can branch based on the state of the whole system or a subset of it.

Example: Triggering an event, continuing to the next phase or looping over a process, completing a time step or deciding to take a beaker off the heat.

4.1.5 Edges

The nodes of our graphs are connected by two sorts of relationship, Table 4.3. The first relationship marks movement of information between nodes. These relationships are always between container nodes and control nodes. Container nodes can not directly transfer information, the same is true of control nodes.

Example: An action node might read information in from a sample to give it two particles to combine and then write to the results of their linking to the sample. This would be a double ended information edge, both a push and a pull edge.

Our second relationship type marks the movement of control. These edges are between control nodes and indicate what order we use the nodes in. For most control nodes there can be only one outgoing control edge. The exception to this rule is decisions whose purpose is to provide a branch point in control flow.

Example: After an action node forms a physical link between particles in the sample we move to a sampler node which returns the content of the sample to the tank. There is a control edge between the action and the sampler describe this ordering. After returning this we have a control edge leading to a decision which checks if the system is finished with linking and has two control edges coming out of it. If the system is finished we loop to continue linking, if not we continue on to the next process in the system.

With this modularisation we can break down an AChem into subcomponents. The level at which we define these subcomponents is the *descriptive level* of our graph.

4.1.6 Graph as an Executable Algorithm

We have so far discussed separating out the parts of an artificial chemistry into nodes and briefly described forming this into a graph using edges between these nodes. Once we have this graph it is important to discuss what this new representation of an AChem is. In this work we treat this graph as a program. During execution we would have a single execution pointer on a control node which would move following the control edges around the graph executing the processes in the control nodes.

In this version our graph is a static object that is predefined before execution and remains unmodified by execution. This is the static graph form of MetaChem.

Our information flow edges can be seen as directing input and output of our nodes. In a way our container nodes act like “blackboard systems” (Hayes-Roth, 1988) being constantly modified and updated by our “experts” the control nodes. Samples in this work exist to section off part of a container and prevent it being modified by multiple ”experts” at once. In terms of a real blackboard they allow us to box the content of our tank and write ”Do not erase” next to it.

4.2 Overview of descriptive levels

To help the reader get an idea of the use and power of MetaChem at different levels of detail we introduce a “toy” chemistry, which we call StringCatChem. The purpose of this chemistry, which is simple and small enough to run by hand, is that it can be fully and succinctly decomposed into its parts.

This is a chemistry whose atoms are character stings and composites are formed via concatenation. StringCatChem is situated in a collection of well mixed tanks. When our particles combine or split they remain in the tank. When we select a string we check if it contains any double letters we split it between them. If not we select a second string at random and concatenate them. We also randomly transfer strings between tanks.

The simplicity of this system means we are unable to build particularly large particles since any selected particle that can split will. This means StringCatChem is a very bad choice of AChem if one wishes to study open-endedness or the transition to life. However it is simple making it perfect for our purpose here of giving an example of an AChem fully expressed in MetaChem at three different levels of abstraction: *Macro, Micro* and *Physics.*

4.2.1 Macro Level

The *Macro* level view is the view that rarely deals with individual values, atoms or interactions in the AChem. Even a generation or time step at this level is just an update process.

4.2.1.1 Macro StringCatChem

At the macro level StringCatChem loads a set of strings into a set of tanks. The observer **o**:time then iterates the time variable. We then have an **a**:process node which is responsible for the reactions that occur inside the individual tanks. This is expanded later in Figure 4.3 but results in new strings replacing old in the tank. Either by concatenating two strings together or by splitting a string between two identical letters. We track the number of reactions we’ve done per time iteration in **V**:reactions using the **o**:reactions observer. We use a decision to check if the update is complete and loop to the process if not. If the time step is finished we move to the sample **s**:transfers which moves particles between our tanks. We then loop back to **o**:time.

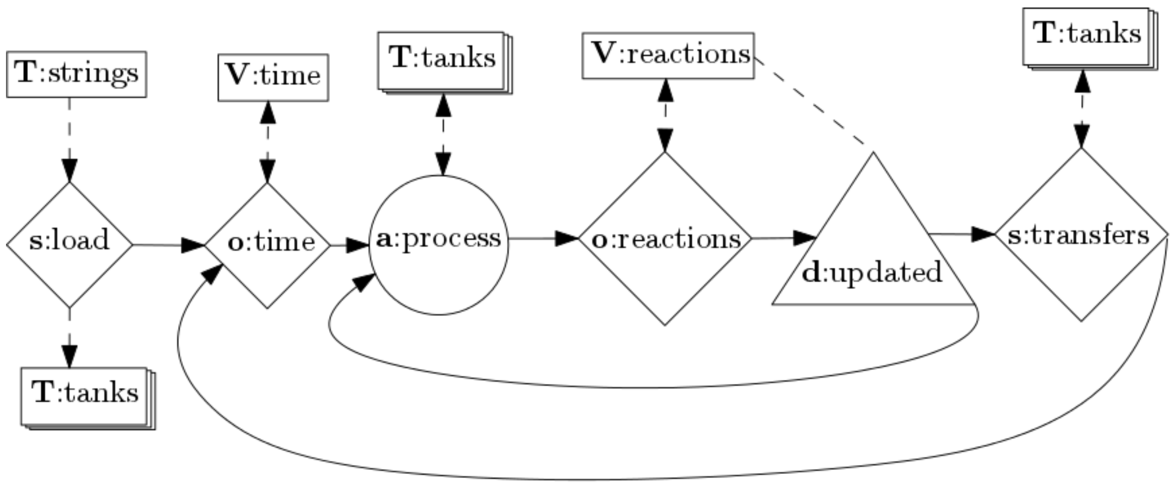


Figure 4.2: Macro level description of string concatenation chemistry.

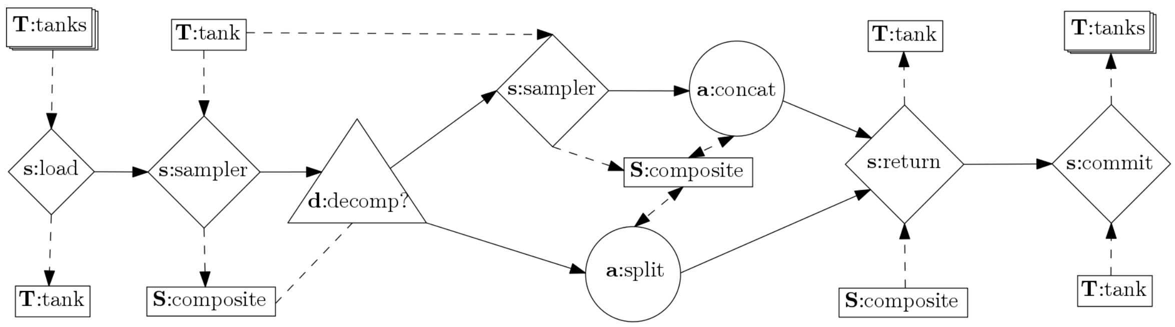


Figure 4.3: Micro level description of the **a**:process node of the StringCatChem.

This description gives us an eagle eye view of the main operating loops of our system and the set of significant processes. We can see here that we have a random unsynchronised update. Our timing is discrete and we have multiple tanks with movement between them. These are the sorts of elements and properties of our system that should be visible at the macro level.

4.2.2 Micro Level

The *Micro* level view focuses more on the actual action and effects on different particles and environments in the system. It can be thought of as the algorithm or pseudo-code level description of the AChem.

4.2.2.1 Micro StringCatChem: a:process

As an example we have expanded the **a**:process into a graph showing the internals of how this action occurs, Figure 4.3.

Briefly, we choose a tank, then choose a particle from it. We then decide based on the presence of a double character whether to decompose our string or not. In one case we then

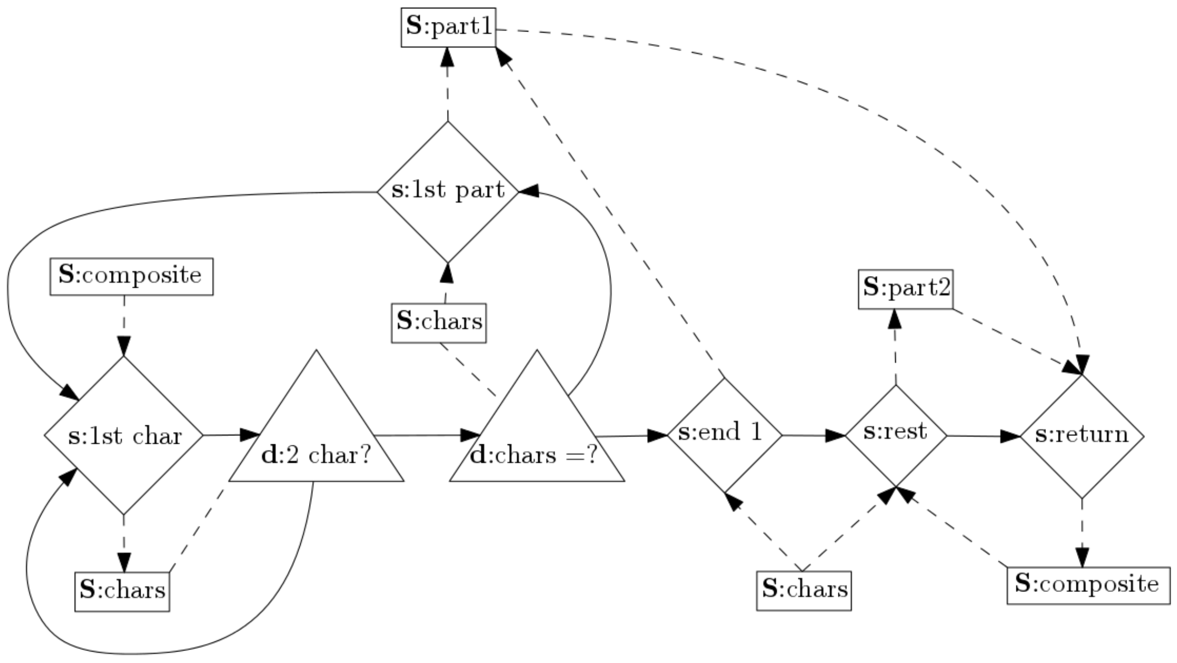


Figure 4.4: Physics level description of **a**:split node from StringCatChem **a**:process.

split the string in the other we sample a second string and concatenate them. The string(s) in **S**:composite are then returned to the same tank the original came from and the tank is returned to the collection of tanks.

This level of description more accurately mimics pseudo code or high level programming language description of the systems algorithm. This can become even more detailed at the physics level where we describe things at closer to machine code or hard coded level, see Figure 4.4 for a description of **a**:split.

4.2.3 Physics Level

The *Physics* level view deals with the hard-coded details of implementation. It is the designers choice what the lowest level of detail needed is but anything the designer considers to be unchangeable occurs at this level. This could be the program code level description or even the machine code level of description or one could even reduce this level of description to the movement of electron on the motherboard. The last however is never the correct choice though as it provides very little insight into our systems to treat actual physics as the physics of our system.

4.2.3.1 Physics StringCatChem: a:split

We can expand the **a**:split node from Figure 4.3 to the graph given in Figure 4.4.Notably here we have expanded an action node into a graph made only of sampler and decision nodes. This is quite reasonable when you consider we have moved from a high level program to low level. The NAND logic gate can give Turing complete computation on its own. Likewise we do not at our lowest levels of computation need an actual function or action to perform our computation. We can do it with just comparison and moving particles.

In Figure 4.4 we select our string particle a character at a time and check if the following character matches, based on this we funnel our string characters into two different particles which we treat as separate samples. We then place these two new particles back into the original sample container.

In the case of most processes the lower level instructions will not need to be explained as they will be simple or familiar processes.

4.2.4 Abstraction levels

These abstraction levels are designed to extend beyond the limitations of three levels. There can be systems made up of systems (Chapter 8) requiring additional macro levels and our physics layers can go down to give as much detail as is wanted. If one wished they could summarise their entire system as a single action node or could extend their physics level description down to logic gates and actions on the circuit board level. In most cases though these extremes are not needed or helpful. It is up to the designer or modifier of the Artificial Chemistry to choose from these levels for abstraction what is needed and useful to properly express and illuminate their own system.

4.3 Formalisation of MetaChem

4.3.1 Static Graph MetaChem

We provide a mathematical formalism here for our static elements. These consist of the mathematical elements that make up the static graph, G, of our system. In the following section we discuss the dynamic properties of our system and its static components.

We start with two graphs, G_Ω, I_Ω, of our system Ω, capturing control and information respectively. For the purpose of this explanation we assume we are always referring to elements of a single system and so drop the system label from all elements such that our graphs G_Ω, I_Ω become G and I.

As shown in our hierarchy, Figure 4.1, our system is composed of containers, control, control flow and information flow. Control and information flow appear in our static graphs as edges, E_G and E_I, of G and I respectively. Containers and control both appear in both our static graphs as nodes, N.

$$G = (N, E_G), \qquad I = (N, E_I)$$

Our graph nodes partition into two sets: Control nodes C and Container nodes B. (We use the notation $X = \langle X_1, \ldots, X_n \rangle$ to mean that the set X is partitioned by its n subsets X_i.)

$$N = \langle B, C \rangle$$

We can see in our hierarchy that each of these is partitioned further. In the case of the containers there are three categories of node: Environment nodes V, Tank nodes T, and Sample nodes S.

$$B = \langle V, T, S \rangle$$

Control nodes are more complicated in the hierarchy but the static components that partition the set of control nodes are: Action (C_a), Decision (C_d), Observer (C_o) and Sampler (C_s).

$$C = \langle C_a, C_d, C_o, C_s \rangle$$

We define an edge as pairs of nodes and are either control edges or information edges. Control edges are between control nodes

$$E_G \subseteq C \times C$$

Information edges come in three varieties.

$$E_{read} \subseteq C \times B, \quad E_{push} \subseteq C \times B, \quad E_{pull} \subseteq B \times C$$

The E_{read} edges are directed from control nodes to containers and indicate readable information for the control node, in visual graphs they are shown as undirected edges as there is no change to the container node. E_{push} edges are directed edges which indicate the containers edges the control nodes can push information and objects to. Similarly E_{pull} edges are directed and indicate the containers which a control node can remove information or objects from.

All E_{push} and E_{pull} must have a corresponding E_{read} edges. So $E_{push} \subseteq E_{read}$ and $E_{pull} \subseteq E_{read}^{-1}$.

4.3.2 Dynamic System State

Once we have these static graphs we can start a dynamic process guided by them. In order to do this we define the system state, $\omega \in \Omega_s$. In systems with a time element this may be denoted as ω_τ for the state of the system at time τ.

We denote the dynamic aspects of our system using the greek alphabet to distinguish it from our static components. Our system state is made up of five components at any time: the Control Graph (G), the Information Graph (I), the current state node (C), a mapping from the container nodes to the bag (or multiset) of particles they contain ($B \to \mathbb{P}(\Phi \to \mathbb{N})$), and the dynamic environment ($V \to \Upsilon$). The state of the system is:

$$\Omega_s = G \times I \times C \times (B \to \mathbb{P}(\Phi \to \mathbb{N})) \times (V \to \Upsilon)$$

	action	decision	sample	observer
read	✓	✓	✓	✓
check	✓			
pull	✓		✓	✓
process	✓	✓		✓
push	✓		✓	✓

Table 4.4: Transition functions used by different types of control nodes; unchecked functions always return their default behaviour

The system being formalised here has static graphs G and I, so we often do not include them explicitly in the state of the system.

The current state node C, a pointer in this static graph case, is a control node dynamically assigned and changing over time. This pointer indicates the current control node, whose transition is to be run in order to find the next state of the system.

4.3.3 Transition Functions

Each node has a transition function.

$$transition : \Omega_s \rightarrow \Omega_s$$

The overall transition function is decomposed into five component functions: *read*(), *check*(), *pull*(), *process*() and *push*(). For any node some of these may be null (identity) functions. Using function composition ($\bullet$) this gives the following definition of *transition*:

$$transition = pull \bullet process \bullet push \bullet check \bullet read$$

A graphical explanation for how this works is given in Figure 4.5. For some kinds of nodes, some components are always null or defaulted, Table 4.4.

Each of these transition function components plays a different role in the transition and thus uses a different aspect of the state.

The *read*() function allows our node to collect information from external containers into temporary local containers to be used by the rest of the transition functions. This does not modify the external containers in any way. In Figure 4.5 we see the default behaviour where the node reads all information from containers with information edges connecting to it into internal container nodes.

The *check*() function uses local information to generate a threshold probability which it checks against a uniform random number, r, to determine if the rest of the transition function (the part that actually alters containers) occurs. If our threshold probability is less than r we continue, otherwise we exit. In Figure 4.5 we see that our node uses the information it has read to choose between moving to the next control node y and moving to the *pull*() function.

The *pull*() function is then used to remove information from external containers where appropriate. The information removed has already been copied to local containers in *read*(), so is available for local processing. In Figure 4.5 we see a subset A^* of A is removed from the A container node, while the internal copy of the contents of node A remains unchanged.

The *process*() function acts as the main computation for the node. It modifies the state of local particles and variables including creating new particles and variables and destroying old ones. In Figure 4.5 we see that the internal information has now been changed and merged into a single internal container node. This is an extreme example as it has destroyed all previously read information in the internal container nodes, including the partitioning into four containers.

Finally the *push*() function pushes variables and particles from the local variables back to external containers and wipes the local containers contents. This preserves information for the system state but means the control nodes themselves do not have state or memory. In Figure 4.5 we see that the internal container has been destroyed but its contents has been added to container C so no information is lost but the control node x is now back in the initial state.

Transition functions operate on local state, which exists only for the duration of the transition. Let L_B be a local particle container and L_V be a local environment container. These are destroyed as soon as the transition function is completed, so the control nodes have no lasting state or memory. Any information used by a control node must come from containers at the start of a transition using the *read*() or *pull*() functions and any information or objects that should remain in the system should be written back to a containers by the *push*() function.

These operations are summarised in Figure 4.5 and formalised below.

4.3.3.1 read()

This function does not change the state of the systems containers. As a default *read*() copies of the content of all containers with read edges to a set of local containers, in practice it need not always read all available containers. This is a null function in terms of the system state. B_c and V_c are the sets of particle container nodes and variable container nodes connected by read edges to our control node c.

$$B_c = \{b|\exists(c, B) \in E_{read}\}, \qquad V_c = \{v|\exists(c, V) \in E_{read}\}$$

Then the sets of local container nodes, L_b and L_v within the control node after the read function have identical mappings to those of the nodes within B_c and V_c respectively.

$$\forall b \in B_c \quad \exists c \in L_b \quad s.t. \quad \phi(b) = \phi(c)$$

$$\forall v \in V_c \quad \exists w \in L_c \quad s.t. \quad \upsilon(v) = \upsilon(w)$$

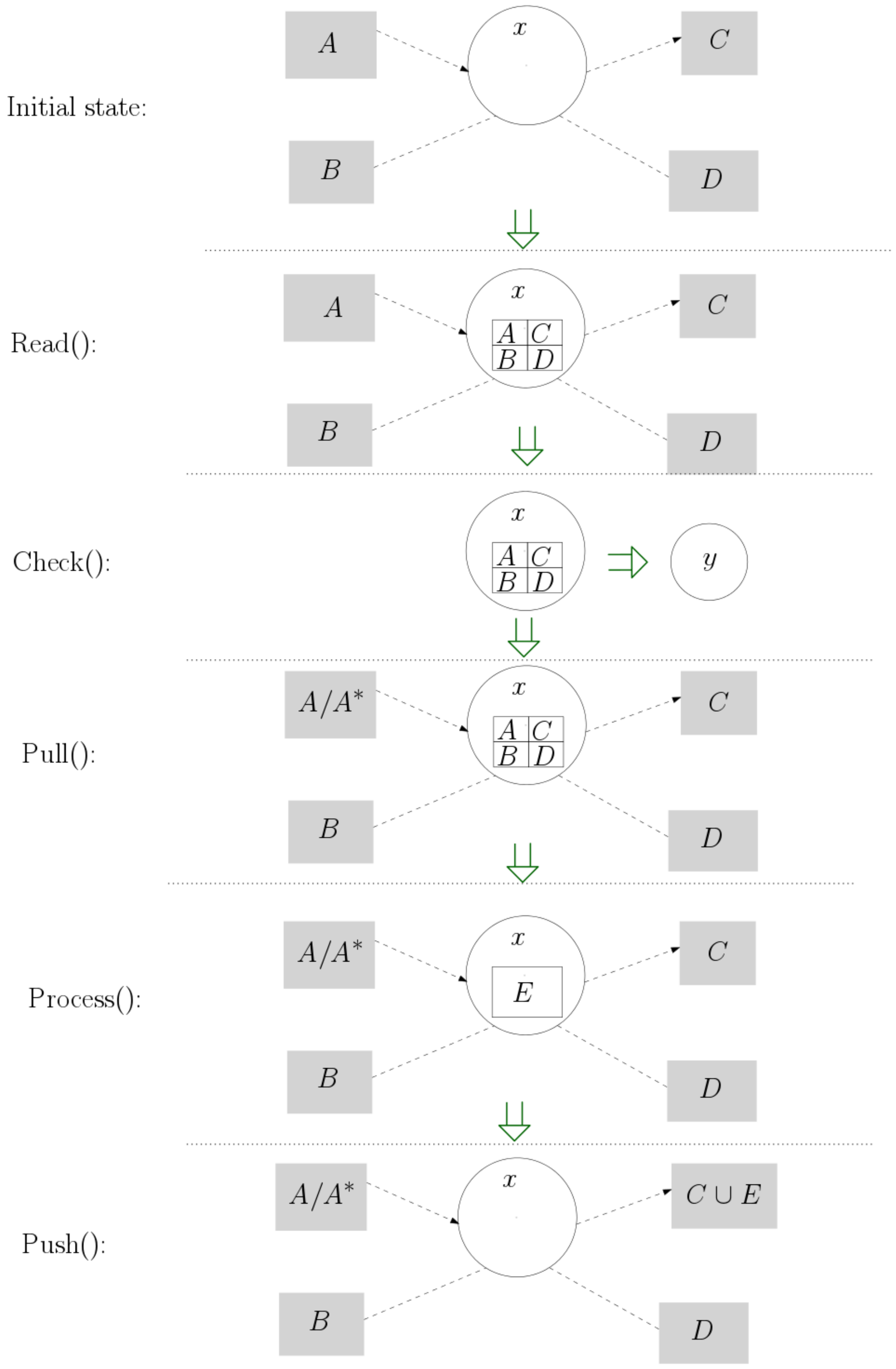


Figure 4.5: Summary of the effect of the five state transition operations

	action	decision	sampler	observer
tank			✓	
sample	✓		✓	
variable	✓			✓

Table 4.5: Containers which control nodes are allowed to have push and pull edges with.

where $\phi(b) \in \mathbf{P}(\Phi \to \mathbf{N})$ is the content of particle container b and $\upsilon(v) \in \upsilon$ is the content of variable container v.

Here we refer to the content of a container using the mapping from the container to the partitioned particle bag but in later chapters we begin to abuse our notation and refer to the content of containers by the container label, this is done for readability and brevity.

The default behaviour of this function is to read everything it can read, however this is not always needed and implementations may choose to only read a subset. This information is then written to the local containers.

4.3.3.2 check()

This function also makes no change to the state of the system, or the local state. The check function uses a psf (probability spawning function) (Faulkner et al., 2017) to determine if the rest of the transition will occur. The default behaviour in this case is to return true, which it does for administrative nodes which always operate in a deterministic manner.

$$\begin{cases} 0 & r > psf(\omega) \\ 1 & r \leq psf(\omega) \end{cases}$$

where $r \in [0:1]$ is a uniformly distributed random number.

On a return of 0 the transition function exits and does not proceed to any further functions, else the other functions are executed as expected.

4.3.3.3 pull()

The $pull()$ function allows the node to delete information or objects from external containers. These containers must be in the set of node connected by $pull$ edges in the information graph.

$$\{b|(b,c) \in E_{pull}, b \in B\}$$

where c is the current pointer.

The default behaviour of this function is to do nothing. This function needs only to delete information and objects as E_{pull} is a subset of E_{read}, so all information pulled should have been copied into local storage previously during the $read()$ function.

There are limits on the containers which different node types are allowed to pull from and push to, these are given in Table 4.5.

4.3.3.4 process()

This is the part of our transition which performs computation on our local information. This function is free to make any modification to the state of the local containers. It does not however have the ability to modify any other containers.

4.3.3.5 push()

The push command allows the node to add information or objects from the local containers to external containers. These external containers must be in the set of nodes connected by push edges in the information graph.

$$\{b|(c, b) \in E_{push}, b \in B\}$$

where c is the current pointer.

The default behaviour of this function is to do nothing.

4.3.4 Examples from StringCatChem

4.3.4.1 Micro level: s:sampler

As listed in Table 4.4, **s:**sampler has a read(), pull() and push() functions. This is a simple sampler which randomly selects a single particle to move to a sample container.

read(): It reads only two containers **T:**tank and **S:**composite. Thus after the read function the local particle container **s:**sampler$_B$ is in the following state:

$$\mathbf{s:}sampler_B =< \mathbf{T:}tank, \mathbf{S:}composite >$$

pull(): It selects at random a particle from the **T:**tank which it then delete from the **T:**tank.

push(): We push the same random particle to the **S:**composite sample.

4.3.4.2 Macro level: o:time

Observers have four of the five functions: read(),pull(), process() and push(). This is a basic observe that simply increments times.

read(): **o:**time has only one information edge: (**o:**time,**V:**time). This means it that after the read() function we have:

$$\mathbf{o:}time_V = \mathbf{V:}time, \qquad \mathbf{o:}time_B = \{\}$$

As **o:**time does not observe the state of any of our particle containers.

pull(): As our **V:**time container only contains a single variable our pull function clears the **V:**time container.

$$\mathbf{V:}time = \{\}$$

process(): This observer is part of a subset referred to as a counter observer as it pulls in a single variable and increments it. In this case the variable is time and the increment is 1. This is performed on the variable now in local storage.

push(): We push the local variable back out into the **V:**time container for storage as the local storage will be destroyed on termination of the transition function.

$$\mathbf{V:}time = \mathbf{o:}time_V, \qquad \mathbf{o:}time_V = \{\}$$

4.3.4.3 Micro level: a:split

Action nodes use all five transition function, or may use all five transition functions. In the case of **a:**split we use some of these functions and others operate under their defaults.

read(): There is only one information edge: (**a:**split, **S:**composite) for **a:**split. So the **S:**composite sample is read into the local particle container.

$$\mathbf{a:}split_B = \mathbf{S:}composite$$

check(): In this particular system reactions are deterministic so we use the default behaviour of the check function. This is to use $psf(\omega) = id(\omega) = 1 \quad \forall \omega \in \Omega$. So the check is always true and the rest of the action always happens.

pull(): We clear the content of the S:composite node as the **a:**split node will process the full content. This is the only possible pull given the only pull edge of **a:**split is (**S:**composite, **a:**split).

$$\mathbf{S:}composite = \{\}$$

process(): This is the function which can be split into the the actions of a metachem subgraph. In this case the function divides the string at the double letter. The two new strings are both stored in the local particle storage.

push(): Both parts of the string are transferred back into the S:composite sample. This is the only possible destination for these strings as the only push edge is (**a:**split, **S:**composite).

$$\mathbf{S:}composite = \mathbf{a:}split_B$$

4.3.4.4 Micro level: d:decomp?

Decisions only use two of the four transition functions. This is because they don't change the state of any of the containers, just the pointer status. This is done with a read and a process that returns an indication of which of the possible next control nodes the transition goes to.

read(): **d:**decomp? has a one read edge, (**d:**decomp?, **S:**composite), which is copied into the local storage.

$$\mathbf{d:}\mathit{decomp}_B = \mathbf{S:}\mathit{composite}$$

process(): This section performs the computation that makes the decision. It returns one of the nodes given by $\{c|(d:\mathit{decomp}?, c) \in E_c\}$. In this case $\{\mathbf{s:}\mathit{sampler}, \mathbf{a:}\mathit{split}\}$, where the return function is:

$$\begin{cases} \mathbf{s:}\mathit{sampler} & \text{if } p_n \neq p_{n+1} \quad \forall n \\ \mathbf{a:}\mathit{split} & \mathit{otherwise} \end{cases}$$

where p is the particle in **S:**composite and p_n is the nth character in p.

4.4 Static and Grown Graphs/Graph Language

In this work we are dealing with static graph MetaChem. The graph exists before the system is run and does not change at run time, similar to most programs. We have both a the mathematical formalism for this form of the graph and have implemented this form in code. This implementation relys on the user defining as set of nodes and a graph structure before running our system in terms of node transitions. The transitions are what we use to limit the length of the implementation as the graph has no built in concept of time with in the system just the number of times it has processed the node.

The advantage of a graph-based form is that with further work we will not be limited to such a static form.

The dynamic edge graph of MetaChem would allow the graph to add and remove edges during run time. Thus G would change in the system state, and we would add further control nodes that would be responsible for this change. In terms of our hierarchy, Figure 4.1, these nodes would fall under the grouping of control flow admin nodes. So the system could reorder its own control flow, or even connect entirely new nodes or subgraphs to the control flow that, while existing at the start, were not connected.

We could use these new nodes to trigger events in our system based on complexity. We could also use this as part of evolving Artificial Chemistries by having a set of nodes to start with and allowing our edges to change over time until the control flow became stable. This could be controlled by the system itself so it would "learn" an artificial chemistry.

For true evolution and change we move to graph language MetaChem (or dynamic graph MetaChem), which would create and destroy its own nodes and edges at run time, so the graph would grow and remove parts that were no longer needed, allowing it to prune its own process. This type of system would allow the artificial chemistry to change completely at run time so it could truly transition and change abstraction levels and experiments as it ran. This could open paths in open-ended evolution and open-ended systems research which have not been possible before.

This system will give us changes in our artificial chemistry's similar to the change in chemistry as we move from atoms to organic molecules. This is the difference between undirected reactions and a chemistry that runs alot like a machine purposefully manufacturing new chemicals needed. This is a very different system to the initial presumed starting state of a stochastic chemical soup. There are other examples such as the development of bacteria and cells which makes these sorts of machines properly contained and mobile. This was not possible with out breaking until they have that sort of distinction.

Finally we have MetaChemistry the artificial chemistry itself to consider. Once we have the dynamic graph language MetaChemistry, we can consider any isolated MetaChemistry subgraph as forming a subAChem or a full AChem. We can treat nodes as atoms and subAChem and AChems as composite particles. An instance of MetaChemistry is there for not a single graph but a collection of graphs (possibly all possible graphs). This gives us MetaChemistry as the universe composed of Artificial Chemistries.

4.5 Next Steps

We now have a formal language in which to discuss different Artificial Chemistries. All current systems we are aware of can be described by the static graph MetaChem.

To show the power of this modularisation and graph based representation, we next present two case studies of Artificial Chemistries. The first of these is our own chemistry Jordan Algebra AChem, originally developed based on algebraic structures, but here re-described in the MetaChem format. The second AChem is Swarm Chemistry, chosen for being a well-established artificial chemistry. Swarm Chem and JA AChem are very different chemistries with next to no overlap in their nature. Swarm Chem also represents an independent example of description of an existing AChem in MetaChem.

We then show that these descriptions can enable further investigation of these AChems. Our first example of this is an in-depth investigation of particular behaviours of JA AChem by variation of a single process. Our second example is the combination of these two chemistries to create a set of new more complicated AChems through the creation of a small set of nodes.

Chapter 5

Jordan Algebra Artificial Chemistry

Jordan Algebra Artificial Chemistry (JA AChem) was initially designed at the beginning of this work to conform to algebraic structures observed in natural chemistry. It was this philosophy of defining the low level algebraic stuctures of a system and building on them to develop an algorithm and experiments that lead to the conception of MetaChem. In this chapter we re-describe JA AChem using MetaChem as a case study. In the following chapter we show how this description makes modifications to JA AChem simpler.

5.1 Introduction

Our MetaChem system, while designed subsequent to our original Jordan Algebra Artificial Chemistry (Faulkner et al., 2016), can allow us to describe this algebraically driven chemistry. We do this using the graphs and functions described in Chapter 4. In this chapter we focus on the mathematical and conceptual motivations of this chemistry, while using MetaChem as a clear language for how our system fits together into a complete algorithm. Later we see how this separation of factors allows us to manipulate and tune our Jordan Algebra AChem (Chapter 6) and combine with another AChem (Chapter 8).

5.2 Open-Endedness through rigorous design

Most Artificial Chemistries (AChems) seek to produce a system capable of displaying specific behaviours associated with abiogenesis, the transition from inorganic to organic (living) materials (Hutton, 2002; Lucht, 2012; Suzuki et al., 2003). Those systems succeed in generating their particular behaviours because that is what they are designed to do. Another approach is to consider that we are seeking open-ended behaviour in our systems. In order to design for open-ended behaviour we need to approach the problem in an open-ended way.

We need to design a system that is rich and complex, with properties that allow us to define all the reactions of our AChem implicitly. We can then start looking for, and finding, behaviours that are emergent from the design, rather than engineered explicitly. We need a set of building blocks and connectors that do not limit the structure we design. Think of this as the difference between a prefabricated house and a brick house. A prefab has pieces that are specifically designed to fit together and form a house, and have a limited capability to do anything else. A brick house is just the bricks and the mortar that joins them. The bricks are not limited to building a certain house, or even a house of a particular size. We could even build a bridge instead or some other structure. With enough bricks and mortar the possibilities are endless. Likewise in an open-ended AChem the only limit should be the material and the amount of energy in the system.

We also need to consider not just constructive but destructive behaviours in our design. Not all reactions need to be irreversible. In fact this is not a desirable property in any system seeking to be open ended with a reaction that combines particles into a larger structure. This is because without destruction we will quickly reach a point where all possible links will have been made and nothing more will change. However destruction brings the potential for links to break and new links to be made.

We need to consider desirable properties of the interactions of our particles, rather than of the whole system, while ensuring that we do not over- or under-constrain the AChem. Here we do this by taking a mathematical approach, and taking advantage of existing mathematical theory and structures. This allows us to discuss not just the properties and behaviours of the particles, but also the different links and linking structures between them. We can then use established mathematics that has many emergent properties with interesting forms of interactions. We can also expand our view to talk about the effects of these properties on the system as a whole.

5.3 Assessment of Desirable Mathematical Properties of Link for correct structure of Composite Particles

Our ssAChem (subsymbolic artificial chemistry) rules have two functions: a set of mathematical products (or mathematical operations) for forming links and composites, and a set of probabilities used to determine probability of a reaction (probability spawning functions, psfs).

In this section we discuss the properties of the mathematical product function. The probabilities will be discussed briefly later and in more detail in the following chapter.

5.3.1 Desirable Properties for Particles

Before we discuss the desirable properties of an operator for joining our particles we need some idea of desirable properties for the particles themselves. We are looking to build a subsymbolic system so our particles will first be required to have some sort of internal structure and properties.

We are also taking an algebraic approach to our system. We therefore want particles which can work with our algebraic operator. If possible we want therefore well known and well studied mathematical objects. For instance the integers, **Z**, do have internal structures. There are various different possible decompositions of an integer, most well known would be the prime factorization but there is also 2-factor integer factorization often used with large primes for cryptography.

However a richer object would be matrices which have many defined operators and mappings that could be used in this work. More interestingly perhaps there are many well defined interesting subsets of Matrices and they come in different sizes such that we could tailor our system to work with a subset well suited to our operator to give us an atomic set as well as the general particles.

5.3.2 Algebraic Axioms

In mathematics there are two properties of a product on a set that are easily defined, and that can be indicative of many further properties of an algebra. These are associativity and commutativity.

$$\text{Associativity:} \quad (a \circ b) \circ c = a \circ (b \circ c) \tag{5.1}$$

$$\text{Commutativity:} \quad a \circ b = b \circ a \tag{5.2}$$

When we have a binary product, thereby linking two particles, combinations of these properties lead to four distinct structures, Table 5.1.

For an associative, commutative binary product we can change the order of evaluation and the ordering within any evaluation. No matter how we link a given set of particles, we get the same result. The structure is a bag. For an associative, non-commutative binary product we can change the order of evaluation such that there is no ordering on the products, but we cannot change the ordering within the product; the structure is a string.

Associativity is an assumed property of most algebras. Non-associative algebras, while rare, normally appear in an applied setting. They have been used in connection with genetics (Reed, 1997) and physics (McCrimmon, 1978) as well as a broad range of applications to mathematical theory (Gonzalez and Martinez, 2003). One of their main attractions is that with their enforced evaluation order they can embody a loose form of time, or at least an ordering of interactions.

For a non-associative, commutative product we can reorder particles in a product, but we have an enforced order of products. The structure is a binary tree, with unordered child nodes. For a non-associative, non-commutative product, we have an enforced order of products and ordering of particles within the products. The structure is a graph, with

Associativity	Commutativity	Structure
Yes	Yes	Bag
Yes	No	String
No	Yes	Tree
No	No	Graph

Table 5.1: Summary of structure provided by different mathematical properties

complicated directionality restrictions requiring labelling on both edges and nodes; these are not simple structures and do not conform to any of the normally used graph subtypes.

Let us consider these four structures in terms of an AChem. A bag has no internal structure, and limits us to a set of composite particles with the cardinality of the power set of the component particles. In real chemistry there are *isomers*: molecules with different inherent properties despite containing the same atoms in different arrangements (Muller, 1994). Isomers add complexity and increase the size of the combinatorial space. An AChem with a bag structure has no equivalent of isomers, so we do not want to base ours on an associative commutative product.

In terms of the real world we can see that bag structures are rare. The world orders itself. things group together in structures. Gases and liquids may seem like bag structures until you see a flow or a convection current that make the whole more than just the individual objects within. Strings are more common appearing in DNA and proteins. In physics even signals produce data with a string type structure and these signals are produced by the stars themselves. Trees though seem to be the mark of life itself. Reproduction either sexual or asexual produces a tree structure in terms of ancestors. As previously stated chemistry has a tree structure. Finally we have graphs that appear everywhere. In physics we can find graphs of force interactions. In chemistry we get reaction networks whose structure are graphs. In biology, sociology, environment all have social and resource graphs for interaction and resource access.

Strings are structures that have received a lot of attention in the computing community, but they are rather simple mathematical objects that lack room for expansion. They have very simple combinatorial power of

$$C_{n-1} = \frac{(2n-1)!}{n!(n-1)!} \tag{5.3}$$

Strings support analogues of isomers, but there are not many of them. There is also no ordering of operations, so how they are formed does not affect the result. So we reject associative non-commutative products.

The tree structure given by the non-associative commutative product not only has more room for expansion to larger trees, it also has an implicit ordering. Because we cannot change the order of operations we get a variety of structures, and a system in which structure is as important as the building blocks themselves. This gives us a system with greater intrinsic flexibility.

The graph structure of a non-associative non-commutative product provides yet more structure, but makes it hard for the product to have any regularity to exploit as it allows so many possible structures. It is not necessary that we work with a structure this complex so we stick to trees.

Keeping the structure simple is particularly a concern in our system as we have both composition and decomposition which will require us to keep a record of this structure in case it breaks apart later.

5.3.3 Expanding to n-ary linking operator

We can look beyond binary products to products that take more arguments, combining multiple particles with a common link.

This does not affect the structure of the system if we have an associative non-commutative product: since it is associative this changes nothing and we still have a string. However in the case of the non-associative commutative product as we expand from a binary product to a larger product we move from a binary tree to a general tree.

There are a larger number of possible trees with $n \geq 4$ leaves than strings with $n \geq 4$ elements. For $n = 3$ we have $s_3 = 6$ and $t_3 = 4$, where s_n is the number of strings with n elements and t_n is the number of trees with n leaves. For $n = 4$ we have $s_4 = 24$ and $t_4 = 37$ using products of any size, see Figure 5.1. We can show that from this point onward there is a larger number of possible trees than strings.

The number of possible strings increases with n such that $s_{n+1} = s_n(n+1)$. For trees we have a faster growth. We can show that if we link the extra element to the result of each of the graphs with n nodes with a binary link then the new element can be swapped with any of the other elements to give at least $t_{n+1} \geq t_n(n+1)$. We also always have more graphs as this does not include the graph of the $(n+1)$-product (see Figure 5.2) making t_{n+1} strictly greater than $t_n(n+1)$. Thus as we have more trees at $n = 4$ and a faster growth in the trees than in the strings, for $n \geq 4$ we always have more possible trees than strings.

In terms of an AChem, these properties show that we have a more interesting selection of possibilities in a non-associative system than otherwise, and these possibilities are controlled by the order in which reactions occur. Hence we focus on non-associative commutative products for our ssAChem design. This means that our particles will consist of a mathematical subsymbolic object with internal structure and a structure emergent from the combination of particles used to create the particle, unless it is atomic.

5.3.4 Jordan Algebras

Having established that these mathematical properties are desirable, we need to find a system in which we have these properties. Mathematics as a field has already found and studied

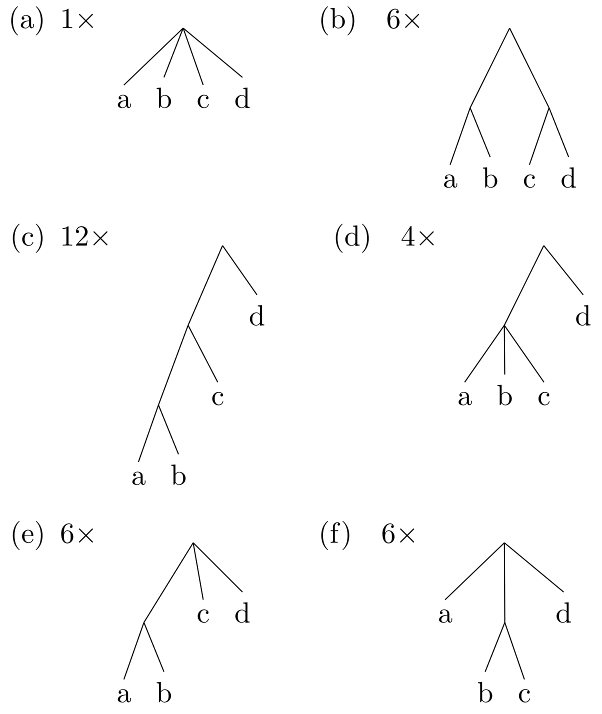


Figure 5.1: Tree structures with four leaves with multipliers indicating the number of relevant rearrangements of leaves, giving an indication of all possible trees with four leaves with in this system.

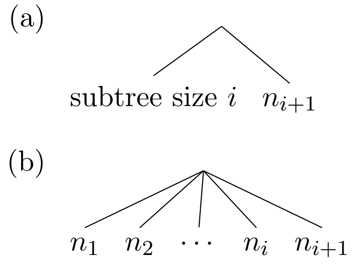

(a)
subtree size i n_{i+1}
(b)
n_1 n_2 $\cdots$ n_i n_{i+1}

Figure 5.2: Trees showing greater growth than strings

systems with such properties, in the case of non-associative commutative systems we have Jordan Algebras (McCrimmon, 2006).

Jordan Algebras were originally conceived to find a solution to describing observables in quantum mechanics, but were later discarded for that purpose because none of the Jordan Algebras were able to solve the problem. They have two important properties which define them:

$$\text{Jordan identity:} \quad (x \bullet y) \bullet x^{\bullet 2} = x \bullet (y \bullet x^{\bullet 2}) \tag{5.4}$$

where $x^{\bullet n} = x \bullet x \bullet \cdots \bullet x \quad (n \text{ times})$

$$\text{Power associative:} \quad x^{\bullet m} x^{\bullet n} = x^{\bullet (m+n)} \quad \forall m, n \geq 0 \tag{5.5}$$

Power associativity tells us what happens when we work with just one kind of particle.

There are several Jordan Algebras (McCrimmon, 2006). One of these exists over the Hermitian matrices (a matrix is Hermitian if it equals its Hermitian conjugate, see Equation 5.14). These are a well defined subset of square matrices, allowing us to choose the size of the matrices we use. This makes Jordan Algebras the perfect choice for our chemistry.

With this Jordan Algebra we start with a binary product formed of familiar matrix multiplication and addition to define the Jordan product:

$$X \bullet Y := \tfrac{1}{2}(XY + YX) \tag{5.6}$$

As one can see $X \bullet Y = Y \bullet X$. It is also non-associative:

$$\begin{aligned}
(X \bullet Y) \bullet Z &= \tfrac{1}{2}(XY + YX) \bullet Z && (5.7)\\
&= \tfrac{1}{4}(XYZ + YXZ + ZXY + ZYX) && (5.8)\\
&\neq \tfrac{1}{4}(XYZ + XZY + YZX + ZYX) && (5.9)\\
&= \tfrac{1}{2}X \bullet (YZ + ZY) && (5.10)\\
&= X \bullet (Y \bullet Z) && (5.11)
\end{aligned}$$

One of the advantages of Jordan algebras is the existence of an expansion from the binary product and the binary tree to a general product and its general tree. We can expand the binary product linearly to give the Jordan triple product:

$$\begin{aligned}
\{X, Y, Z\} &= (X \bullet (Y \bullet Z) + (X \bullet Y) \bullet Z - (X \bullet Z) \bullet Y)\\
&= \tfrac{1}{2}(XYZ + ZYX) && (5.12)
\end{aligned}$$

We can further extend this to an arbitrary length n product, called an n-tad in Jordan theory (McCrimmon, 2006):

$$\{X_1, X_2, \cdots, X_n\} = \tfrac{1}{2}(X_1 X_2 \cdots X_n + X_n \cdots X_2 X_1) \tag{5.13}$$

Using the n-tad notation, $(X \bullet Y) = \{X, Y\}$.

Commutativity of this product means that we can fully reverse the order of the elements in the product, but not freely rearrange the order completely. So there is a large number of possible n-tad products for a particular set of n objects, increasing our combinatorial power and the ability of our system to exploit some properties of composite particles. Thus Jordan Algebras equip us with products that are open-ended, and are applicable to the open set of Hermitian matrices.

5.4 Particles and their subsymbolic structure

Other AChems have used 'matrices' as the basis of their particle set S. In particular, the binary string chemistry (Banzhaf, 1993), dubbed the matrix-multiplication chemistry by Dittrich et al. (2001), makes use of matrix multiplication. However, it does not treat its particles as mathematical objects; rather, it folds binary strings into a matrix in order to give a simpler definition of a function over the binary strings. This is common for the use of 'matrices' in systems that use 'matrix' to mean a two dimensional storage array rather than the mathematical object that we use here.

All of the previous discussion in this chapter has been building towards creating a system that uses mathematical objects for both the particles *and* links of our system. This is the beauty of a mathematical product: it is in some ways an object with properties in its own right.

Additionally, the matrices themselves are rich in emergent properties that might be exploited by our system.

5.4.1 Hermitian Matrices as Subsymbolic Particles

The atoms in the Jordan Algebra AChem used here are 3×3 Hermitian matrices.

Hermitian matrices use the Hermitian conjugate of a complex matrix:

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix}^{\dagger} = \begin{pmatrix} \bar{a_{11}} & \bar{a_{21}} & \bar{a_{31}} \\ \bar{a_{12}} & \bar{a_{22}} & \bar{a_{32}} \\ \bar{a_{13}} & \bar{a_{23}} & \bar{a_{33}} \end{pmatrix} \tag{5.14}$$

The elements a_{ij} are complex numbers, and $\bar{a}$ is the complex conjugate of a. A matrix M is Hermitian if $M = M^{\dagger}$. Hermitian matrices are closed under the Jordan product (McCrimmon, 2006).

Hermitian matrices provide a rich variety of properties such that we can use them as prime material for creating a subsymbolic AChem (ssAChem) where emergent properties of the matrices dictate the linking capabilities/probabilities of a particle, and the algebra gives the structure of the composite particles.

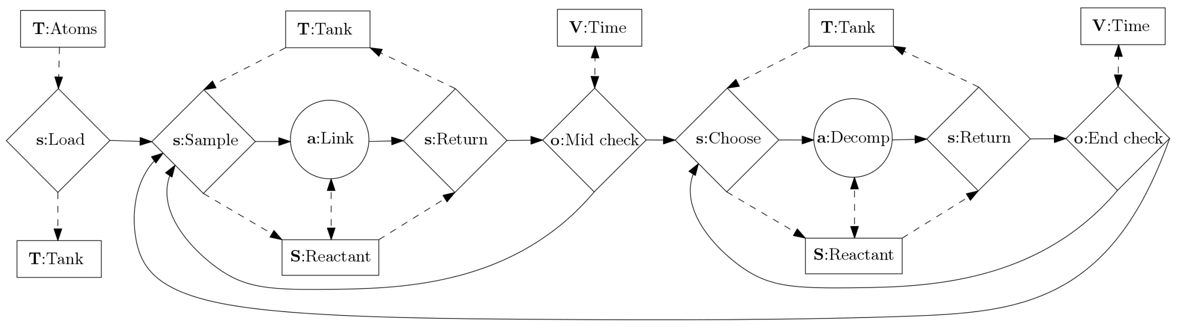


Figure 5.3: Macro level description of JA AChem operating over link and decomp loops

5.4.2 Atoms

We have 14574 atoms with 66 different sets of eigenvalues. Many of these overlap and are different. We have a lot of options and a lot of different sorts of operations and linking behaviours. The atom set is given in Equation 5.15.

$$\left\{ A = \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix} : a_{ij} \in \pm 1, 0, \pm i, \pm 1 \pm i \right\} \tag{5.15}$$

5.5 Macro Level Description of Jordan Algebra AChem

Now that we have described the motivation behind the system and the objects within it, we describe the system as a whole at the macro level, including observations, processes and containers.

Our algorithm loads our initial atom or particle set and then operates over two loops. These two loops are remarkably similar, starting with a sampling of the tank followed by an operation. The loops finish by returning their samples, updating timing variables and checking if enough operations or time has passed to say whether the loop continues or if the system moves to the other loop.

If we make observations of our system, we add them to our logger, which may or may not be included in our system. The logger pushes to an external environment which is never pulled from. These observations can provide many different summary statistics. In later examples relating to linking and probabilities we will observe: number of atoms in each particle, number of different atoms in each particle, size of particle trace, weight of particles, size of largest link in particles.

This macro system level description does not clarify the internal workings of our link and decomposition nodes. These are expanded in the next two sections to give the micro level linking and decomposition processes for this work in terms of the mathematical and graph terminology given in the previous chapter.

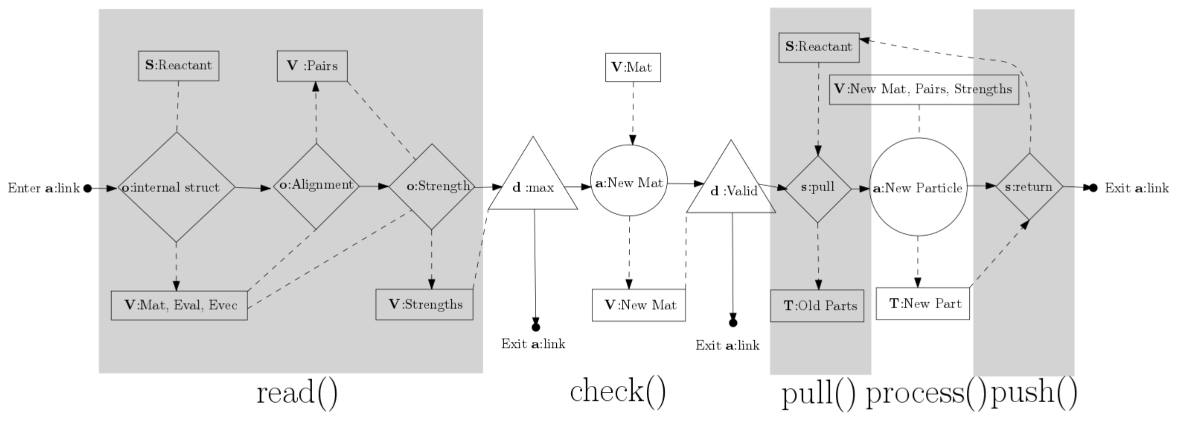


Figure 5.4: Micro Level description of **a:**link node from Figure 5.3. Below are the labels for the grouping of control nodes according to transition function in the the outer node. External nodes are depicted my a double box. All other containers are lost at completion of control flow.

5.6 Linking Process

The **a:**link action has all five transition functions. The read, pull and push functions are all performed from and to the **S:**reactant sample. More interesting are the check and processing functions in this case. To highlight these actions we expand the **a:**link action node into a micro level subgraph of the macro system. This means that all five transition functions are made more explicit, and we see the local containers L_B,L_V.

5.6.1 read()

The observers gathers the information needed for the probability check. They do this by reading and processing information from **S:**reactants. The information about the internal structure and other derived values are stored under various names in the local environment storage. In Figure 5.4 we denote this by labeling our environment with the variable names for each element being stored there.

Our first observer extracts the matrix, normalised eigenvalues and unit eigenvectors for each particle in our samples reactants. For a matrix M, consider

$$Mv = \mu v \tag{5.16}$$

The solution vectors v_i are the eigenvectors; the corresponding scalars μ_i are the eigenvalues. Here we choose these unit eigenvectors and the corresponding normalised eigenvalues λ_i as our emergent properties of interest to define our linking probabilities:

$$\lambda_i = \mu_i / \sum \mu_j \tag{5.17}$$

We normalise the eigenvalues to ensure sensible linking probabilities of larger composites.

Our second observer reads those values and uses them to calculate the best pairs of eigenvalues to use based on their “alignment”. This does not take into account the sign of their eigenvalue. This was done on purpose to create a situtation where a particle can happen to be generated such that it is ”facing the wrong way” for linking. This is something we see as adding dimension and richness to our system.

$$a_{A_iB_j} = \left(1 - \frac{1}{2}((v_{A_i} \cdot v_{B_j}) + 1)\right)$$

Alignment uses the dot product between the corresponding unit eigenvectors of each pair of particles. The dot product between two unit vectors is the cosine of the angle between them. The overall term has a value between 0 and 1, and is 0 if the vectors are perfectly aligned and 1 if the vectors are anti-aligned. We choose the highest value pair for each pair of particles and record the pair (i,j) and its alignment value in **V:**Pairs.

Our third observer uses this information to then calculate the strength between each pair of particles in our reactants. Using the (i,j) pair selected by the alignment in the previous step.

$$s_{A_iB_j} = \mathcal{N}(\lambda_{A_i} - \lambda_{B_j}) \tag{5.18}$$

This is the probability density of the normal distribution ($\mu = 0$, $\sigma = 1$) at the point given by the difference in the normalised eigenvalues. This means the probability of linking is larger for more similar normalised eigenvalues. The normal distribution is not the only option; we simply need a symmetric distribution centred on zero, and the normal distribution is a well-known such one. We investigate other options for this distribution in Chapter 6.

This ends our read() function, which has done a lot of processing but has only read from the **S:**Reactants. Processing does not normally occur in the read action, but as we are taking the lower level view of this function we can reveal more of our implementation and with it the shortcuts we take to ensure less repetition of calculations and a smoother flow. This is why we are using observers for our read function.

5.6.2 check()

Our check() function in the **a:**link node looks like:

$$\begin{cases} pass & r < \text{particle strength} \quad \& \quad \text{Matrix valid} \\ fail & otherwise \end{cases}$$

In our micro level view we break this down into two decisions. The first compares the minimum of the linking probabilities with a uniformly distributed random value on the interval [0:1]. If the minimum linking probability is more than the value, the decision node transitions to the next macro level node, else we continue with our **a:**link. This means the weakest pairing dictates if our link happens.

The probability of linking check based on the strength of the link and its alignment.

$$p_{A_iB_j} = s_{A_iB_j} a_{A_iB_j} \tag{5.19}$$

In-between our two decisions we again sneak in some early processing. We use a deterministic action (one that always runs in full) to calculate what the new matrix would be if we did create the new particle. We store this for possible later use as **V:**New Mat.

Our second decision uses this result to make its decision based on the trace of the new matrix. As long as the matrix has a non-zero trace we continue otherwise we exit the a:link node with out ever pulling information.

5.6.3 pull()

This single node action just empties the existing reactants out of the **S:**reactants container. We have collected all the information we need from them to create our new particle and no longer need these particles, they are dumped in a local container which will disappear at the end of the process.

5.6.4 process()

This single node uses the new matrix to create a new particle. To do this it creates a link with a strength (this will be used later). In the case of a binary link the strength is the $s_{A_iB_j}$ value we have already calculated, for larger links it is the minimum of the set of these. This link also includes the memory of the reactants that were used to form the link, taken from our local container. These properties, the list of eigenstate pairs used and the matrices are used to generate a new particle object which is stored in its own labeled section of the the local storage, **T:**New Part.

5.6.5 push()

Finally we use a sampler to move the content of **T:**New Part in to the empty **S:**Reactants and then exit the node. Exiting the node means losing the content of all the local containers, so there is no need to explicitly delete the generated information.

5.6.6 Subsymbolic Link

Because of the subsymbolic nature of our link, our composite particles can be seen as having more complicated overall strengths and probabilities.

The probabilities and strengths above relate to a single reaction but a composite particle can be formed over the course of many reactions. So the final particle has a different total strength and probability of existing in our system.

5.6.6.1 Composite Particles Probability

In our system the links have properties of their own. Equation 5.19 is the probability of the link forming given the presence of its components and that we choose that many reactants.

We need a further two probabilities to work out the probability of the resulting composite of the link existing, f_A. We need a probability of a particle existing, e_A:

$$e_A = \begin{cases} 1 & \text{if } A \text{ is an atom} \\ f_A & \text{if } A \text{ is a composite} \end{cases} \tag{5.20}$$

And we need a probability that each particle takes part in the reaction. All reactions that form a link require us to select at least two components. We define the probability of selecting further components for the reaction as 0.1 for each component. This choice discourages our system from constantly forming large links and quickly becoming one large composite. This is implemented within the **s:**Sample node in the macro level description.

$$r_{\{A_1,\cdots,A_n\}} = 0.1^{n-2} \tag{5.21}$$

So together the probability of a composite forming in terms of its last link is:

$$f_X = f_{\{A_1,\cdots,A_n\}} = p_X r_X \prod_{i=1}^{n} e_{A_i} \tag{5.22}$$

5.6.6.2 Composite Particles Strength

Link strength in our system is truly a property of the link rather than the composite. The composite particle contains a series of links all with different properties; each link has the link strength given in Equation 5.18. The overall composite strength is given by the probability of each of the links not decomposing, and is the product of all the link strengths in the system. The result is that larger links are stronger than a series of smaller links as there is less chance for the composite to break down. This is given by:

$$s_X = \prod_{Y \subset X} l_Y \tag{5.23}$$

where l_Y is the link forming Y a sub-particle of X.

5.6.7 Elemental Table

One of the results of this model of linking is that the defining linking characteristic of an atom is its eigenvalues. The eigenvectors may be considered simply rotations of the same atom if they are paired with the same set of eigenvalues.

This allows us to categorise our atoms into elements, with in which the isotopes have the same linking properties but different orientations. This is equivalent to saying that we are more interested in the linear mapping represented by the matrix than in its orientation.

We start with our atomic set of 14,574 atoms which we can partition into the isotopes of 69 elements defined by eigenvalues. For each different set of eigenvalues we assign a set of capitol letters which are reused if the set of eigenvalues appear again. We do the same thing with atoms traces to give us a label for each element. These can be sorted into a periodic table. To do this and to find a representative isotope for each atoms we define property which we call "weight":

For matrix $A = (a_{ij})$ and $a'_{ij} = a_{ij} \quad mod \quad 3$:

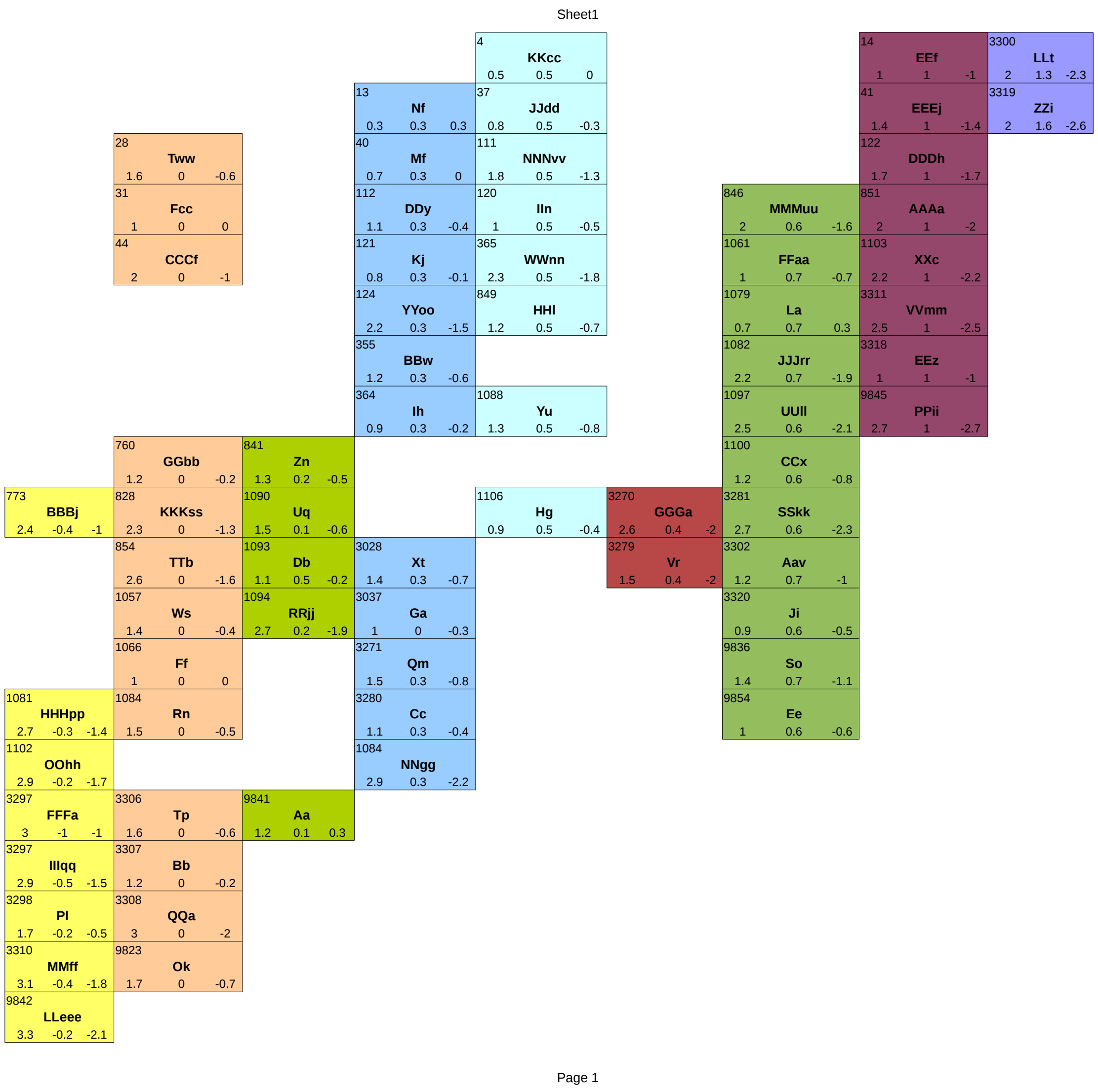
$$\begin{aligned} w_A =& a'_{11} + 3a'_{22} + 3^2 a'_{33} \\ &+ 3^3 Im(a'_{12}) + 3^4 Im(a'_{13}) + 3^5 Im(a'_{23}) \\ &+ 3^6 Re(a'_{12}) + 3^7 Re(a'_{13}) + 3^8 Re(a'_{23}) \end{aligned} \tag{5.24}$$

By minimising this we designate the representative isotope to be the isotope with the largest number of zero value real and imaginary elements ($mod \quad 3$ sets $0 \cong 0$, $1 \cong 1$ and $-1 \cong 2$). Since we are minimising we prefer zero or positive elements. We particularly penalize real numbers off the diagonal and prefer numbers on the diagonal. We organise this in the period table such that similar sets of eigenvalues are in the same column as each other and so that nothing is below or two the left of another element with a higher weight.

To determine if this is a good definition of an isotope we generate the self-synthesis of each atom with itself and check given different groupings how similar the linking values of the resultant composites are. We do this across the entire set, the set of trace based partitions, the set of linking value based partitions and the set of isotopes of each element. This gives us the distributions shown in Figure 5.6, as is shown isotopes/atoms with the same trace or the same linking values are identical under self-synthesis, this means the element is well defined as it incorporates both of these strong similarities. It is also important to note that while there are 66 different sizes and 66 different sets of linking values there are 69 elements meaning these are not all the same partition so it is important to use both metrics rather than one or the other. It also means that for future work due to the similarity of isotopes it is valid to simply use the isotope with the lowest weight to represent the element most of the time.

5.6.8 Structure

The Jordan Algebra underlying this system means that the structure of the composite is as important as the particles that make it up. Through this we can see that not only does the structure add to the properties of the composite, but also we can find behaviour in the structure independent of the particles. By this we find the analogue of an *isomer* from real chemistry.



Sheet1

				4 **KKcc** 0.5 0.5 0			14 **EEf** 1 1 -1	3300 **LLt** 2 1.3 -2.3
			13 **Nf** 0.3 0.3 0.3	37 **JJdd** 0.8 0.5 -0.3			41 **EEEj** 1.4 1 -1.4	3319 **ZZi** 2 1.6 -2.6
	28 **Tww** 1.6 0 -0.6		40 **Mf** 0.7 0.3 0	111 **NNNvv** 1.8 0.5 -1.3			122 **DDDh** 1.7 1 -1.7	
	31 **Fcc** 1 0 0		112 **DDy** 1.1 0.3 -0.4	120 **IIn** 1 0.5 -0.5		846 **MMMuu** 2 0.6 -1.6	851 **AAAa** 2 1 -2	
	44 **CCCf** 2 0 -1		121 **Kj** 0.8 0.3 -0.1	365 **WWnn** 2.3 0.5 -1.8		1061 **FFaa** 1 0.7 -0.7	1103 **XXc** 2.2 1 -2.2	
			124 **YYoo** 2.2 0.3 -1.5	849 **HHl** 1.2 0.5 -0.7		1079 **La** 0.7 0.7 0.3	3311 **VVmm** 2.5 1 -2.5	
			355 **BBw** 1.2 0.3 -0.6			1082 **JJJrr** 2.2 0.7 -1.9	3318 **EEz** 1 1 -1	
			364 **Ih** 0.9 0.3 -0.2	1088 **Yu** 1.3 0.5 -0.8		1097 **UUll** 2.5 0.6 -2.1	9845 **PPii** 2.7 1 -2.7	
	760 **GGbb** 1.2 0 -0.2	841 **Zn** 1.3 0.2 -0.5				1100 **CCx** 1.2 0.6 -0.8		
773 **BBBj** 2.4 -0.4 -1	828 **KKKss** 2.3 0 -1.3	1090 **Uq** 1.5 0.1 -0.6		1106 **Hg** 0.9 0.5 -0.4	3270 **GGGa** 2.6 0.4 -2	3281 **SSkk** 2.7 0.6 -2.3		
	854 **TTb** 2.6 0 -1.6	1093 **Db** 1.1 0.5 -0.2	3028 **Xt** 1.4 0.3 -0.7		3279 **Vr** 1.5 0.4 -2	3302 **Aav** 1.2 0.7 -1		
	1057 **Ws** 1.4 0 -0.4	1094 **RRjj** 2.7 0.2 -1.9	3037 **Ga** 1 0 -0.3			3320 **Ji** 0.9 0.6 -0.5		
	1066 **Ff** 1 0 0		3271 **Qm** 1.5 0.3 -0.8			9836 **So** 1.4 0.7 -1.1		
1081 **HHHpp** 2.7 -0.3 -1.4	1084 **Rn** 1.5 0 -0.5		3280 **Cc** 1.1 0.3 -0.4			9854 **Ee** 1 0.6 -0.6		
1102 **OOhh** 2.9 -0.2 -1.7			1084 **NNgg** 2.9 0.3 -2.2					
3297 **FFFa** 3 -1 -1	3306 **Tp** 1.6 0 -0.6	9841 **Aa** 1.2 0.1 0.3						
3297 **IIIqq** 2.9 -0.5 -1.5	3307 **Bb** 1.2 0 -0.2							
3298 **Pl** 1.7 -0.2 -0.5	3308 **QQa** 3 0 -2							
3310 **MMff** 3.1 -0.4 -1.8	9823 **Ok** 1.7 0 -0.7							
9842 **LLeee** 3.3 -0.2 -2.1								

Page 1

Figure 5.5: Periodic Table of Elements for normalised 3x3 matrixes

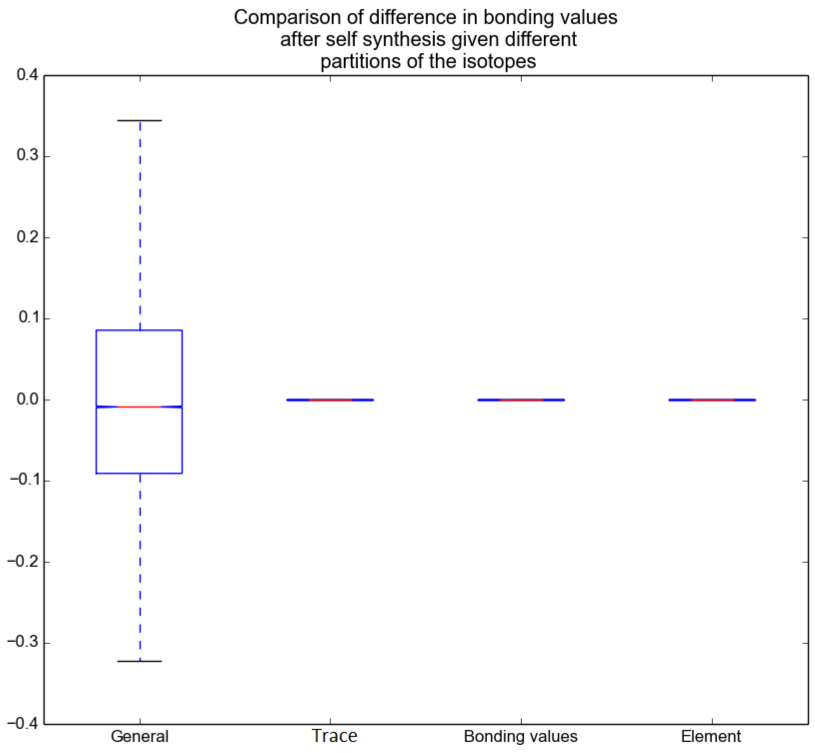


Figure 5.6: Comparison of difference in linking values in different partitions of self-synthesis composites

5.7 Decomposition Process

The decomposition process in JA AChem, expanded in Figure 5.7, attempts to break the different bonds in a JA AChem composite starting with the weakest bond. This was done to simplify matters, we only want to break a single bond so we test them starting with the one most likely to break. Since the structure of a JA AChem bond is important the decomposition of one of these composite is nontrivial. We have marked out the different parts of this process in terms of the transition functions they perform in the macro decomposition node. We step through the graph in the following sections.

5.7.1 read()

Uses two observer nodes to extract the link structure of the particle and the strengths of the links.

5.7.2 check()

Chooses acts based on whether the min of the strengths is greater than a uniformly distributed random number in [0:1]. If this is greater it exits the node. Otherwise we continue.

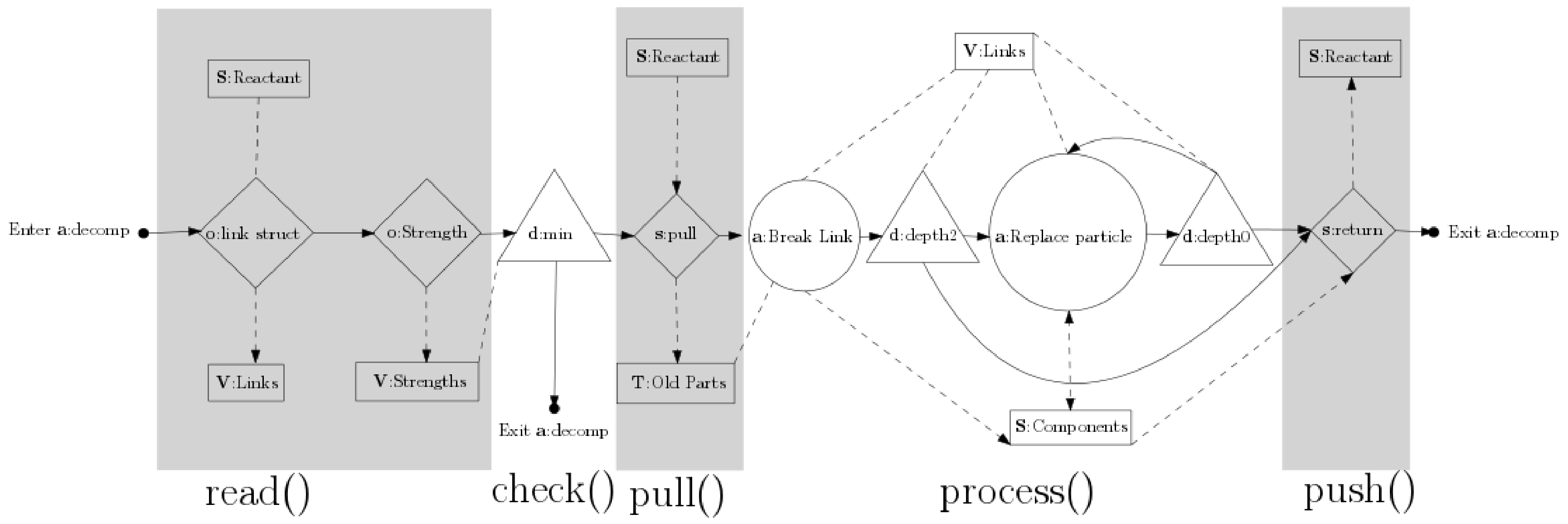


Figure 5.7: Micro level description of decomposition in JA AChem, shows loop used to correct hanging links caused by decomposition in process function

5.7.3 pull()

As in link, we pull our reactants into a local tank, to be disposed of at the end of the transfer functions.

5.7.4 process()

This is the most complicated section of our system. We start with the most basic part. The first action node breaks our weakest link and removes it from the list of links. We add the components of the link to the S:components node.

Our second node check in our links to see if there is a particle whose link is missing as the component of another link. If there is we then have to work out any need for the remaining structure to fix itself. If there is this is done by the following node and decision. The conditions for a structure to fix is described in the next section. Otherwise if the decomposition is finished and there are no "hanging" links we move to the push() function.

If we do have hanging links we replace the missing particle with the first siblings we find. This means that if there are other siblings in the link we form a different new link with just the siblings to create a replacement particle. If there is only one sibling then that sibling is considered the replacement particle. We then remove the hanging link from the list of links and store the replacement particle in sample.

Finally we check again for hanging links. If we have any then we loop back to replace the missing particle again with the replacement particle from this point forward. As long as there is a replacement particle it is used for this process and swapped out for the result.

If any of these particles are completely unable to form (the particle would be invalid) we place the components we would use to form them into the S:components sample.

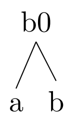
Once this loop finds no hanging links we continue to the push() function.

5.7.5 push()

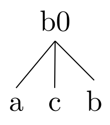
Our push function returns the content of both components and the replacement samples to the S:reactants container. We then clear our local containers and exit the node.

5.8 Resolving "hanging" link

When we break internal links we have to occasionally form new links. This is dictated by the following structure change rules, in these rules we denote the link being broken by bn where n is the depth of the link in the structure. In the following diagrams we will list the possible situtations when breaking a link and the resultant products. The initial graph before the link is broken is given on the left. The resultant products are given on the right. In all cases we have labelled, bn, the bond we are breaking. We use lower case letters here to denote atoms and upper case to denote arbitrary particles (composites or atoms). There are 5 different sorts of situtations to consider here (1) single link of 2 atoms, (2) single link of m atoms, (3) single link of particles, (4) one level nested link of particles, (5) two level nested link of particles.

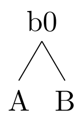
1. 

$\rightarrow$ (b0) a b

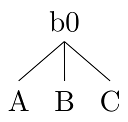
2. 

$\rightarrow$ (b0) a b c

3.

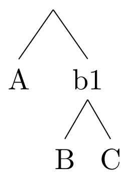
3.(a) 

$\rightarrow$ (b0) A B

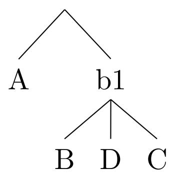
3.(b) 

$\rightarrow$ (b0) A B C

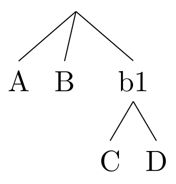
4.

4.(a) 

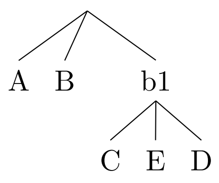
$\rightarrow$ (b1) A B C

4.(b) 

$\rightarrow$ (b1) A B C D

4.(c) 

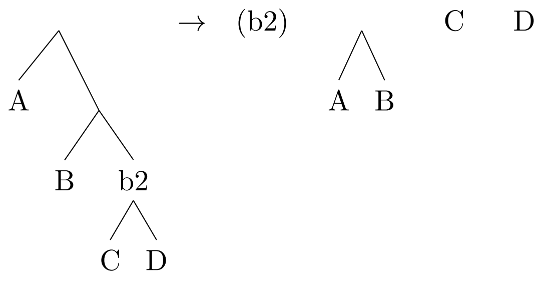
$\rightarrow$ (b1) A B C D

4.(d) 

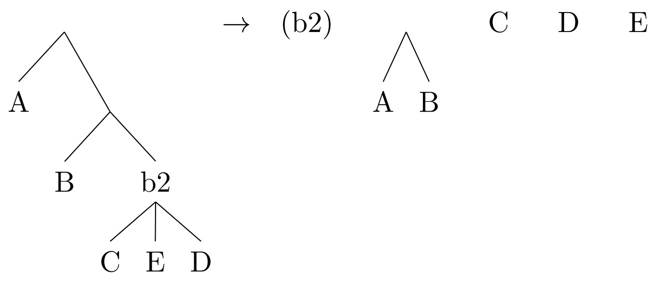
$\rightarrow$ (b1) A B C D E

5.

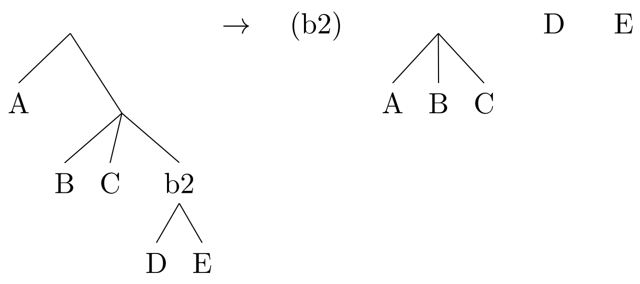
5.(a)


A
B
b2
C
D
→ (b2)
A
B
C
D

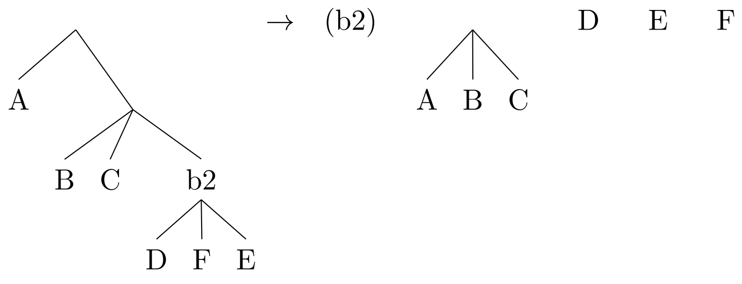
5.(b)


A
B
b2
C
E
D
→ (b2)
A
B
C
D
E

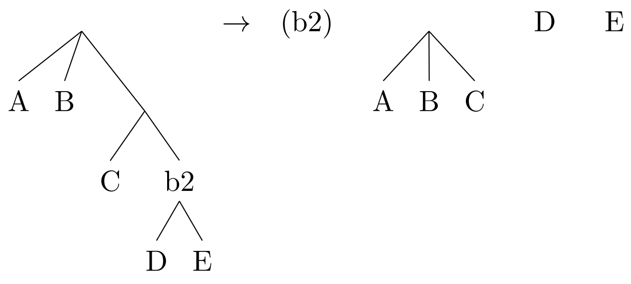
5.(c)


A
B
C
b2
D
E
→ (b2)
A
B
C
D
E

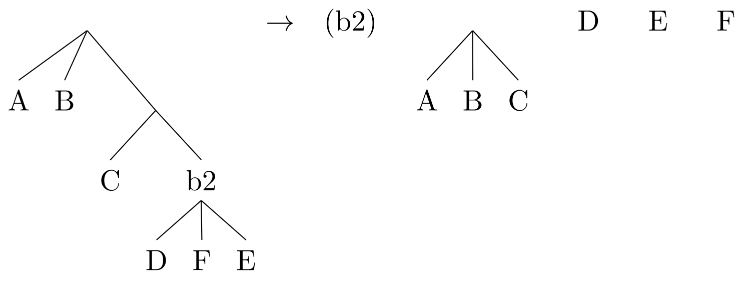
5.(d)


A
B
C
b2
D
F
E
→ (b2)
A
B
C
D
E
F

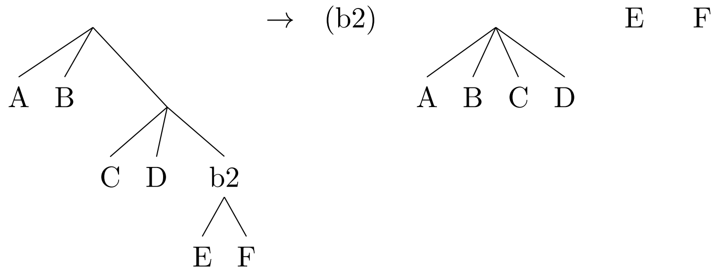
5.(e)


A
B
C
b2
D
E
→ (b2)
A
B
C
D
E

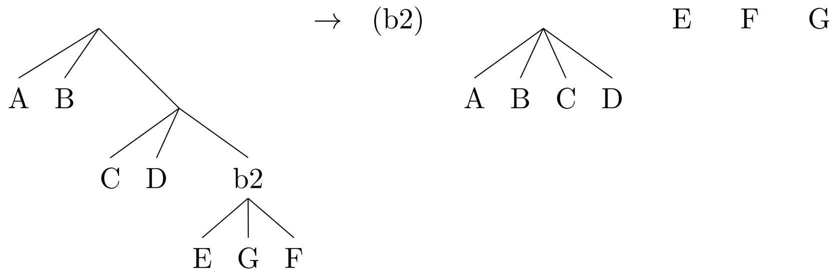
5.(f)



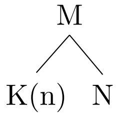
5.(g)



5.(h)

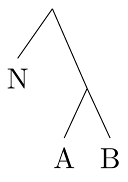


We can apply these rules to larger composites by considering each of the starting composites in rule 5. Call them K(a), K(b),...,K(h) and make them a subcomposite of a larger composite M, we call the other subcomposite N:

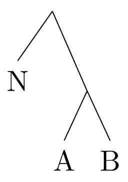


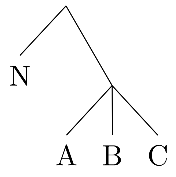
Then decomposition occurs as before but the following new composite is formed in each case by taking the result given the composite result in (x) above and using it to replace K(x) in M to give a new composite, M^*:

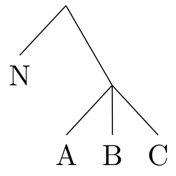
(a)

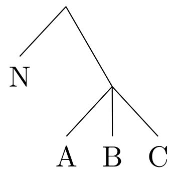


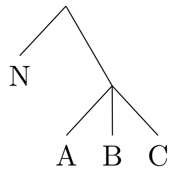
(b)

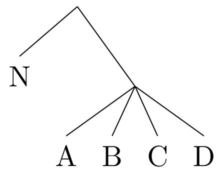


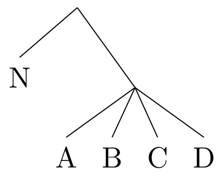
(c)


(d)


(e)


(f)


(g)


(h)


This can then be applied iteratively to allow for a link of any depth to be broken by letting M be K(x) in the larger composite and repeating the process replacing K(x) with M^*. In our system this is applied by our loop including **a:**replace part.

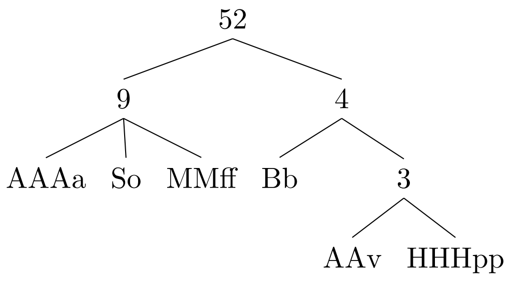
Decomposition and the forming of new composites caused by it occurs in the same manner regardless of the position of the link in any larger links. This means that while all examples here use at most a 3-tad link and all nested links occur on the far right position the concepts are the same. In links nested two deep all remaining components of the previous one deep link are used to form a new link at zero deep along with any remaining zero deep components. This means we have a decomposition algorithm now which will work on any composite of any size.

This replacement obviously causes the replacement of all composites above it such that the whole composite changes and is replaced by M*. During this process we do not change the eigenvalue indices used in each link, these same indices used with new components and no probabilistic checking during link formation means that decomposition can allow the forming of extremely weak links that will easily lead to further decomposition. In general as we use the strongest linking indices in linking decomposition will on average weaken the links in a composite. Sometimes though it will cause the creation of stronger links with composites that would not otherwise exist.

5.8.1 Example

For this section we show each possible decomposition of a randomly generated composite (generated without decomposition) and the strength of the link before decomposition.

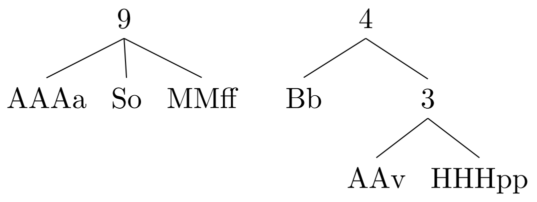
The particle we use for this was in this run the 52 generated composite:



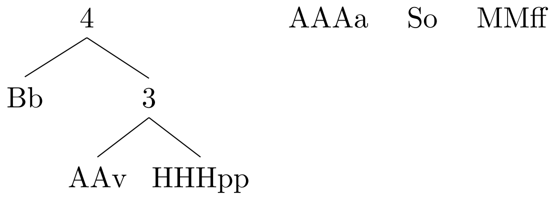
For the associated matrices for the atoms in 52 see Appendix.

We then can list the result of decomposing the composite at the bond forming each of our subcomponents: 3, 4, 9 and 52. We list these bellow with the number of the composite that resulted from the bond, the strength of the bond and then the resultant particles from the decomposition:

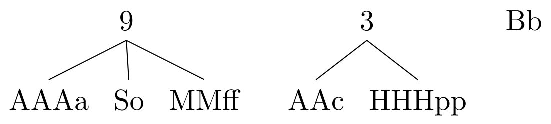
52: Strength -¿ 0.3918



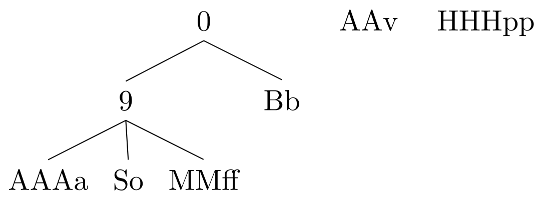
9: Strength -¿ 0.3124



4: Strength -¿ 0.3484



3: Strength -¿ 0.3669



In general the decomposition of this composite proceeds in a normal fashion with composites simply breaking apart except in the last case of composite 3 breaking. This leads to the left over Bb atoms maintaining a connection to the 9 composite to form a new composite, here called 0.

5.9 Conclusion

We have used a rigorous algebraic approach to define an artificial chemistry capable of both constructive and destructive behaviour. The algebraic structure of our particles has provided us with an equivalence of atoms giving us a period table based on the properties of our atoms within larger particles. Our algebraic links have enabled greater combinatorics in particles with the same atoms. Our destructive behaviours have allowed otherwise unseen or highly unlikely particles to form.

These properties of our chemistry are all identifiable in particular elements of our graph description of it. The particles themselves, the linking node and the decomposition node. This chapter contains no systematic numeric simulation results. These are presented in the following chapter. There are other aspects of our graph description that have not been considered here such as our choice of probability spawning function used in both linking and decomposition action nodes. In the following chapter we explore what effects this has on our system. In this case we do so through observation (top-down) rather than through axioms (bottom-up).

Chapter 6

Modifying AChems with Probability Spawning Functions

We explore the potential of a non-deterministic system through probability spawning functions. We define this functions and provide a set of examples in our Jordan Algebra Chemistry. We vary these in an attempt to affect the emergent properties of the system. We then briefly consider combinations of these functions to make building complex probabilistic effects quicker to build.

Natural chemistry deals with non-deterministic processes, and this is reflected in some artificial chemistries. We can tune these artificial systems by manipulating the functions that define their probabilistic processes. In this work we consider different probabilistic functions for particle linking, applied to our Jordan Algebra Artificial Chemistry. We take five basic functions and their variations to investigate the possible behaviours of the system, and try to connect those behaviours to different traits of the functions. We find that, while some correlations can be seen, there are unexpected behaviours that we cannot account for in our current analysis. While we can set and manipulate the probabilities in our system, it is still complex and still displays emergent behaviour that we can not fully control.

6.1 Probabilistic AChems

We present an exploration of the space of and effect of 'probability spawning functions'(psfs) in artificial chemistries (AChems). We use Jordan Algebra AChem (JA AChem) as an example of the use of psfs.

Natural chemistry is a probabilistic process where environmental variables (such as temperature) can affect the probability of bonding. AChems are inspired, to a greater or lesser degree, by natural chemistry, but there are many probabilistic attributes of natural chemistries that are often made deterministic in artificial chemistries, such as some aspects of movement, linking, and decomposition of links. This ignores a key feature of natural chemistry that could help our AChems to exhibit more complex behaviour.

Different AChems take different approaches in terms of determinism. Some systems, such as Hutton (2002), always link particles that encounter each other and match a linking rule. Young and Neshatian (2015) investigate different approaches by which reactants are chosen for linking, but the linking is then deterministic. There are a few AChems that have probabilistic processes, such as the selection of parameter sets in a recipe in SwarmChem (Sayama, 2009), or probabilistic movement in the 2D membrane AChem (Ono and Ikegami, 2001).

Here we focus on our Jordan Algebra Artificial Chemistry, JA AChem (Chapter 5, (Faulkner et al., 2016)), which is a *subsymbolic* AChem, one where various particle and linking properties emerge from the underlying structure of the particles (Faulkner et al., 2017). Subsymbolic AChems may include stochastic processes, such as stochastic decay in Stringmol (Hickinbotham et al., 2010; Clark et al., 2017), but others are deterministic, for example RBN world (Faulconbridge et al., 2010, 2009) and Spiky-RBN (Krastev et al., 2016, 2017).

6.2 Jordan Algebra AChems

JA AChem used the matrices' eigenvalues and vectors to define a few probabilistically driven processes, including particle linking and decomposition. In Faulkner et al. (2016), and the previous chapter, the design of these probabilistic processes is chosen in a somewhat arbitrary manner. Here we explore how different choices for these processes may be exploited to "tune" our system in different ways and towards different behaviours, while still keeping an open mind about systemic properties.

6.2.1 Probability Spawning Functions Options

There are many psfs we can use to determine linking probability. For example, a link would always be formed using the constant function $c(x) = 1$.

There are many aspects of such functions that we can consider. Here we focus on three: total area (under function between 0 and 3), position of peak, and size of tail. To help us identify effects on the system caused by each of these aspects, we start with a basic set of functions that cover a variety of different possibilities for each of these. Our basic functions are shown in Figure 6.1.

Consider the uniform function (Figure 6.1a):

$$u(x) = 0.15 \tag{6.1}$$

Here we have an infinite area overall, no peak, and a very large tail. All links, regardless of the eigenvalues, have a probability of 0.15. This particular value is chosen to give it a similar

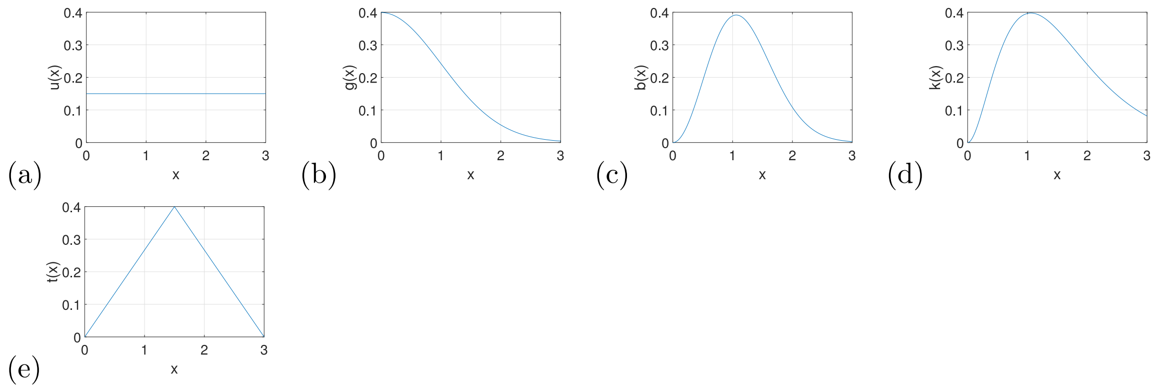


Figure 6.1: Plots of the five basic functions used in JA AChem. (a) uniform $u(x)$; (b) Gaussian $g(x)$; (c) Maxwell-Boltzmann $b(x)$; (d) Energy Maxwell-Boltzmann $k(x)$; (e) triangle $t(x)$.

area as the other basic functions within our main zone of interest. The other functions that extend to $+\inf$ all converge to 0. The zone we are mainly interested in here is $x \in [0, 3]$.

Consider the Gaussian distribution (Figure 6.1b), as used in chapter 5 ($s_{A_iB_j}$) and Faulkner et al. (2016):

$$g_\sigma(x) = \frac{1}{\sqrt{2\sigma^2\pi}} \exp \frac{-x^2}{2\sigma^2} \tag{6.2}$$

Here we take $\sigma = 1$, so use $g = g_1$. Our prior use of the unit standard deviation Gaussian explains the choice of constants in the other functions here, to roughly match area and cutoff. This function has a peak at zero, so near equal eigenvalues have the highest probability of linking. It has a slightly larger area in our zone of interest, $x \in [0, 3]$, than does u. It has a long but quickly diminishing tail, so eigenvalue pairs with large differences can link, but with a quickly decreasing probability.

Consider the Maxwell-Boltzmann velocity distribution: (Figure 6.1c):

$$b_a(x) = \sqrt{\frac{2}{\pi}} \frac{4x^2}{a^3} \exp \frac{-4x^2}{2a^2} \tag{6.3}$$

Here we use $b = b_{1.5}$. This has a similar area to g, but has a shifted peak. So linking is more likely with pairs of eigenvalues that are similar but not the same.

Consider the alternative Maxwell-Boltzmann distribution, $k = k_{0.23}$; this has a different tail shape due to the change to the exponential. This requires different variables to match the peak height and rough area of the other base functions. (Figure 6.1(d)):

$$k_a(x) = \sqrt{\frac{2}{\pi}} \frac{0.04x^2}{a^3} \exp \frac{-0.2x}{2a^2} \tag{6.4}$$

This has a fatter tail, to see the effect of a large tail and a shifted peak.

Finally, consider the triangle function, which we use to examine the separate effects of long tails and non-zero peaks (Figure 6.1e):

$$t(x) = \begin{cases} \frac{4}{15}x & : 0 < x \leq \frac{3}{2} \\ \frac{4}{15}(3-x) & : \frac{3}{2} < x \leq 3 \\ 0 & : 3 < x \end{cases} \tag{6.5}$$

This has a shifted peak at a similar position and height to b, but has no tail. We use it to assess the effects of long tails. It is also the only function in this set whose slope is discontinuous, which may allow us to assess the appropriateness of functions with discontinuous derivatives for our purposes.

6.2.2 Psfs in linking in Jordan Algebra AChem

We test these psfs g, b, u, k, and t in our JA AChem (Faulkner et al., 2016), to investigate how changes in peak and tail affect the behaviour of our system. Three probabilities contribute to the overall probability of linking in JA AChem: *Xcoll*, a and s (and p which is a combination of others).

The initial occurrence of a probability is when we select the list of reactants for our link. We randomly sample with replacement from our well-mixed tank, using a uniform distribution. After we sample the first two components, we have a probability for sampling further reactants, to produce an n-tad Jordan product link. We continue to sample, for one reactant at a time, with success probability *Xcoll*, until we fail. This gives a small but non-zero probability for attempting links with more than two reactants. Here we consider this to be part of the sampling of how our tank functions, rather than an aspect of the linking node, and we set $Xcoll = 0.2$.

If we have more than two reactants in our link attempt, we take the reactants in sampling order, and consider the linking probabilities between each neighbouring pair of reactants. The *minimum* of these is taken as the linking probability of the overall link. So the probability of success depends on the probability of the weakest link forming. For four reactants, this is:

$$p(A, B, C, D) = \min\{p(A, B), p(B, C), p(C, D)\} \tag{6.6}$$

This linking probability p comprises two probability terms, combined here, and used separately elsewhere. We relate these to two different analogies with natural chemistry. We have the strength or potential strength, s, of the link, and the relative orientation, a, of the particles.

The orientation probability is defined as:

$$a(A, B) = 1 - \frac{(\mathbf{v}_a \cdot \mathbf{v}_b) + 1}{2} \tag{6.7}$$

where $\mathbf{v}_a \cdot \mathbf{v}_b$ is the scalar (dot) product of the real parts of the two unit eigenvectors, this is again not taking into account sign so it matches with alignment. When the vectors are parallel, $a = 0$; when they are anti-parallel, $a = 1$.

This orientation probability is used in two ways. It first defines which pair of eigenvalues are used to calculate link strength. We select the pair of eigenvalues e_a, e_b (where e_a is one of the three eigenvalues of matrix A) such that their corresponding eigenvectors $\mathbf{v}_a, \mathbf{v}_b$ maximise a. We can interpret this as the best aligned set of eigenvalues. The orientation probability also contributes to the probability of the link actually occurring.

Once we have used a to select e_a and e_b, we calculate $x = |e_a - e_b|$. This value of x is used to calculate $s(A, B) = s(x)$, the strength of the link that the particles would form using these eigenvalues. As well as its part in forming links, s is also the strength of the link once formed, and is the probability of the link *not* decomposing on a decomposition attempt.

We define the probability of the link occurring to be:

$$p(A, B) = \max\{a(A, B), s(A, B)\} \tag{6.8}$$

So linking happens if the link will be strong or if the particles are particularly well aligned.

Here we compare the effect on the system's behaviour when we use each of the above functions, u, g, b, k, t, as s. For each function, we run 5 rounds of linking and decomposition phases. We initialise the tank with 69 atomic particles. A linking phase performs a number of linking attempts equal to the number of particles in the tank at the beginning of the phase. For each linking attempt, we select a sample of particles from the tank. If the attempt is successful, and if the resulting particle is a novel particle, it is added to the tank. The reactants are not removed, allowing them to take part in further reactions. Hence the tank contains one of each kind of atom, and one of each kind of composite particle made. This particular system does not have mass conservation, as here the "tank" is just the collection of possible particles found by the system so far. We have no interest here in the frequency with which any particle is formed; our focus is on finding novel particles so only these are added to the tank.

6.2.3 Psfs in decomposition in Jordan Algebra AChem

A decomposition phase performs a decomposition attempt on particle in the tank. If this leads to any new particles, they are added to the end of the tank and we attempt to decompose them in this phase. (We do not perform a decomposition phase after our final linking phase due to computational memory limitations.)

Function	Area	Particles
u	0.45	1331
g	0.49865	1796
b	0.499433	1938
k	0.715772	1852
t	0.6	1900

Table 6.1: The area under each of the probability spawning functions (integrated between 0 and 3), and the number of unique particles produced on running algorithm for 200 generations including the 69 atoms.

6.3 Effects of psfs on JA AChem

6.3.1 Known effect: Speed and Area

The area under the psf directly relates to the general probability of any particular link occurring. The larger the area the more links will occur and the stronger they will be. We can see the area then as a sense of 'speed' in this system. If we do not change the shape of the function but simply decrease its area, then we should see roughly the same behaviour. It would be slowed down, as fewer links will form and decompose per generation. Due to this we try to keep the area of our functions reasonably similar with in the zone of interest (here taken to be $x \in [0, 3]$). We do not keep the areas exactly the same, preferring to have the peak height, shape and position more equal across functions, e.g. matching triangle peak height and span in t to peak height and span of zone of interest in b.

From the areas shown in Table 6.1 we can see that the functions' varying areas have similar numbers of particles produced in each system, allow for a sensible comparison of results. It is important to note that Table 6.1 gives the area in our "zone of interest", $x \in [0, 3]$. The overall area of u is much larger than that of the other functions as it is constant along the entire x axis while the other functions all tend to 0.

6.3.2 Looking for unknown effects

In this section we look at the influence of these functions on different properties of the particles in our system. While we can collect data on many properties (such as total number of atoms in particles, number of distinct atoms in particles, . . .), we focus here on three properties that show the most variation across the shape of the functions used. Other properties vary primarily with the size of the system. Systems that generate more particles tend to having bigger particles (ones with more atoms in them).

We provide graphical descriptions of these properties in the form of whiskered boxplots overlaid (except in the case of strength which is on a logarithmic scale) with violin plots. We use violin plots, so called for their shape, to show the distribution of density of points with in the boxplots so we can see the modality of the distributions as well as spread and normality.

The violin plot is shaped to be wider where there are more points in the distribution therefore showing if the results are clustered centrally or in multiple groupings or if the values are uniformly distributed. We also make use of the Vargha-Delaney A test (Vargha and Delaney, 2000) which determines the effect size for the difference in two distributions medians. This is done using correction for the size of the original data sets. The A value gives a small effect for values greater than 0.56, medium effects are greater than 0.64 and large effects are greater than 0.71.

The three properties reported here are:

1. **Largest Link:** The largest number of particles involved in a single link within a particle (a link is formed by an n-tad Jordan product involving n particles).

2. **Strength:** The product of the strengths of each link in the composite particle:

$$P_s = \sum_{l \in P} l_s \tag{6.9}$$

   Atomic particles have no strength as they have no link, and are excluded from the strength statistics

3. **Self-synthesis:** The number of times in each particle there is a link linking two particles (atoms or composites) of the same kind

We consider how two features of our psfs may have affected each of these properties in our system:

(a) **Peak position:** In our base functions there are three different sorts of peak positions: none (u), zero (g) and non-zero (b,k and t).

(b) **Tail size/length:** We have four different sorts of tail in our functions: constant large (u), constant zero (t), small (g, b, $O(\exp -x^2)$) and large (k, $O(\exp -x)$).

In comparing these functions we are looking for how their features may be contributing to "interesting" distributions of particle properties. Here we are looking for wide distributions of values, preferably with outliers and a wide interquartile range. There should be a strong sense of median or common particle properties, but there should also be outliers and less common particles that stretch over a large range, indicating a rich variation in particles produced.

6.3.2.1 Largest Link

We start with how our functions affect the ability of our system to form larger (and possibly more complex) particles by looking at the largest links (Figure 6.2(a)).

Given a null hypothesis that psf has no effect on the number of particles in the largest link, we calculate the p-values, using the ranksum test, and the effect size, using the A

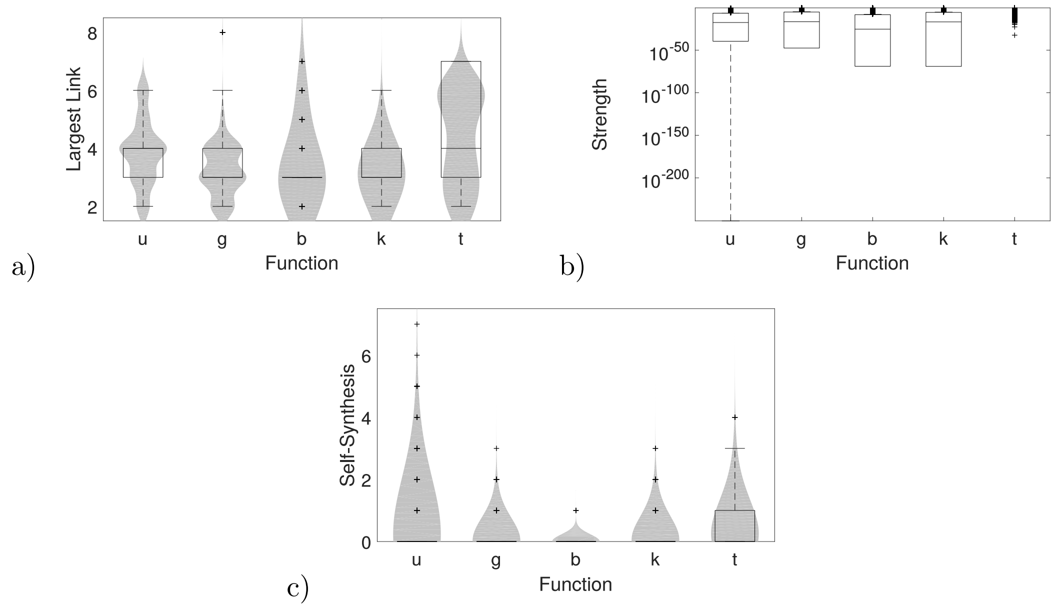


Figure 6.2: How particle properties vary with probability spawning function: (a) largest number of components in a link in each particle; (b) log-strength of the particles; (c) number of occurrences of self-synthesis in each particle.

	g		*b*		*t*		*k*	
u	**0**	**0.64**	**0**	**0.64**	**0**	0.58	**0**	**0.67**
g	-	-	0.07	-	**0**	**0.67**	**0.003**	0.53
b	-	-	-	-	**0**	**0.65**	0.835	-
t	-	-	-	-	-	-	**0**	**0.69**

Table 6.2: p-values (left column) and effect sizes (right column) for size of largest link. Statistically significant p values ($p < 0.05$) are shown in bold; effect size A values are calculated for these; medium or larger effect sizes ($0.64 \leq A$) are shown in bold.

test (Vargha and Delaney, 2000), between the tanks produced with each function (Table 6.2). These results split our functions into two groups whose distributions only have small differences: u and t; and g, b and k

For g, b and k there is very little spread and almost all particles have a maximum of 4 components in a link, but a few have as many as 7 or 8. Looking at u and t we see they have larger interquartile ranges, but spanning a different range of values: half of the particles produced with u have a largest link size of 3–4, and half of those produced with t have 4–6.

These groupings do match any particular peak position but do group to separate our constant and non-constant tails.

6.3.2.2 Strength

Since many of our particles have very small strength we use a log scale to make the distributions clearer (Figure 6.2b). All of u, g, b and k have skewed distribution, with the largest range being with k.

Notably t has a near non-existent distribution. Most of the particles in the t system have zero strength, and so a probability of 1 of decomposing. This is due to the constant zero tail. The selection of eigenvalues is based on alignment (see earlier), not on the similarity in value. This means links can occur with large differences in values, that is, large values of x. This results in low strength with long tails, but zero strength with no tail. So the tails on our distributions are important for being able to generate stable particles.

There is no indication that peak position has any influence on strength in the system.

6.3.2.3 Self-synthesis

There may be an advantage to a lack of tail. All the distributions have at least one instance of self-synthesis, but a median of zero (Figure 6.2c). The only function that produces significant self-synthesis is t.

The u, g and k functions also have at least one instance of a particle with more than one link with self-synthesis.

Again there is no connection to peak position, but there is a strong indication that having no tail affects this property.

6.4 Variations of functions for tuning and testing effects

In order to further investigate the above results we need a larger number of functions with different features. We now further test the effect of constant zero tails, and of tail size in general. We look at four sets of variations on our existing functions:

1. b: $b_{1.5}$ (original), and b_1, b_2. We also abuse notation with b_c the peicewise function formed of $b_{1.5}$ cut off at 3 with equation:

$$b_c = \begin{cases} b_{1.5} & x < 3 \\ 0 & \text{otherwise} \end{cases} \tag{6.10}$$

2. k: $k_{0.23}$ (original), and $k_{0.13}, k_{0.33}$. We also abuse notation with k_c the peicewise function formed of $k_{0.23}$ cut off at 3 with equation:

$$k_c = \begin{cases} k_{0.23} & x < 3 \\ 0 & \text{otherwise} \end{cases} \tag{6.11}$$

3. g: g_1 (original), and $g_{0.5}, g_{1.5}$

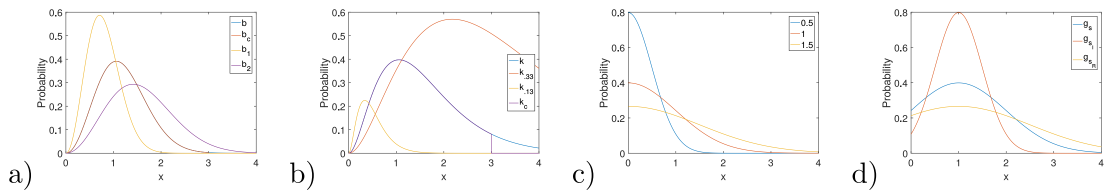


Figure 6.3: Sets of variations of base functions: (a) four variants of b; (b) four variants of k; (c) three variants of g; (d) four variants of g based of on the shifted peak variant: g_s.

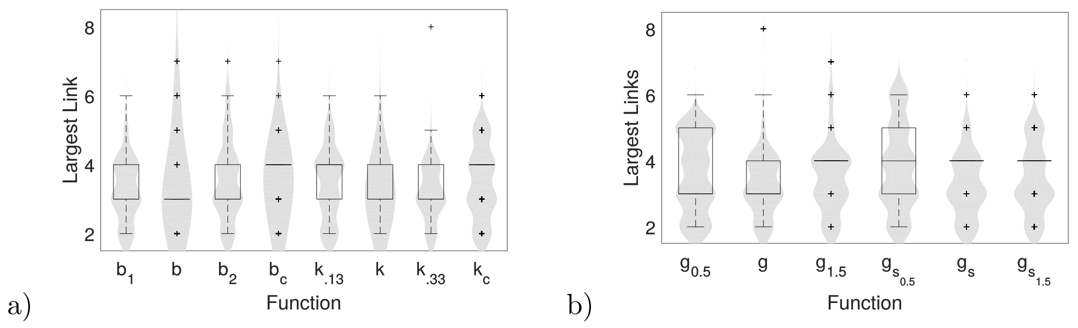


Figure 6.4: Distribution of largest link sizes with: (a) variants of b and k; (b) variants of g

4. shifted gaussian, gs: $gs_{0.5}$, gs_1, $gs_{1.5}$, where

$$gs_\sigma = \frac{1}{\sqrt{2\sigma^2\pi}} \exp \frac{(x-1)^2}{2\sigma^2} \tag{6.12}$$

The areas under these curves are not particularly similar, however any area effects caused by this difference should be identifiable across all four sets of variations so normalisation is not required. By allowing the areas to change we also allow ourselves to investigate if there are any area effects not previously noted.

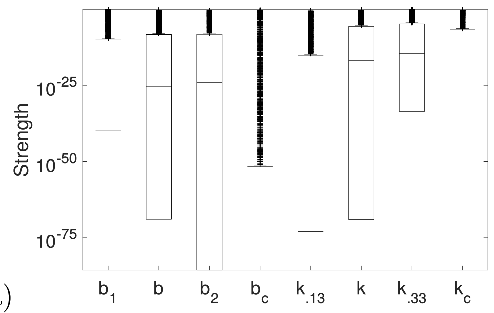
6.4.1 Largest Link

Although the results in Figure 6.4 show statistically significant differences ($p < 0.05$), these all have small effect size ($A < 0.61$).

So the size of the largest link in each particle is not affected by the area under the curve, its steepness, the position and height of its peak or the size of its tail. This means there must be some other property involved that effects the size of the largest links. The largest effect size between any pair of variations is between gs and $gs_{0.5}$, with $A = 0.61$.

6.4.2 Strength

The results shown in Figure 6.5 show near non-existence of variation in the strength values of both b_c, with the cut-off, and b_1, the larger area b function. This agrees with the previous result: b_c has no tail, and, because of the increased steepness of the curve in b_1, it has much

a)


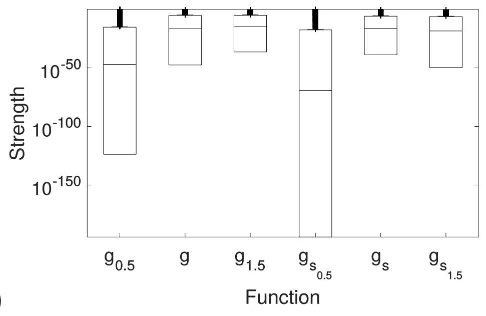
b)


Figure 6.5: Strengths of particles for: (a) variants of b and k, (b) variants of g

less of a tail than the other functions.

We can also see that b_2, with its larger tail, has a larger range of strengths than the base b functions. This further supports the idea that a longer tail results in a wider variety of link strengths.

When we look at the k variants we see k_c and $k_{.13}$, the smaller of the functions, both have low variation in strength. However $k_{.33}$ has a larger tail than k—much like b_2 and b—but a smaller range of strengths, suggesting the spread of strengths is not simply correlated with tail size.

Our g variants further break our established pattern. $g_{0.5}$ has the smallest tail of the g variants, but the largest range of values. $g_{1.5}$ with the largest tail has the smallest range of strength values. This is contrary to our results look at just the base functions but agrees with our k results. The b and g functions have similar tails, so clearly something other than the tail has a strong effect on strength.

The large range found on the $g_{0.5}$ strengths might be connected to the larger range of values the function has.$g_{0.5}$ has a higher peak than g and a larger range of strengths. However b_1 also has a higher peak but has a very limited range of values.

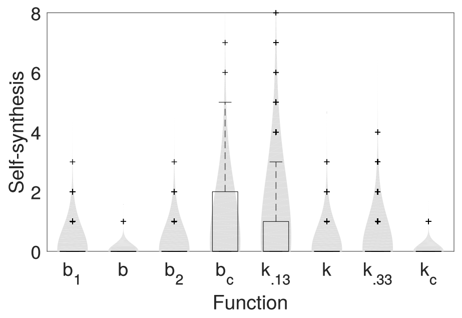
The strength distributions of our particles seem to be more complicated than a simple aspect of tails or peak position, though in some cases there is still a correlation.

6.4.3 Self-synthesis

The self-synthesis results are shown in Figure 6.6. These show another effect of the lack of tail: an increased rate of self-synthesis. The only variant of b with any range of self-synthesis is b_c, which also has far more outliers than in the other functions.

The k variations seem to be anomolous until we consider the overall shape of $k_{.13}$. This function has a very small area and a very long tail meaning that because it has a very small tail that starts early it acts more like a function with no tail than our other functions. On the otherhand k_c has a very little self-synthesis. The reasons for this are not clear but it is the only function with a high value before the cut-off which may be influencing this.

Parameter changes have very little effect on self-synthesis in g functions, Figure 6.6b.

a)


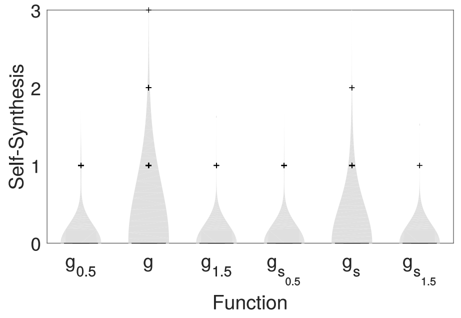
b)


Figure 6.6: Occurrences of Self-synthesis with: (a) variants of b and k, (b) variants of g

6.5 Combinations of psfs

In natural chemistry multiple factors influence the probability of different reactions. We already have a set of functions for our system that use the state of the system to yield a probability for some event or reaction. The system state space Σ can encompass anything from environmental variables, spatial positioning in the system, to the attributes of the particles involved, and more. We can develop probability spawning functions to reflect the effect of different parts of the system on the probability of linking.

For example, we concluded that to have strong links, requires the probability function to have a long tail, but that functions without tails exhibit a better amount of self-synthesis. These are both properties of the specific AChem (Faulkner et al., 2016) and the chosen probability functions. It would be a step towards more control of our systems if we could combine such probability functions that we know to produce certain effects in order to give us both behaviours, rather than have to hand-craft new functions. To do so we need a controlled way to combine our probability functions.

Following our philosophy of using tried-and-tested algebraic approaches for defining AChems (Faulkner et al., 2016) we use a semi-ring algebraic structure to do so. Here we define two operators that can be used to combine probability spawning functions, and demonstrate their effect on two different AChems, the Non-Constructive Explicit Chemistry (Dittrich et al., 2001), and our Jordan Algebra Artificial Chemistry (JA AChem) (Faulkner et al., 2016).

6.5.1 Semi-rings

A ring is an algebraic generalisation of integer addition and multiplication, with their distributive and associative properties. A semi-ring does not require the additive operator to have an inverse. Formally, a semi-ring $(R, +, \bullet)$, is a set R with two binary operators, referred to as the semi-ring's addition and multiplication, that satisfies the following axioms:

- **Addition forms a commutative monoid:**

$$\forall a, b \in R \;.\; a + b \in R \tag{6.13}$$

$$\forall a, b, c \in R \;.\; (a + b) + c = a + (b + c) \tag{6.14}$$

$$\exists 0 \in R \;.\; \forall a \in R \;.\; a + 0 = 0 + a = a \tag{6.15}$$

$$\forall a, b \in R \;.\; a + b = b + a \tag{6.16}$$

- **Multiplication forms a monoid:**

$$\forall a, b \in R \;.\; a \bullet b \in R \tag{6.17}$$

$$\forall a, b, c \in R \;.\; (a \bullet b) \bullet c = a \bullet (b \bullet c) \tag{6.18}$$

$$\exists 1 \in R \;.\; \forall a \in R \;.\; a \bullet 1 = 1 \bullet a = a \tag{6.19}$$

- **Annihilation:** 0 annihilates over $\bullet$

$$\forall a \in R \;.\; 0 \bullet a = a \bullet 0 = 0 \tag{6.20}$$

- **Distributivity:** $\bullet$ distributes over $+$

$$\forall a, b, c \in R \;.\; a \bullet (b + c) = (a \bullet b) + (a \bullet c) \tag{6.21}$$

6.5.2 Semi-ring of probability spawning functions

We take the set to be the set of probability functions $\mathscr{F}$. We need to find binary operators that combine two probability functions to produce a further function, that obey the semi-ring axioms. For usable probability functions we require our operators to have further desirable properties.

We use a semi-ring formalism to allow us to work with multiple operators in a structure that allows us to combine the results of our operators. While this would be possible without resorting to a semi-ring, we could not then reach conclusions about the resultant combined probability function, and could not simplify using associativity and commutativity, without case-by-case proofs.

Closure (eqns.6.13,6.17) means that our binary operators must produce a probability function, with values in the interval $[0, 1]$. This limits our options of binary operators; in particular it disallows normal addition. It does not disallow normal multiplication: $f(x) \bullet g(x) = f(x)g(x)$. However, we avoid it here: multiplication in this context rapidly decreases overall probability of linking the more it is used. This is not a desirable property in many cases, so we choose instead binary operators with less constrictive effects on our resulting probabilities.

Two simple choices for binary operators are min and max over functions. Both of these meet the requirements of our axioms. In these cases the identities are the constant functions:

$$Id_{\min} = Id_{\min}(x) = 1 \tag{6.22}$$

$$Id_{\max} = Id_{\max}(x) = 0 \tag{6.23}$$

Since our functions are limited to producing values between 0 and 1 the min functions Id with be constant as 1 as nothing can be larger than this meaning that taking the minimum for this function and any other will always given the value of the other function. Similarly for the max function there can not be a value lower than 0 so the max operator will only return the value of the Id function if the other function also equals 0.

Both these operators meet the requirements for the addition operator. They also meet the requirements for being each other's multiplication operator, as min and max distribute over each other, and their identities annihilate each other:

$$\max(Id_{\min}(x), f(x)) = \max(1, f(x)) = 1 = Id_{\min} \tag{6.24}$$

$$\min(Id_{\max}(x), f(x)) = \min(0, f(x)) = 0 = Id_{\max} \tag{6.25}$$

Proof of distribution is not provided here; it requires case by case evaluation of the different ordering of the function values (e.g. $f(x) < g(x) < h(x)$); a version of it can be found in ProofWiki (2011).

These two functions form a semi-ring over the probability spawning functions. We can use combinations of probability functions in many ways in different artificial chemistries.

6.5.3 Example

As an example of how these semi-rings can be used, we consider the Non-constructive Explicit Chemistry (Dittrich et al., 2001). This system has deterministic reactions. If two particles are chosen for which there is a reaction rule in the system, then the reaction takes place. The set of particles of the system are $\{A, B\}$. The reaction rules for this system are:

$$\begin{aligned} r_1 :& A + A \to A + A + B \\ r_2 :& A + B \to A + B + B \\ r_3 :& B + A \to B + A + B \\ r_4 :& B + B \to B + B + A \end{aligned}$$

The newly produced particle randomly replaces an existing particle in the tank, so the total number of particles is conserved. We start in all cases with an equal number of A and B particles, 500 of each.

In this AChem, the level of B increases (as 3 of the 4 rules add a B) and then stabilises

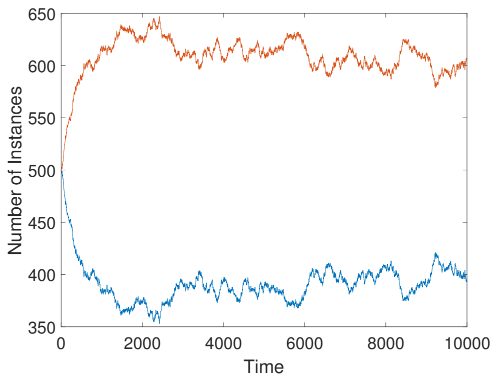


Figure 6.7: Original behaviour of Non-Constructive Explicit Chemistry (Dittrich et al., 2001)

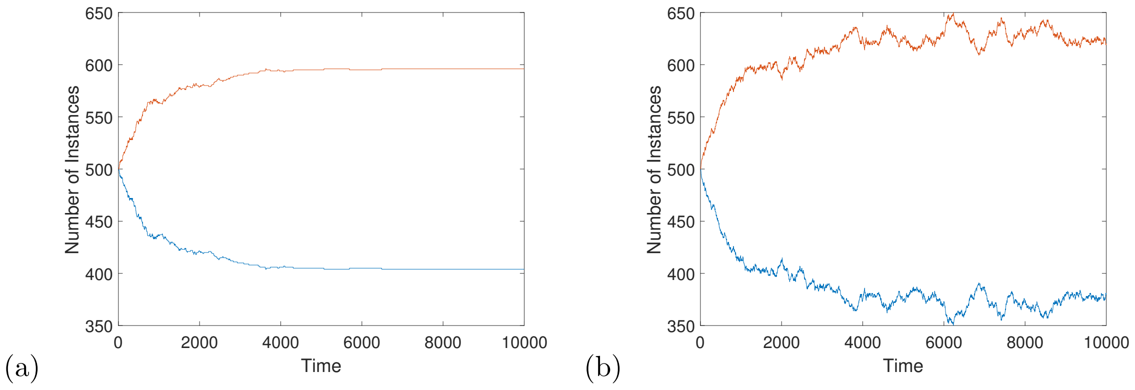


Figure 6.8: Behaviour of Non-Constructive Explicit Chemistry with (a) p_t; and (b) p_c

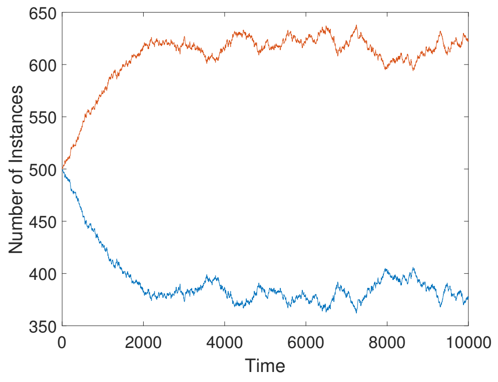
and holds steady (with some noise). We can see the behaviour in this system in Figure 6.7. This stabilises completely over time. It can be described by differential equations, meaning we know from the start where and how this system is going.

We now perturb and change the behaviour of the system without changing the linking rules, just their probabilities. To do this we include a probability of the reaction happening based on two environmental variables: time (t) and concentration of the particles c_B (concentration of B). We have time measured in reactions and a time decay type probability function. We also have the concentration of B particles in the system.

$$p_t(t, c_B) = e^{-t/1000} \tag{6.26}$$

$$p_c(t, c_B) = c_B \tag{6.27}$$

Using these two functions does change and perturb the system, Figure 6.8. Using p_t causes the system to stabilise more, and using p_c slows the upward trend of B. However, neither of these systems are very different from the original system, though p_t does change the stabilisation point for B: with the original p_c, B tends towards 650 while with p_t it stabilises closer to 600.

a) 

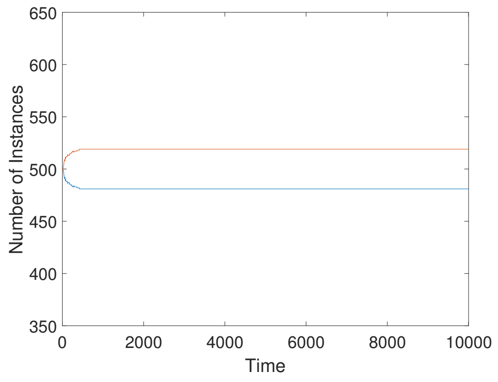
b) 

Figure 6.9: Behaviour of Non-Constructive Explicit Chemistry with (a) $max(p_t, p_c)$ and (b) $min(p_t, p_c)$

We could experiment with other functions and see what they do to change this system. However, with the semi-ring approach described earlier, we can instead combine these existing functions to see if a combination of them changes the behaviour in interesting ways. And we do not need to prove that the resulting functions are probability functions: they are by construction. Our two combined functions, using our operators, are:

$$\max(p_t, p_c) \tag{6.28}$$

$$\min(p_t, p_c) \tag{6.29}$$

These give us different behaviours from the uncombined functions, Figure 6.9. The max function performs similarly to our original functions, and stabilises a little faster than the original system. Not much happens in the min system: it stabilises much closer to the original 500 than to the 650 that most of the other systems head towards.

6.5.4 Combinations in JA AChem

The Non-Constructive Explicit Chemistry is too simplistic to tune much further than this as there is a limit to the possible behaviours of a system with this particular rule set. Thus we now move to testing this method of tailoring AChem behaviour on the strength versus self-synthesis problem of Jordan Algebra AChem.

We use two of the probability functions investigated in Faulkner et al. (2017).

$$b(x) = \sqrt{\frac{2}{\pi}} \frac{4x^2 \exp \frac{-4x^2}{2a^2}}{a^3} \tag{6.30}$$

$$t(x) = \begin{cases} \frac{4}{15}x & : 0 < x \leq \frac{3}{2} \\ \frac{4}{15}(3 - x) & : \frac{3}{2} < x \leq 3 \\ 0 & : 3 < x \end{cases} \tag{6.31}$$

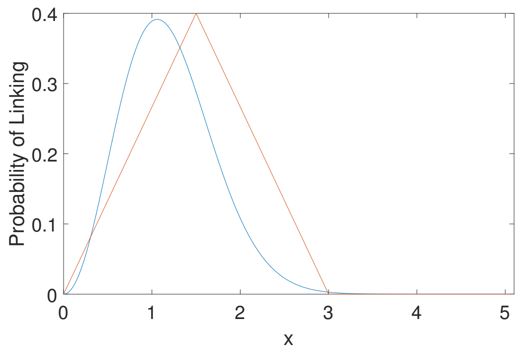


Figure 6.10: Base functions from Faulkner et al. (2017)

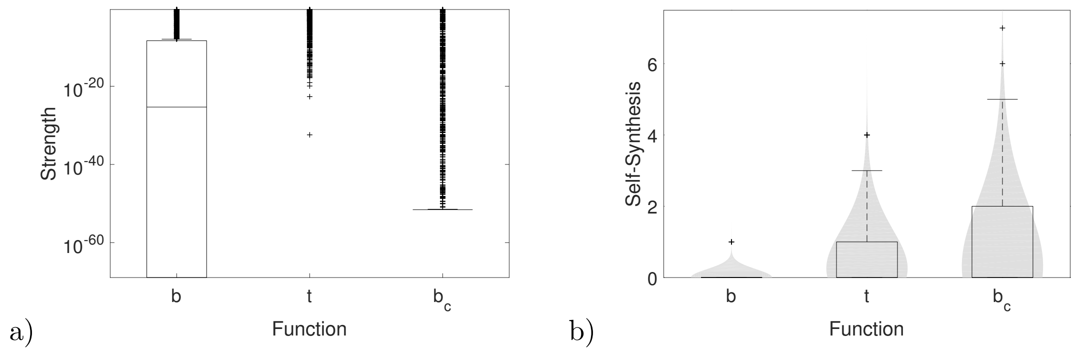


Figure 6.11: Comparison of (a) strength and (b) self-synthesis with different functions (Faulkner et al., 2017).

These two functions are shown in Figure 6.10. Faulkner et al. (2017) find that function b, based on the Maxwell-Boltzmann velocity function, produces a large range of particle link strengths and next to no self-synthesis, while the triangular function, t, has a near non-existent range of strengths but a large amount and range of self-synthesis. (See that work for the details of the algorithm used to produce the data; in brief, data is gathered after many reaction and decomposition attempts have been made in a well-mixed reactor.) Faulkner et al. (2017) conjecture that both of these effects are controlled by the function's tail. In the case of self-synthesis the behaviour is caused by the lack of tail, while the range of strengths is related to the large tail on b.

Faulkner et al. (2017) consider an alternative function to test that conjecture, by using a version of b with a cut-off at $x = 3$, b_c, mirroring the lack of tail of t. This exhibits the expected features of t, with a limited strength range with an increase in range and amount of self-synthesis, Figure 6.11. While b_c does provide us with an increased range of strengths and self-synthesis occurrences, nevertheless the range of strengths observed are still very constricted compared to the original b function.

Instead of arbitrarily cutting off the tail as in (Faulkner et al., 2017), we can now use our semi-ring approach to combine the original functions to get new functions with potentially

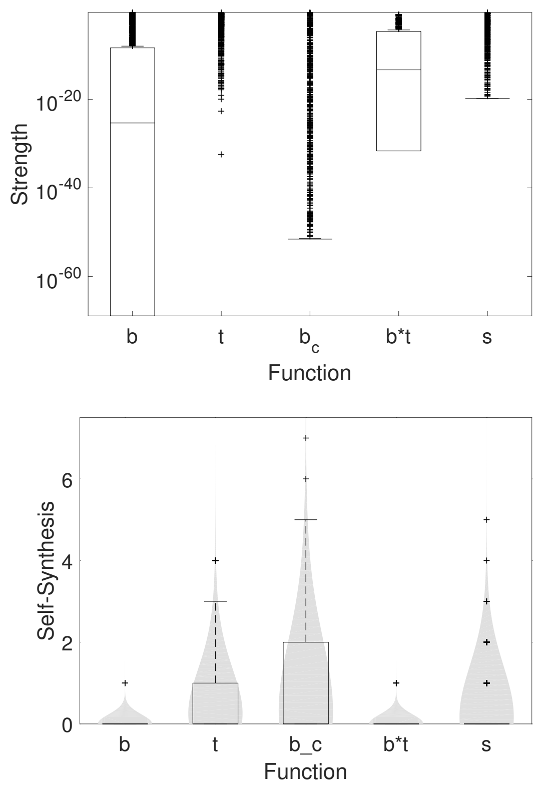


Figure 6.12: Comparisons of strength and self-synthesis with different functions

both desirable behaviours. Figure 6.10 shows that in the tail region ($3 < x$), $\min(b, t) = t$, so no tail, whereas $\max(b, t) = b$, with its tail.

If the conjecture is correct and the tail is responsible for both features in opposing ways, then we need a way to have a tail and no tail. This is possible to some degree if we switch between the two functions at different intervals of x in a regular manner. We introduce two square wave probability functions.

$$s_0(x) = \begin{cases} 1 & : 2n - 1 < x \leq 2n \\ 0 & : 2n < x \leq 2n + 1 \end{cases} \tag{6.32}$$

$$s_1(x) = 1 - s_0(x) \tag{6.33}$$

If we take the $\min(s_1, b)$ and $\min(s_0, t)$, then we get the value of b when s_1 is 1, that is in intervals starting with an even value, and we get the value of t where s_0 is 1, in intervals starting with an odd value. We use max to combine these: $s = \max(\min(b, s_1), \min(t, s_0))$. We see in Figure 6.13 that s is non-zero between 0 and 4, then as it continues onwards it alternates between 0 from the t functions and the very small tail values of b.

The results for s are also are shown in Figure 6.12 (rightmost boxplot). We still do not have a large range of strengths, but we do have a larger range of self-synthesis than before,

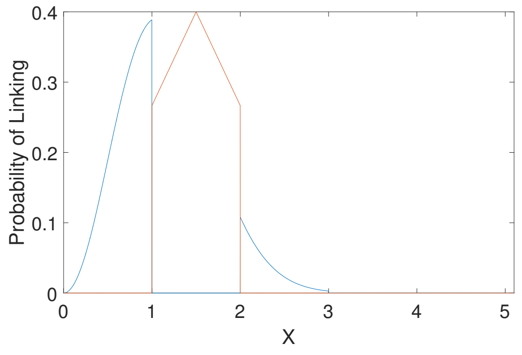


Figure 6.13: Developing piecewise functions using square waves s_0 and s_1, and the min and max functions. The resulting function s is shown; it is zero between 3 and 4, but then has a small tail again between 4 and 5.

even if they are outliers. This function both does and does not have a tail, to some degree, meaning it may be correct that the tail controls both effects.

Clearly more investigation is needed to discover the precise effects of different features. However, we now have a formalism within which to perform such investigations in a systematic manner.

Exploiting the semi-ring algebraic structure has allowed us to consider combinations of existing probability functions. This provides us with the means to generate a large number of possible probability functions generated from a small set of base functions.

We have used the semi-ring approach to test our conjecture from (Faulkner et al., 2017) that the tail of the probability function used for link formation in Jordan Algebra AChems influences the range of both strength and self-synthesis in opposing manners. As part of this test we have developed the mathematical basis for a piecewise function to be formed using the semi-ring. This means we can work with piecewise functions without having to check the resultant is still a valid probability function.

This has allowed us to investigate the conjecture in a way that we were not able to in (Faulkner et al., 2017). It is, of course, possible to define the piecewise function s from scratch, but what the semi-ring approach gives us is that we can always form a valid probability function as a piecewise combination of existing probability functions.

This means that in later systems we may be able to automatically generate probability functions, or change them in response to changing state variables. Using the semi-ring structure given here we can do this in multiple ways without having to test validity.

6.6 Summary

We have developed a set of probability spawning functions for use in linking in Jordan Algebra Artificial Chemistries. These have covered a wide range of options in terms of peak height, area, tail length, and height. These have produced various and different effects in our system. Some of these effects correlate with particular features, such as the tail cut off seeming to be connected to an increase in self-synthesis.

We have also found that there is a lot of complexity in our system. We see that different sets of functions can have very different effects. There are complex interactions between our psfs and the rest of our system.

6.7 Further Work

Here we have investigated only one of the probabilities used in our AChem: that used to give the link strength. There are other probabilities that should be similarly investigated, and potentially tuned, to give different behaviours. For example, parameters to the functions, such as a and σ, could be coupled to states of the system, such as temperature, to allow the probabilities to change with the system state.

The orientation probability p_a (eqn.6.7) is a candidate for variation. It has a role in both the overall linking probability p_l, and, crucially, in the choice of pair of eigenvalues. Currently we chose the eigenvalues and vectors that maximise p_a. We might instead chose by minimising p_a, or weight the choice in a less deterministic manner. Allowing the system, at times, to use less optimally "aligned" eigenvalue pairs may produce weaker or stronger bonds between the same particles. This would allow hard-to-form but then very strong links to occur. It would also provide weaker versions of particles. Because of their weaker links, these may be better at enabling catalysis by linking to further particles before decomposing.

Currently p_l uses the maximum of its two probabilities. Again, a different choice, such as the minimum, or some other combination of the two functions, would give different system behaviour.

We see there are many potential ways of combining probabilities. Further work will investigate how algebraically-defined combinations of psfs could be used to give combined properties that allow finer tuning of our system, for example, taking the maximum of the high-peaked $g_{0.5}$ and the large-tailed b_2 to give both properties.

Chapter 7

Swarm Chemistry

In this chapter we modularise the existing Swarm Chemistry system. We describe our variant of swarm chemistry using MetaChem notation. We discuss the advantages of this description and implementation before it is used in later chapters.

So far we have developed a framework for describing artificial chemistries to replace the limited (S, R, A) format. However all the chemistries we have described in the new framework so far were built originally within (S, R, A).

In this chapter we look at describing Swarm Chemistry (Sayama, 2009), a system built to explore beyond the (S, R, A) format. Despite not having a comfortable description in the (S, R, A) framework, SwarmChem is widely known and accepted as an Artificial Chemistry. It is therefore important to show that while (S, R, A) may struggle with SwarmChem, MetaChem comfortably describes it. In the description of SwarmChem in MetaChem we start to see that SwarmChem is not some borderline AChem some have thought it. It has many close similarities to other more classical AChems when we consider its controls and algorithms, rather than simply its lack of physical connections.

Swarm Chemistry does not fit in the (S, R, A) format because it does not have direct interactions between particles. The individuals in SwarmChem, often referred to as boids or agents, interact by each boid changing its own velocity based on the local positions and velocities of its neighbours. This involves no knowledge of the neighbours' parameters, just observation of velocity and position. This gives the effect of swarming or flocking like that seen in birds. Different parameters sets produce different swarms in terms of the density of the swarm and how it moves. In SwarmChem boids with different parameters are allowed to mix as in Figure 7.1.

SwarmChem is a framework for a class of artificial chemistries. Its intention is to explore how higher level statistical rules for chemical systems emerge from lower level local interactions. It does this with the basic concepts of Reynolds (1987)'s Boids. The key change it makes to the boids system is to assign parameter sets to individual boids rather than having

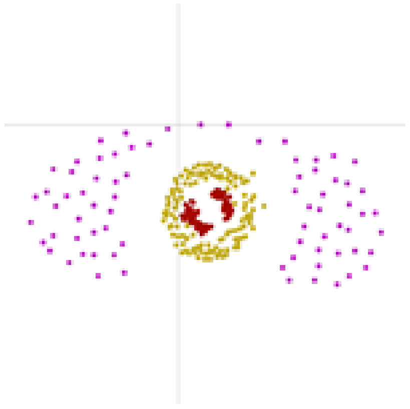


Figure 7.1: Pulsating Eye swarms contributed to SwarmChem by Benjamin Bush using recipe: 102 * (293.86, 17.06, 38.3, 0.81, 0.05, 0.83, 0.2, 0.9) 124 * (226.18, 19.27, 24.57, 0.95, 0.84, 13.09, 0.07, 0.8) 74 * (49.98, 8.44, 4.39, 0.92, 0.14, 96.92, 0.13, 0.51), http://bingweb.binghamton.edu/ sayama/SwarmChemistry/. An example of interesting 2D organisation using 3 different parameter sets for three hundred boids.

them being global fixed values. This allows one to consider interaction of heterogeneous swarms and the patterns these can form.

Flocking in both boids and swarm chemistry works as follows: at each time step for each boid we first work out the neighbourhood of the boid. We then steer the boid towards the centre of the group of boids; this is called *cohesion*. We then steer toward the average heading of the neighbouring boids; this is called *alignment*. We then steer to prevent crowding, moving to increase the *seperation* between boids. Finally we check the pace of the boid towards its normal speed and move the boid. This is done on all boids at once so we use the information of position and velocity from the current time step to calculate the next. See Figure 7.2 for a visual description.

This framework has been extended to investigate open-ended evolution. This is done through the addition to the system of *recipes*, which are a set of parameter values. Each boid is currently operating based on a particular recipe, but in the extended versions might carry many recipes (Sayama, 2010b,a, 2011). Recipes, or the weightings used to choose the active recipe, can to exchanged and changed by other boids. This can be done based on collision or other factors.

This allows boids to change and optimise to maintain structures giving us evolution in the system if we consider a boid to be a child of itself when its parameters change. More recently we have seen this go a step forward by identifying these larger structures and considering them as their own entities.

Another extension has been to place the boids in the 3 dimensional environment (Sayama, 2012).

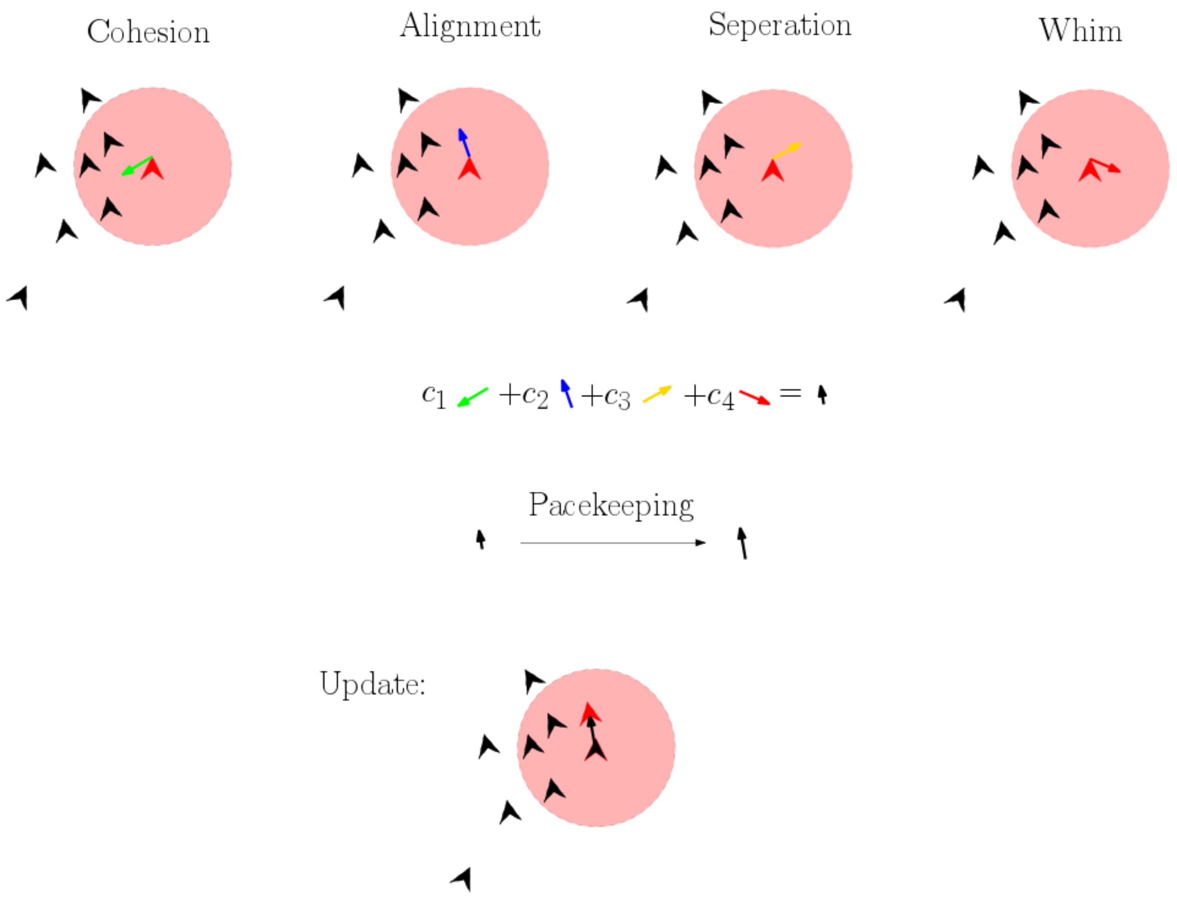


Figure 7.2: A pictorial description of flocking in Reynolds boids and swarm chemistry. The red disk shows R the perception distance of our boid.

7.1 Swarm Chemistry

SwarmChem boids have 8 parameters, Table 7.1. These are used to control the flocking, so the agents move in group behaviours and do not crash about.

These parameters are used to define the properties of how to update each boid (i). The algorithm elements for the update that we use are the position of the boid (x_i), velocity(v_i), and acceleration(a_i). We also use the properties of the boids neighbours, N, in the perception range including the average velocity ($\bar{v}$), position($\bar{x}$) and separation($\bar{s}$).

$$\bar{v} = \Sigma_{j \in N} \quad v_j/|N| \tag{7.1}$$

$$\bar{x} = \Sigma_{j \in N} \quad x_j/|N| \tag{7.2}$$

$$\bar{s} = \Sigma_{j \in N} \quad (x_i - x_j)/|x_i - x_j|^2 \tag{7.3}$$

We can then undergo a series of processes described by the following equations: straying, cohesion, alignment, separation, whim, acceleration, prohibit overspeeding and pacekeeping. These are applied to each boid to give a new velocity, v_i^*

Parameter	min	max	Description
R	0	300	Radius of local perception radius
v_n	0	20	Normal speed
v_m	0	40	Maximum speed
c_1	0	1	Strength of cohesive force
c_2	0	1	Strength of alligning force
c_3	0	100	Strength of seperating force
c_4	0	0.5	Probability of random steering
c_5	0	1	Tendency of pacekeeping

Table 7.1: Swarm Chemistry parameters controlling flocking and their minimum and maximums. These are set individually for each agent rather than globally.

$$\text{Straying:} \qquad a_i < -(r_{\pm q}, r_{\pm q}) \tag{7.4}$$

where $r_{\pm q}$ is a small random change to the acceleration with random direction. Straying is our random walk option otherwise:

$$\text{Cohesion:} \qquad a_i < -c_1(\bar{x} - x_i) \tag{7.5}$$

$$\text{Alignment:} \qquad a_i < -a_i + c_2(\bar{v} - v_i) \tag{7.6}$$

$$\text{Separation:} \qquad a_i < -a_i + c_3\bar{s} \tag{7.7}$$

$$\text{Whim:} \qquad a_i < -a_i + (r_{\pm q}, r_{\pm q}) \tag{7.8}$$

$$Whether we flocked or strayed we perform the remaining actions. \text{Acceleration:} \qquad v_i^* = v_i + a_i \tag{7.9}$$

Where v_i^* is the new velocity leaving v_i as the old.

$$\text{Prohibit Overspeeding:} \qquad v_i^* = \min(v_m/|v_i^*|, 1) \bullet v_i^* \tag{7.10}$$

$$\text{Pacekeeping:} \qquad v_i^* = c_5(v_n/|v_i^*| \bullet v_i^*) + (1 - c_5)v_i^* \tag{7.11}$$

Once we have calculated a new velocity for all our boids we update their velocity and apply it to update the position.

7.2 Description

Here we work with our own variant of SwarmChem. It has previously been presented in our work with nested chemistries which can be found in chapter 8. Here we will describe it in more detail with a full macro graph and micro graph of the update process, an expansion of the flock action.

We load the initial parameter set and randomly position our boids using **s**:Load Parame-

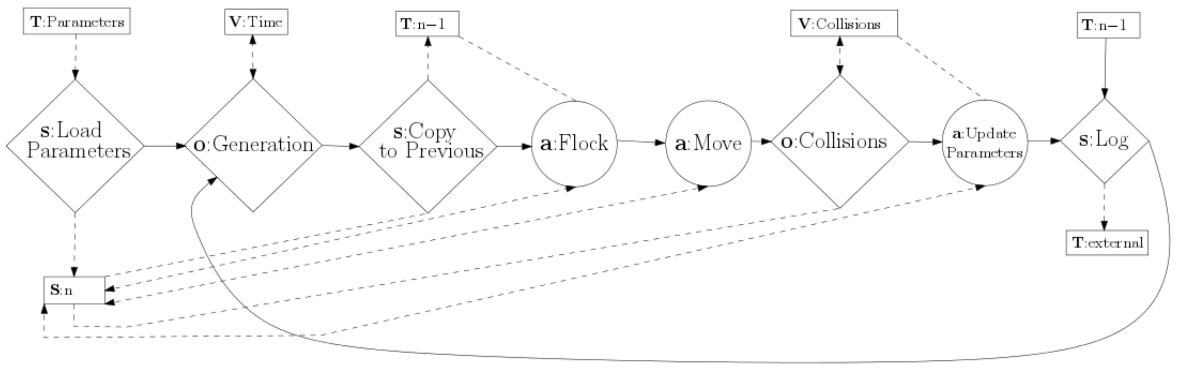


Figure 7.3: Macro level Swarm Chemistry graph. It includes the timing counter **o:**generation, flocking and moving as well as collisions and the logging sampler to track the chemistry.

ters. We then iterate our clock with **o**:Generation. This "tick" is part of our discrete timing system that is consistent with all current swarm systems. This is very evident in the rest of the macro system as well.

We use the **s**:Copy to Previous sampler to copy the current generation to the tank marking it as the previous generation. This gives the current position of all the boids and their parameters. These will remain unchanged as we calculate the next generation in **a**:Flock. **a**:Flock is where the update occurs, applied to each boid in turn and following the classic boid set of effects. This is expanded upon in Section 7.3.

We then move all our boids based on their parameters and currently facings and velocities with **a**:Move. This again is common to all swarm chemistries. As part of our variant we then check for collisions (**o**:Collisions) and record them in **V**:Collisions. This then allows us to update our parameter sets which are changed by collision. In our variant of swarm chemistry we exchange a random number of parameters when a collision occurs. This is different to the weighted recipe method used else where. It also means that collisions in our system are "no fault" in that both boids are equally changed. In other versions one boid is dominant in the collision and enforces its recipe on the other. There is a small amount of rationalization in our version to prevent trading normal or max parameters if it would mean that our normal would exceed our max.

Finally we complete the loop by logging the previous generation to external storage and clearing the tank ready for the current generation to be copied in. This ends our macro algorithm which then loops back on itself to the start.

7.3 Flocking

The flocking action in this system contains most of the activity of the system so in this section we expand it in a micro level graph, Figure 7.4.

Here we see the break down of the graph into the different transition functions looping over the entire set. In some sense this is actually a large number of action nodes and all the transition functions occur each time.

7.3.1 read()

We start by reading various individuals into different tanks. First we randomly read a single boid from the current set in **S**:n for updating. We then read out all the neighbours of this boid, defined by its perception distance, to speed up analysis. Finally we use an observer to generate $(\bar{v}), (\bar{x})$ and $(\bar{s})$ of the boids in the neighbourhood. This gives us a full set of values for the rest of the process.

7.3.2 check()

This check not really a complete check; it never exits the function, but it does decide the actual process of the function. It makes a choice between a random walk and normal flocking behaviour based on the number of neighbours. If $|N| > 0$ then flocking else a random walk.

7.3.3 pull()

It is at this stage that we remove the selected boid from the current generation. This is done at this stage regardless of which choice has been made. It would be possible to do this earlier but then we would be breaking with the transition function format. This is not always an issue but cannot correctly then be summarised into a single node.

7.3.4 process()

In the lower case processing consists of a random walk, so the boid sets itself with a random velocity as shown in Equation 7.4.

In the case of flocking with neighbours four actions are needed from the formulae given previously. We implement them in the following manner **a**:Cohesion implements Equation 7.5, then **a**:Alignment Equation 7.6, followed by **a**:Separation and Equation 7.7. Finally a:Whim implements 7.8 which keeps the system from behaving too predictably.

Finally our branched systems rejoin and our remaining Equations 7.9-7.11 are applied in **a**:Pacekeeping is applied. The intent here is to prevent boids from constantly increasing in speed by modifying their speed back towards their normal velocity.

7.3.5 push()

Finally having finished processing we push the boid to a different sample to track the boids which have already been updated. We then check to loop based on if the original generation sample has been emptied yet. Finally when we are finished looping we push the content of our new sample to the old sample so that it is available outside the flock function.

Note that at no point within this function do we update velocity, this is done along with position in the outer **a**:Move node.

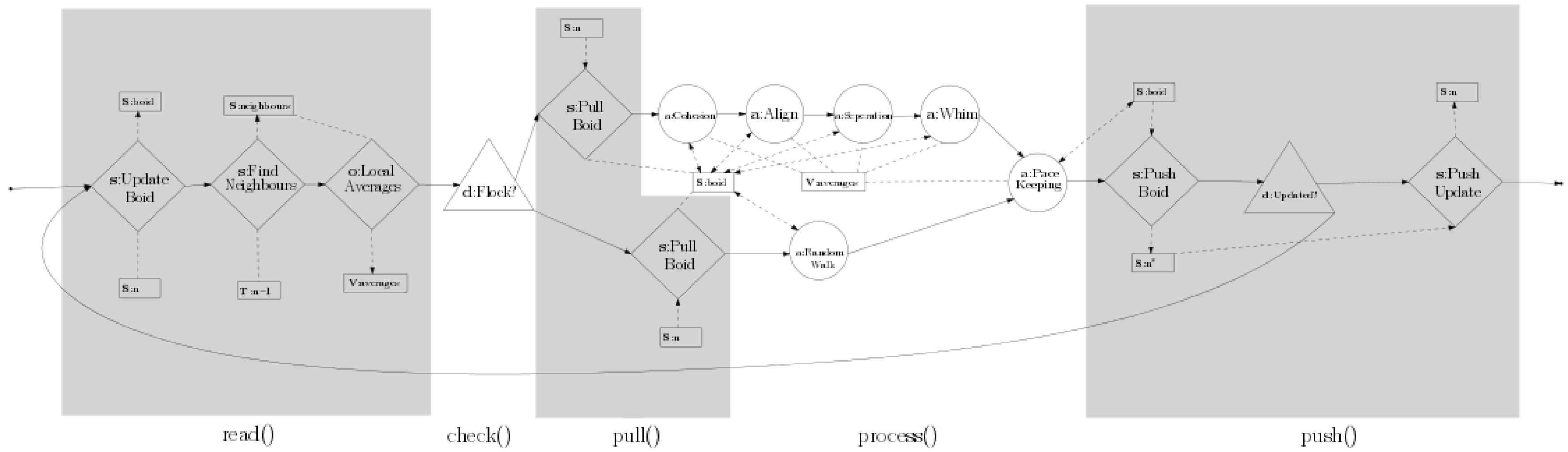


Figure 7.4: Micro Level description of flocking in Swarm Chemistry. Due to the large loop in this we could consider it a description of many identical nodes or as one node at the Macro level.

7.4 Advantages for SwarmChem of MetaChem representation

This modularisation has the potential to seem overkill for SwarmChem. The system described here has very little difference from basic versions of the chemistry and next to no advantage as a system on its own. The advantage to this redescription and reimplementation lies in the modularity through MetaChem itself. This holds advantages in three areas: variation, extension and analysis. The first is done through the modularity and the last allows the potential for eventually developing this into a self-reflective system.

7.4.1 Modularity

This is the simplest advantage. There are already at least 5 different methods for resolving collisions in Swarm Chemistry including the method given here. In the modular system these can be easily interchanged. With little variation to the rest of the graph, a decision could be added to allow different collision methods to be used in a single system.

Other aspects could also be swapped out to create new systems, such as the addition of an event cycle that could flow new boids into the system and/or drain existing boids from the system. Energy or temperature effects could be added through modification of only a minimum of nodes and addition of new nodes, keeping coding required to a minimum.

Such modular design can help speed up development of new systems and allow the designer to focus on the new features without concern for maintenance of back compatibility and comparison, as long as their logging and analysis nodes (normally samplers and observers) remain unchanged.

7.4.2 Extension

We can also now extend beyond SwarmChem alone. With this MetaChem implementation we can link this chemistry with others to create more complex multipart systems. This is discussed more in the next chapter.

As well as the hierarchical nested chemistry we present in the next chapter, other options are possible. For example, SwarmChem could be used as a basis for a larger chemistry in which the swarm is the subsymbolic structure for larger particles. We could join it to another chemistry such that the boids were paired with particles of another sort, and through the other chemistry's rule set it would become possible for the joint particles to form elastic or rigid links. We could simply have SwarmChem occupy the same space as another chemistry and introduce completely new interaction rules between the two species of particles.

We could even extend the current work on analysis of SwarmChem for higher level organisms to a form of swarm chemistry within swarm chemistry. All of this can now be done with a minimum of re-implementation because of the common language and structure of our MetaChem.

7.4.3 Self-reflective SwarmChem

We briefly mentioned earlier (Section 4.4) the concept of the current static form of MetaChem being the first in a process that could eventually develop dynamic graph MetaChems and evolving MetaChems.

In terms of SwarmChem, in the first phase we would be able to produce swarm chemsitries that could change at run time. This could be used to let them assess their own organisms and build a population for a higher level chemistry that it could then trigger at the correct moment. A designer would still have to preempt the development of a broad system and provide the rule set and nodes for the system to build these new objects when needed.

The second phase takes the next step beyond this, and would allows swarm chemistry to create the nodes and rules it needs to change itself and create a new chemistry. It could introduce and remove elements as needed to reach a goal set by the designer or simply to explore the space of artificial chemistries.

Chapter 8

NestedChem

We introduce a modularisation of artificial chemistries (AChems). This allows us to define a standard linking method between AChems. We illustrate the approach with a system that nests a Jordan Algebra AChem (JA AChem) inside agents of SwarmChem, and show how our modular approach allows us to define and experiment with multiple variants in a standard manner. Potential for future formalisation is discussed.

8.1 Introduction

Sub-symbolic artificial chemistries (AChems) (Faulconbridge et al., 2009; Faulconbridge, 2011; Faulkner et al., 2018) are generally AChems whose atoms and particles have internal structure that defines their behaviour. These systems so far have been analogous in their behaviour to natural chemistry viewed at the level of atoms and molecules.

Many other AChems work to reflect the properties of chemistry at the level of cells (Madina et al., 2003; Hutton, 2007) or chemical reaction systems (Soula, 2016). In natural chemistry these different levels are closely related: cells contain chemical reaction systems, and chemical reaction systems are based on individual particle and atom interactions. While attempts have been made to bridge the gaps between such levels in individual systems (Liu, 2018), so far the systems are very simple and lacking in more complex features.

We propose to take advantage of feature-rich existing AChems by combining them to give a system that can span different levels of activity and behaviour in a single AChem system. We demonstrate this approach through our own Jordan Algebra Artificial Chemistry (JA AChem) (Faulkner et al., 2016, 2017) and Sayama's SwarmChem (Sayama, 2009, 2010b, 2011, 2018b).

JA AChem particles are algebraic objects. It can form particles with complex structures. It is a sub-symbolic system that has probabilistic linking and decomposition. So far this system has been aspatial, working on a well-mixed tank model in which all particles can link with each other.

SwarmChem is based on Reynolds' Boids (Reynolds, 1987). Its particles or agents move

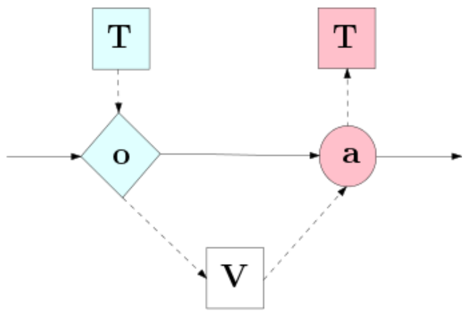


Figure 8.1: Communication link between first AChem (blue) and the second AChem (pink). Solid arrows show control flow, dashed arrows show information flow. The observer node observes the first AChem's tank and pushed the information to the environment. The action node acts on the second AChem's tank based on information read from the environment.

and flock based on their parameter values. In existing SwarmChem instantiations the sets of parameter values are generally evolved with human interaction to generate flocking behaviours. The system agents contain all sets of values in the system and choose one to use based on a weight matrix. These are changed by collisions. In the version of SwarmChem used here we do not have a predetermined set of possible values. In our version the agents swap some number of parameter values. The only limitation is that the parameter values must make sense after the exchange (for example, normal velocity must not exceed maximum velocity).

8.2 Method

We can connect any two AChems by giving them the ability to communicate via their environment, Rainford et al. (2018). This communication can be uni- or bi-directional. In order to talk about combining AChems we need to be able to talk about different parts of different AChems. We do this by representing information and control flow in an AChem in a graph with multiple types of nodes and edges, Tables 4.2 and 4.3 respectively. The basic graph structure of communicating AChems is given in Figure 8.1.

We use shading to indicate the ownership of a node by a single system. A node owned by one AChem cannot directly communicate with the nodes owned by a different AChem. Instead, information is shared using an environmental node that is not owned by either AChem. So in Figure 8.1, the dark grey observation is of a tank in the 'dark grey' AChem, and the light grey action is on a tank in the 'light grey' AChem. The figure shows uni-directional communication, in which one AChem influences the other. By adding a second link in the other direction we could establish bi-directional communication. Both the action and the observation are defined by the designer.

For example, if we wish to establish side-by-side chemistries then our observation will produce a summary statistic that is a value, or set of values, based on the whole system, which will uniformly affect the entire system in the second chemistry. Alternatively, the observer

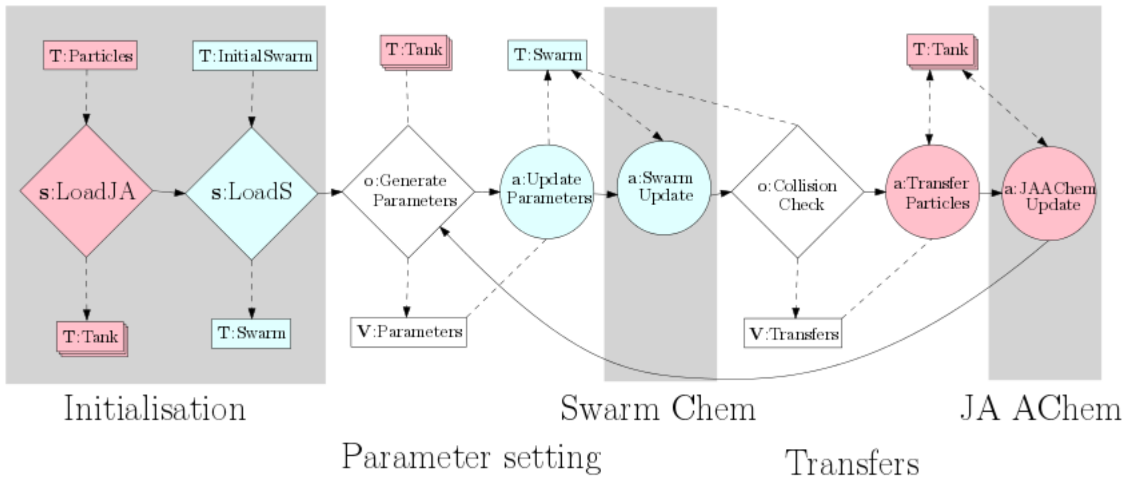


Figure 8.2: Graph of nested chemistry using metachem modular graph notation. JA AChem nodes are shown in pink, SwarmChem nodes in blue. White nodes are either shared or not natively part of either AChem.

can generate statistics based on individual particles, which can then affect individual particles in the second AChem.

We give an example of a “nested”, or multi-level, AChem with bi-directional communication. The observer of the lower-level AChem generates a set of values over a large number of particles in that AChem. These values are then used to influence the behaviour of a single individual in the higher-level AChem. In turn the behaviour and interaction of one or two particles in the higher-level AChem influence a large number of particles in the lower-level AChem.

8.3 Example: Nested Chemistry

We generate a new set of chemistries by combining two AChems from the literature (Faulkner et al. (2016) and Sayama (2009)) by linking the modules in various ways to give seven distinctive systems; an eighth system is achieved through a change in system settings.

The largest of these systems is a fully nested AChem that contains all modules used in our systems, Figure 8.2.

In this system we treat each of the agents of SwarmChem as a well-mixed tank of JA AChem particles.

We have two links in our nested system: Parameter Setting and Transfer. In Parameter Setting we generate a set of parameters values for each SwarmChem agent based on the particles in its tank. These sets are saved into the environment and used to update the Swarm. In Transfer we detect collisions between members of the Swarm. This information is saved in the environment and then used to transfer particles between the well-mixed tanks of the JA AChem system.

This provides a means of communication between the two systems. SwarmChem's spatial movement provides a limitation and control on particle exchanges in JA AChem between different tanks. Likewise, the JA AChem tanks communicate with SwarmChem by changing its parameter values, which influences the agents' spatial movement and likelihood of collision.

We divide the nested chemistry system into five sections, as shown in Figure 8.2:

Initialisation Initial tanks of JA AChem particles and initial swarm agents are loaded into the system and stored separately with matching indexing to allow for reference between the two.

Parameter Setting The current contents of the tanks are analysed to produce a set of parameter values for each tank. These are stored in the environmental variable "**V:**parameters" (8.2). The swarm then updates itself based on the content of this variable.

SwarmChem The agents of SwarmChem update themselves using a single SwarmChem time step.

Transfer SwarmChem assesses whether any collisions have occurred between its agents in the system. It stores out a record of these collisions in the environmental variable "**V:**transfers". JA AChem uses this variable to exchange parameter values between tanks based on the SwarmChem collisions.

JA AChem Updates by performing a number [1] of attempts at bonding and an equal number of attempts at decomposition. All tanks are independent and mass-conserving well-mixed tanks.

It might be noted that there are four invalid edges in the macro system graph of NestedChem: (**a:**Update Parameters, **T:**Swarm), (**a:**Swarm Update, **T:**Swarm), (**a:**Transfer Particles, **T:**Tank) and (**a:**JA AChem Update, **T:**Tank). All of these edges appear to allow actions to push to tanks which is not allowed. In the case of **a:**Swarm Update and **a:**JA AChem Update we have seen the expanded graphs of these nodes in the system graphs of each system. In those graphs we see that the actions carried out by these nodes mean they always move the particles to samples before making any changes. Here we connect directly to the tanks simply as an abuse of notation.

In the case of **a:**Transfer Particles if we were to expand this node we would see that all the operations of this node are carried out by samplers and there is therefore no issue that before starting the particles have not been moved to a sample. Finally in the case of **a:**Update Parameters the process function is applied over all particles in the system meaning the sample would be the entire tank so in another abuse of notation and to avoid introducing a further two control nodes and a container to move the entire contents back and forth we

[1]See "gen size" in Table 8.1

allow the node to connect directly to the tank. Please take it as given that in the expanded form this sampling would occur.

8.3.1 Modular Systems

From this full system we can derive eight variant systems. The control flow of these systems is shown in Figure 8.3.

I. Nested. The full Nested AChem system as shown in Figure 8.2

II. Nested without collision. JA AChem particles are not transferred between tanks, but still determine the parameter values of agents in the SwarmChem

III. SwarmChem. SwarmChem agents randomly exchange parameter values on collision; there is no communication with the JA AChem.

IV. SwarmChem without collision. A very basic form of SwarmChem in which the agents interact only through Boid like flocking behaviours.

V. JA AChem single tank. A single well-mixed tank of JA AChem. The same number of evaluations are used per generation and the same number of starting particles are also used as the other systems.

VI. JA AChem multiple tanks with no interaction. A JA AChem with the same number of tanks as in the nested version; there are fewer atoms and particles in each tank, but the same number of overall atoms and evaluations are used.

VII. JA AChem multiple tanks with random transfers. The same system as in **VI** but with tanks randomly selected to randomly transfer particles between them.

VIII. JA AChem multiple tanks with grid transfers. The same as in **VII** but transfer tanks selected based on a Moore Neighbourhood, Figure 8.4

8.3.2 Settings: Jordan Algebra AChem

The Jordan Algebra AChem has previously been presented as a single well-mixed tank. JA AChem's properties emerge from its underlying algebraic structures. When linked, the resultant particle can be represented as a tree, with atoms as its leaves and sub-particles at each node. The algorithm for one generation of JA AChem is given in Chapter 5.

In order to nest JA AChem into SwarmChem we need to work with multiple tanks. The algorithm is an update cycle for a single tank. We can apply this to each tank in the system. The relevant settings for the six systems that include JA AChem here are given in Table 8.1. [2]

[2] All systems use an additional collision probability of 0.2 and the Boltzmann inspired b probability from Faulkner et al. (2017)

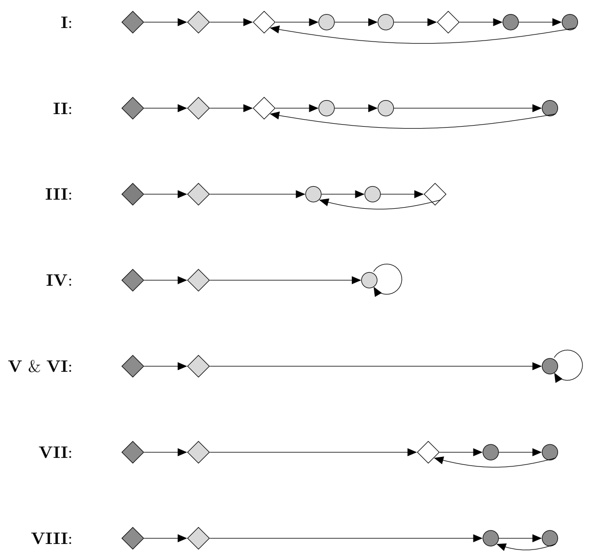


Figure 8.3: System Combinations

Table 8.1: JA AChem Settings for each system

System	**I**	**II**	**V**	**VI**	**VII**	**VIII**
#tanks	50	50	1	50	50	50
tank size	200	200	10000	200	200	200
gen size	200	200	10000	200	200	200
#rounds	100	100	100	100	100	100
transfers	collision	none	none	none	grid	random

These settings give us the same number of linking and decomposition attempts in all systems. The three collision methods work as follows:

Collision: Transfers happen between two tanks when their associated SwarmChem agents collide.

Grid: Transfers happen by randomly selecting a single tank and then selecting a second tank from a grid based neighbourhood, see Figure 8.4.

Random: Two tanks are chosen at random and exchange materials.

8.3.3 Settings: SwarmChem

SwarmChem is a spatial AChem based on Boids (Reynolds, 1987). There is one set of settings relevant to SwarmChem. In all experiments we have 50 agents in a 2000 × 2000 space of arbitrary spatial units.

IV is the most basic version of SwarmChem, in which parameter values are fixed throughout and agents do not change their behaviour during a run.

In **III** we further develop the SwarmChem system to include changes in parameter values. In traditional SwarmChem this is done with weight matrices that assign priority to different

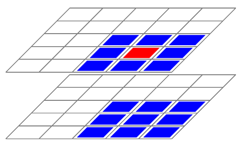


Figure 8.4: Grid based neighbourhood

parameter value sets. We use another method, still based on collisions being the trigger for change. For comparison purposes, we make the parameter value changes more analogous to the transfer of particles in the nested system. In the case of a collision in **III** we swap a random number of the agents' parameter values. Swaps are then reversed if they give contradictory settings (e.g. maximum velocity less than normal velocity).

In **I** and **II** the SwarmChem agents have an internal state defined by the tank of associated JA AChem particles. These particles change over time as they react, and in doing so change the parameter values of the SwarmChem agent.

In **I** we also have collisions that cause the transfer of JA AChem particles from one agent's tank to another, allowing for information transfer between agents, which may prevent the agents from becoming stable.

8.3.4 Analysis methods: Jordan Algebra AChem

We focus our analysis of the JA AChem level of our systems on the size of the particles and the resultant number of particles in the system or tanks over time.

These numbers should stabilise quickly in the system, but we expect the systems with transfers to be less stable than others. Particles being transferred in and out of the tanks should disturb any equilibrium.

We also expect to see larger particles in the partitioned systems as the smaller size of the tanks limit the sampling possibilities, increasing the chances of selecting molecules which already contain multiple particles. As these are used and the number of particles in the tank decreases, these probabilities should further increase.

8.3.5 Analysis methods: SwarmChem

We can observe many different statistics on the agents of the swarm; here we focus on the relative position of an agent to its visible neighbours. This provides us with measure of how well our agents are clustered with each other. In homogeneous flocking this parameter's value should be very similar across agents, as a flock all have the same perception radius and tendency for avoidance. In SwarmChem these have greater variation but should be similar in sets of agents forming a swarm. Here we therefore expect to see greater variation in the nested SwarmChem where all values of perception radius and tendency for avoidance are possible.

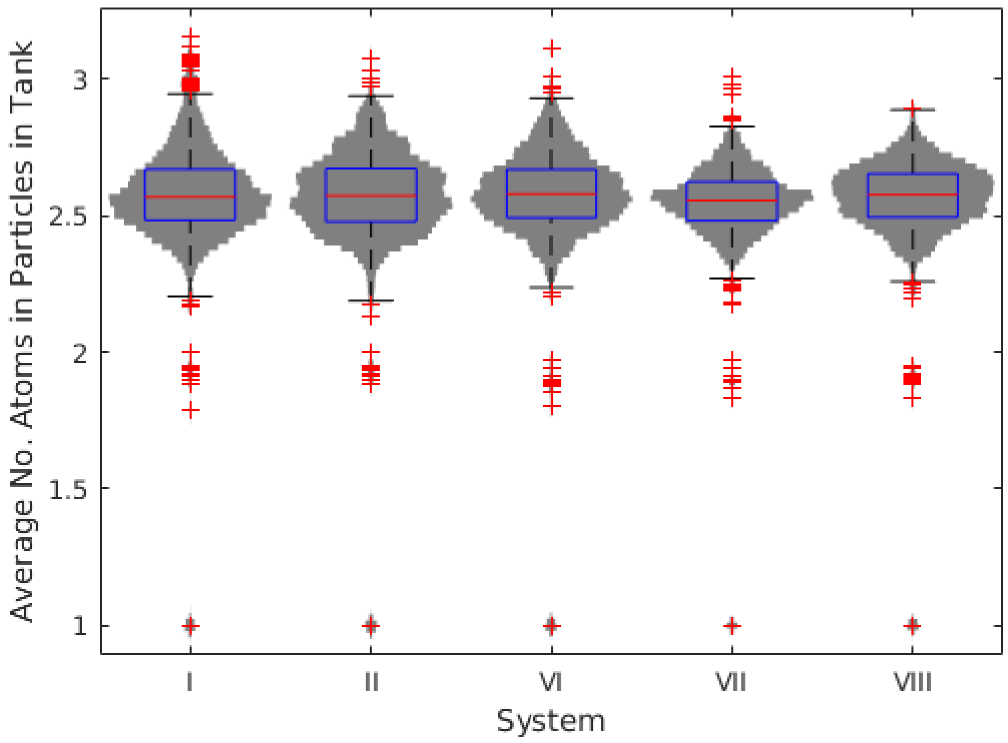


Figure 8.5: Average number of atoms in each particle in each Jordan Algebra AChem tank in a multitank system

8.4 Results and Discussion

8.4.1 Effect on JA AChem

There is a statistically significant difference in the sizes of particles in the system between the single tank, **III**, and everything else ($p < 10^{-33}$ ranksum test, large effect size $A = 0.99$ (Vargha and Delaney, 2000)). It also stabilises at a much higher number of particles (8824 particles) compared to the other systems (**I**, **II**, **VI**, **VII**, **VIII**, with 3750 – 3850 particles). We can therefore reject the null hypothesis that our changes to the system have no effect.

Looking at the multitank systems we can see that these systems have similar particle sizes, Figure 8.5. These have similar medians, but somewhat different distributions. Most distinctive is **VIII**, which is statistically significantly different (ranksum) from all the medians (except **II**), with medium or larger effect sizes.

We can predict that **II** and **VI** should be equivalent at the JA AChem level. This is because there is only a single uni-directional link in **II**. This is true in our results ($p = 0.057$, ranksum). This is particularly true as we are using Bonferroni corrections giving us a threshold for significance at 0.0032 rather than 0.05. This suggests that the number of runs and number of generations in the JA AChem has been sufficient in this example to give us the general behaviour of these systems.

8.4.2 Effect on SwarmChem

There are many different versions of SwarmChem, which can best be categorised in terms of their morphogenic hierarchy:

Category A: Homogeneous Swarm

Category B: Heterogeneous Swarm

Category C: Heterogeneous Swarm with redifferentiation

Category D: Heterogeneous Swarm with redifferentiation and information sharing

Here differentiation means that a particular agent can have different parameter values and redifferentiation means that those values can change.

Category A represents the original Boids work with its heterogeneous flocks of agents. Category B can be seen in the early forms of SwarmChem where we have a heterogeneous swarm. SwarmChem then moves to a Category C system by adding differentiation by changing the weighting of parameter value sets based on its local neighbours In more recent work (Sayama, 2018b) SwarmChem introduces a version which could be seen as bridging the gap to Category D by allowing local information sharing.

Our full system, **I**, falls solidly into Category D with transfers and the differentiation that comes from the JA AChem. When we lose the transfers in **II**, we have a Category C. Our simpler AChems both fall mostly in Category B as heterogeneous swarms.

We get flocking behaviour in all systems, despite no control or evolution to develop it. Figure 8.6 shows that **I** and **II** form similar swarms to **III** and **IV**. This is contrary to our original expectation that forming swarms would be harder for nested systems. It may be because the JA AChem tanks tend to similar configurations of particles over time. In comparison the SwarmChem-only systems finish with the same parameter values as they start with. This seems to be reflected in our observations which show very little difference in our systems. However we do over all runs find a large difference in the relative position, p_r, of agents to their neighbours (Figure 8.7)

$$p_r = |\langle x \rangle_i - x_i|^2 / R_i^2 \tag{8.1}$$

where $\langle x \rangle$ is the average position of an agent i's neighbours. x_i is an agents current position and R_i is the agents perception distance.

This shows we have significant differences between our nested systems (**I** and **II**) and our swarm systems (**III** and **IV**). They are statistically different ($p = 10^{-207}$, ranksum) with large effect size ($A = 0.90$). Looking to further differentiate our systems we find no difference between the swarm systems. We do not find any significant difference between **I** and **II** at the SwarmChem level. This may be the length of run though as the transfers will have very

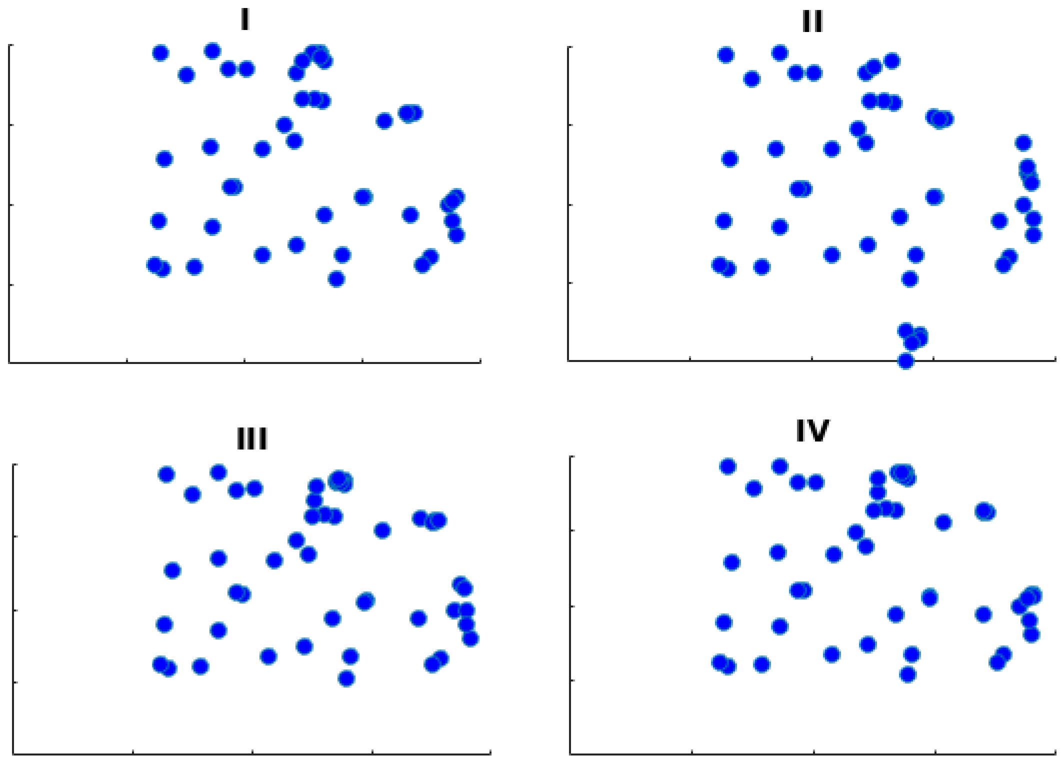


Figure 8.6: Image of each swarm system at the end of a run which started from the same initial conditions. Swarms are depicted in the full 2000 × 2000 (units arbitrary) space

little chance to have an effect in 100 generations. We see no distinction between our two pure SwarmChem systems either. Further research should look at the effect over longer runs.

This gives us distinction between systems with redifferentiation and without. This shows that a multilevel system such as system **I** and **II** can produce distinctive behaviour in SwarmChem. While System **I** does meet all the criteria of a category D system, more work is needed to show what effect this has as opposed to **II** which is only a category C system.

8.5 Conclusions and Further Work

We have introduced the idea of modular graph based descriptions of AChems that allow 'plug and play' composition. The modular nature of these representations allows us to build and implement a wide variety of AChems from a single set of components. We can expand the number of AChems simply by by expanding the number of modular components available to us. We have implemented a general method of composition using indirect communication for environment orientation (Hoverd and Stepney, 2009) at the level of our system graphs.

This approach enables us to use existing AChems to explore new questions with minimal new code required. As well as new systems, this serves to expand the implementations and capabilities of the individual AChems.

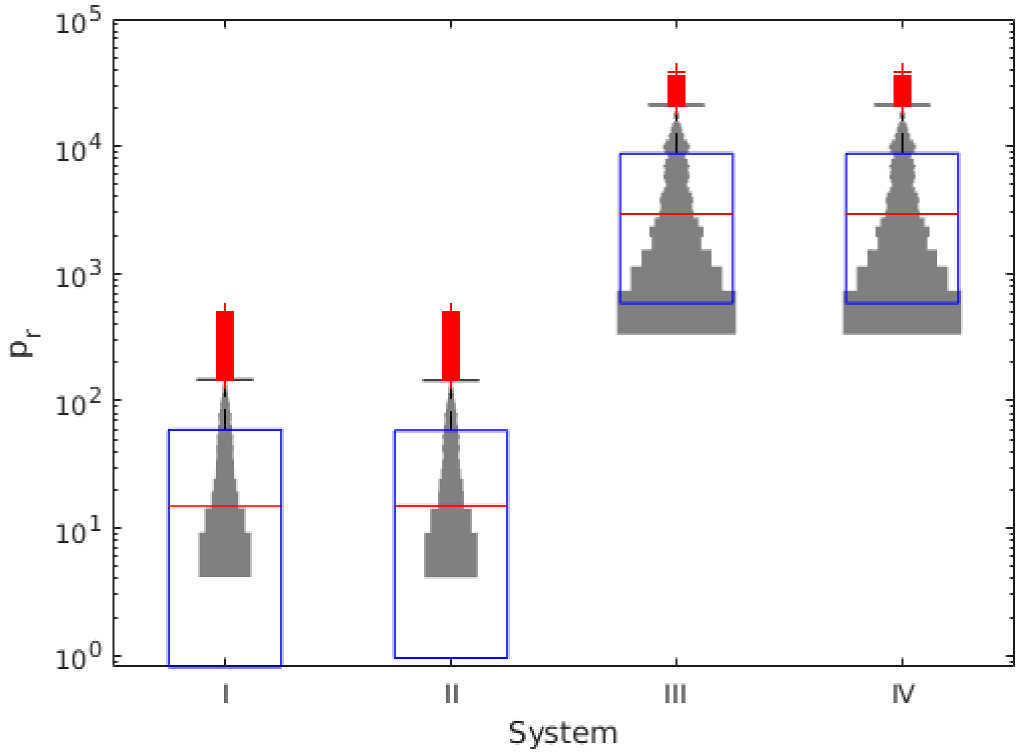


Figure 8.7: Log scaled final position of each agent to its neighbours relative to the agent's perception distance

In this work we chose to nest JA AChem inside of SwarmChem, as this seems a natural choice of combinations. Further work could investigate different forms of nesting, such as inverse nestings and side-by-side combinations of these two AChems, or with entirely different AChems.

Chapter 9

Discussion

9.1 Contribution

9.1.1 Algebraic approach

One of the tangible contributions of this work is the approach itself. The method of using algebraic structures and axioms in our design and implementation of everything, from our framework itself to how we assign a probability of linking to a reaction, has helped to shape this work, and provides a guide for further work to take a similar approach. The advantage of this approach is that it helps to prevent bottom up work from over-designing. It does this by helping to generate a solid and complete structure based on the simple properties we derive, so there is less temptation to "add-on" to the system. If we have to add-on to get a particular behaviour then we are designing it in to our system. The algebra makes that very visible in a system, and suggests that either that behaviour does not work how we think, or we have the wrong algebra if it does not emerge for the system without add-ons.

9.1.1.1 Axiomatic approach to JA AChem

The axiomatic approach taken in designing JA AChem has allowed us to be very clear about our design decisions. Thus we have a system which may contain isomers, and has a elements as categorisation of our atoms. These are both features we have designed for, but in future work their implications for the behaviour of the system can be explored.

The other advantage of this approach is that it lead us so easily to the Jordan Algebra itself and the representation using Hermitian matrices, both of which have interesting implications and properties. It provided us with a product operation that can be applied to any number of particles at a time to produce a new particle whose matrix has the same dimensions still. This helps to keep the computations for this system tractable.

9.1.1.2 Formalisation of psf

The formal definition of a way of handling probabilistic processes in this work will ensure greater consistency in future descriptions of artificial chemistries. It also provides a direct manner in which to derive intended or designed cause-and-effect relations from those that have more inadvertent effects. The probability spawning functions have direct information from well-defined sources which have a well-defined effect on the likelihood of an action occurring. This is traceable. If we find other correlations we can then determined that they are “emergent” from the system.

The definition of a semi-ring structure over these psfs gives us a simpler definition for more complex interactions. It allows us to define many separate influences and then combine them. This also has potential for providing a structure around automatically generating psfs for automatically generated nodes.

9.1.2 Static Graph MetaChem

The static graph MetaChem developed here is a first step towards standardised metrics for artificial chemistries. It also provides a move forward in designing artificial chemistries. As we gather more descriptions in this framework we can begin make use of parts of existing descriptions in new systems. This will allow more efficient design of artificial chemistries. It should also mean we can start to look at standardizing output values from artificial chemistries to make use of standard visualizations and reporting of results. All of this will assist in further development of artificial systems.

In particular the use of modularised structured nodes with defined functions should allow us to generate new nodes easily. Algebraic structures around these functions such as the semi-ring of psfs shown here will make it possible to combine existing functions and algorithms into nodes to translate them into our framework. In turn they will also allow us to add to our artificial chemistries without needing large changes to our systems by modifying existing functions.

9.1.3 Developing JA AChem

We have developed a rigorously designed artificial chemistry. We have done so in such a way that we were able to take advantage of the pre-existing Jordan Algebra to make our artificial chemistry axiomatically sound. This produced a system with high combinatorics and interesting particle structures.

We have also analysed the system and shown that the arbitrary choice of probability spawning function could be changed in order to tune the system. This is a good demonstration of the general power of probability spawning functions and their uses in artificial chemistries. In particular using the same function in both our linking and decomposition functions allowed for large variation in the systems behaviours.

9.1.4 MetaChems for use with AChems

The most direct and immediate contribution of this work is the potential metachem and the link described for nestedchem open up in the field of artificial chemistry. The ability to join up systems and use modularity to share parts of algorithms could provide, after more development, significant speed ups in how long it takes to build a new AChem and what is possible in the course of further PhD projects, which is the time frame under which much research takes place.

Even more than this is the potential that we have modularity and clear designation of particles and environmental properties. This means we can begin to design analysis tools, visualisation tools, and metrics that can be used across similar systems. We can provide a set of particles and their position to a visualiser regardless of our system's other properties, and use it to produce images and video of the AChem's motions. We can look at writing a more general purpose proximity based analysis for higher level object identification such as that used in more recent Swarm Chemistry work (Sayama, 2018a,c).

9.2 Further Work

9.2.1 Dynamic MetaChem

Our static graph MetaChem can be treated as a graph, meaning once we have a set of possible nodes we can define a set of rules for generating a graph. This can be a rule set intended to produce a particular graph or multiple possible graphs. If we tied this process of graph production through rule to the implementation and evaluation of the graphs as programs we can then generate our graph as it runs. This means our system can grow at run time and could grow differently dependent on differences in our particles and variables.

This could allow us to have a system capable of self-reflection and change at run time. This opens new possibilities for transitions towards open-ended evolution, as we can build systems capable of reacting to new emergent objects or behaviours if they can be identified. For example, if a system identified a set of objects within itself, it could then attempt to model those objects at a higher level and improve that model with information from the original low-level implementation.

9.2.2 Evolving MetaChem

Our MetaChem can then itself be treated as an artificial chemistry. It has nodes as particles with links between them. With a graph rule set to generate them we can start to see reactions. We could run an artificial chemistry of artificial chemistries. Through this method we could evolve an artificial chemistry.

If we then take one step further, make and consider instead our set of particles to be the graph *rules* that generate our systems and reactions to change those rules, then we can evolve how our graphs form. This would allow systems not only capable of changing at run time but of changing their basic components and how they can run during their execution. With the correct initial rule set and reactions this could allow for the production of a completely unexpected artificial chemistry with as little design bias as possible.

9.2.3 Jordan Algebra Artificial Chemistry

Jordan Algebra Artificial Chemistry is a simple system with a lot of potential to it. There are many possible avenues for future work with this system, from looking at interesting initial conditions, to performing longer runs and analysing the results looking for polymers or monomers. A reaction-based analysis would be interesting as well in search of any reaction networks. Beyond this there is the potential to look for other uses for the chemistry, for example it could be used for reservoir computing. By feeding an input in that interacts with chemistry, possibly through causing a flow, reaction or modifying reactions based on proximity to input and then observing an output or set of outputs somewhere in the system.

9.2.4 Combining AChems

We have only touched on one option for combining artificial chemistries here, but the potential is much broader. Firstly we could try combining Jordan Algebra AChem with other spatial AChems to see if we get different results from a similar combination. Equally, nesting different AChems inside Swarm Chem could be attempted, to see if that would have different effects on behaviour and densities of swarms. Equally, completely new and different hierarchical AChem combinations could be tried.

As well as nesting systems, there are also mixed systems and joint systems to consider. Mixed systems would be systems sharing tanks or spatial environment with a new interaction added between the different types of particles. Joint systems would work with a combined particle made up of a particle from each system,. Which system is used in an interaction would need defining in this case but may be probabilistic, depend on the kind of reaction, or be defined by the result of the interaction. An example might be to use the reaction rule that produces the stronger link.

9.2.5 Scaling and databases

Now that we have a framework for our systems, part of our further work must be to scale up our systems. Artificial chemistries, much like other artificial life systems, should be working over very large collections of particles and at very long timescales to allow for complex emergence. However for us as designers this poses a complex trade off. If we wish to have such large systems then we must make our components and calculations highly tractable to manage within the memory limitations of systems available to us.

In the age of big data analysis there is a set of tools and skills that are being fast developed in computer science that seem to be oddly overlooked in artificial life research.

Databases are commonly used to store and analyze even relatively small datasets. A business will start to use a database with only a few hundred customers, while artificial life systems looking to contain thousands or more agents or particles still seek to operate without these storage structures. These could allow us to work with far broader systems and run them for longer while still having timely access to the results for analysis. They would allow us to have complicated agents and particles, and only deal with those aspects for a particular calculation.

While there is some added complexity to using databases with these systems those which are properly modularised and built around a sensible framework such as MetaChem should only need a database interface creating that would then take all issues with this technology away from the user of any particular system. In large artificial chemistry systems this means only entities currently being analyzed or edited would need to be in the computers memory rather than stored externally in the database.

9.2.6 Analysis of AChems and graph based databases

If we are going to make use of databases we need to consider the options available to us. The classic relational table database would solve the issues of data storage and access described above, but the more recently emerging graph based databases have some advantages. These sorts of databases are very flexible, without the fixed table structures. This means that new types of objects and connections are always possible to store. In the case of an implementation in MetaChem this means we can write an interface with our database into our read(), push() and pull() functions at a very low level. This would take away the need for any direct interaction with the database, because the actual structure of the particles would not change the push(), pull() interface to the database. The graph based databases are focused on the relationships between the entries, and in most cases we can even infer further relationships. This is a good fit for artificial life and the search for emergent entities in the interactions between the elements. In particular this fits well with the the new research into identifying these larger entities through clustering and mutual perception in Swarm Chemistry (Sayama, 2018a).

Appendix: Representative matrices of elements

Here we provide a list of the representative matrices of elements referred to and used in 5. They are provided in no particular order. These are the lowest weight atomic matrices for each possible set of eigenvalues and trace. They are named based on their eigenvalues (upper case letters assigned as they are generated) and trace(lower case letters assigned for each new trace generated).

Element	Matrix	Element	Matrix
Yu	$\begin{pmatrix} -1 & 1+i & i \\ 1-i & -1 & i \\ -i & -i & 0 \end{pmatrix}$	Nf	$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$
BBw	$\begin{pmatrix} 1 & i & i \\ -i & 1 & i \\ -i & -i & 0 \end{pmatrix}$	Tp	$\begin{pmatrix} 0 & 1-i & 1+i \\ 1+i & 1 & i \\ 1-i & -i & 1 \end{pmatrix}$
VVmm	$\begin{pmatrix} -1 & 1-i & 1+i \\ 1+i & 1 & i \\ 1-I & -i & 1 \end{pmatrix}$	Zn	$\begin{pmatrix} 1 & 1+i & i \\ 1-i & 1 & 0 \\ -i & 0 & 0 \end{pmatrix}$
IIn	$\begin{pmatrix} 0 & i & i \\ -i & 1 & 0 \\ -i & 0 & 1 \end{pmatrix}$	Ga	$\begin{pmatrix} 1 & 1+i & 1+i \\ 1-i & 1 & 0 \\ 1-i & 0 & 1 \end{pmatrix}$
LLLtt	$\begin{pmatrix} 0 & 1-i & 1+i \\ 1+i & -1 & i \\ 1-i & -i & 0 \end{pmatrix}$	YYoo	$\begin{pmatrix} 1 & i & i \\ -i & -1 & 0 \\ -i & 0 & 1 \end{pmatrix}$
Qm	$\begin{pmatrix} 1 & 1+i & 1+i \\ 1-i & 1 & i \\ 1-i & -i & 0 \end{pmatrix}$	ZZi	$\begin{pmatrix} 1 & 1-i & 1+i \\ 1+i & -1 & i \\ 1-i & -i & -1 \end{pmatrix}$
Kj	$\begin{pmatrix} 1 & i & i \\ -i & 1 & 0 \\ -i & 0 & 1 \end{pmatrix}$	Fcc	$\begin{pmatrix} 1 & i & 0 \\ -i & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix}$
Vr	$\begin{pmatrix} 0 & 1+i & 1+i \\ 1-i & 1 & i \\ 1-i & -i & 1 \end{pmatrix}$	Hg	$\begin{pmatrix} -1 & 1+i & i \\ 1-i & -1 & i \\ -i & -i & -1 \end{pmatrix}$

Element	Matrix	Element	Matrix
KKcc	$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix}$	So	$\begin{pmatrix} -1 & 1+i & 1+i \\ 1-i & -1 & 1+i \\ 1-i & 1-i & 0 \end{pmatrix}$
Aa	$\begin{pmatrix} 1 & 1+i & 1+i \\ 1-i & 1 & 1+i \\ 1-i & 1-i & 1 \end{pmatrix}$	Pl	$\begin{pmatrix} 1 & 1-i & 1+i \\ 1+i & 1 & i \\ 1-i & -i & 0 \end{pmatrix}$
JJdd	$\begin{pmatrix} 1 & i & 0 \\ -i & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}$	GGGa	$\begin{pmatrix} 0 & 1+i & 1+i \\ 1-i & 1 & i \\ 1-i & -i & 0 \end{pmatrix}$
Dd	$\begin{pmatrix} 1 & 1+i & i \\ 1-i & 1 & i \\ -i & -i & 1 \end{pmatrix}$	MMMuu	$\begin{pmatrix} 0 & 1+i & i \\ 1-i & 0 & 0 \\ -i & 0 & 1 \end{pmatrix}$
OOhh	$\begin{pmatrix} 1 & 1+i & i \\ 1-i & 1 & i \\ -i & -i & -1 \end{pmatrix}$	Xt	$\begin{pmatrix} 1 & 1+i & 1+i \\ 1-i & 1 & 0 \\ 1-i & 0 & 0 \end{pmatrix}$
NNNvv	$\begin{pmatrix} 0 & i & i \\ -i & 1 & 0 \\ -i & 0 & 0 \end{pmatrix}$	QQa	$\begin{pmatrix} -1 & 1-i & 1+i \\ 1+i & 1 & i \\ 1-i & -i & 1 \end{pmatrix}$
Ws	$\begin{pmatrix} 1 & 1 & i \\ 1 & 1 & i \\ -i & -i & 0 \end{pmatrix}$	DDDh	$\begin{pmatrix} -1 & i & i \\ -i & 1 & 0 \\ -i & 0 & 1 \end{pmatrix}$
IIIqq	$\begin{pmatrix} 0 & 1-i & 1+i \\ 1+i & 1 & i \\ 1-i & -i & 0 \end{pmatrix}$	LLee	$\begin{pmatrix} -1 & 1+i & 1+i \\ 1-i & 1 & 1+i \\ 1-i & 1-i & 1 \end{pmatrix}$
Ih	$\begin{pmatrix} 1 & i & i \\ -i & 1 & i \\ -i & -i & 1 \end{pmatrix}$	TTb	$\begin{pmatrix} -1 & 1+i & i \\ 1-i & -1 & 0 \\ -i & 0 & 1 \end{pmatrix}$
Cc	$\begin{pmatrix} 1 & 1+i & 1+i \\ 1-i & 1 & i \\ 1-i & -i & 1 \end{pmatrix}$	DDy	$\begin{pmatrix} 1 & i & i \\ -i & 1 & 0 \\ -i & 0 & 0 \end{pmatrix}$
Ff	$\begin{pmatrix} 1 & 1 & i \\ 1 & 1 & i \\ -i & -i & 1 \end{pmatrix}$	EEEj	$\begin{pmatrix} -1 & i & 0 \\ -i & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$
Ee	$\begin{pmatrix} -1 & 1+i & 1+i \\ 1-i & -1 & 1+i \\ 1-i & 1-i & -1 \end{pmatrix}$	NNgg	$\begin{pmatrix} 1 & 1+i & 1+i \\ 1-i & -1 & i \\ 1-i & -i & 1 \end{pmatrix}$
GGbb	$\begin{pmatrix} 1 & 1+i & 0 \\ 1-i & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix}$	Rn	$\begin{pmatrix} 1 & 1+i & i \\ 1-i & 1 & i \\ -i & -i & 0 \end{pmatrix}$
Ji	$\begin{pmatrix} -1 & 1-i & 1+i \\ 1+i & -1 & i \\ 1-i & -i & -1 \end{pmatrix}$	Ok	$\begin{pmatrix} 1 & 1+i & 1+i \\ 1-i & 1 & 1+i \\ 1-i & 1-i & 0 \end{pmatrix}$

Element	Matrix	Element	Matrix
HHl	$\begin{pmatrix} 0 & 1+i & i \\ 1-i & 1 & 0 \\ -i & 0 & 1 \end{pmatrix}$	UUll	$\begin{pmatrix} -1 & 1+i & i \\ 1-i & -1 & i \\ -i & -i & 1 \end{pmatrix}$
Bb	$\begin{pmatrix} 1 & 1-i & 1+i \\ 1+i & 1 & i \\ 1-i & -i & 1 \end{pmatrix}$	FFFa	$\begin{pmatrix} 1 & 1-i & 1+i \\ 1+i & 0 & i \\ 1-i & -i & 0 \end{pmatrix}$
Uq	$\begin{pmatrix} 1 & 1+i & i \\ 1-i & 0 & i \\ -i & -i & 1 \end{pmatrix}$	WWnn	$\begin{pmatrix} -1 & i & i \\ -i & 1 & i \\ -i & -i & 1 \end{pmatrix}$
La	$\begin{pmatrix} -1 & 1 & i \\ 1 & -1 & i \\ -i & -i & -1 \end{pmatrix}$	Tww	$\begin{pmatrix} 1 & i & 0 \\ -i & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}$
AAv	$\begin{pmatrix} -1 & 1-i & 1+i \\ 1+i & -1 & i \\ 1-i & -i & 0 \end{pmatrix}$	EEf	$\begin{pmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix}$
RRjj	$\begin{pmatrix} -1 & 1+i & i \\ 1-i & 1 & i \\ -i & -i & 1 \end{pmatrix}$	CCx	$\begin{pmatrix} -1 & 1+i & i \\ 1-i & 0 & i \\ -i & -i & -1 \end{pmatrix}$
JJJrr	$\begin{pmatrix} -1 & 1+i & i \\ 1-i & 0 & i \\ -i & -i & 0 \end{pmatrix}$	MMff	$\begin{pmatrix} 1 & 1-i & 1+i \\ 1+i & -1 & i \\ 1-i & -i & 1 \end{pmatrix}$
FFaa	$\begin{pmatrix} -1 & 1 & i \\ 1 & -1 & i \\ -i & -i & 0 \end{pmatrix}$	CCCf	$\begin{pmatrix} -1 & i & 0 \\ -i & -1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$
EEz	$\begin{pmatrix} 0 & 1-i & 1+i \\ 1+i & -1 & i \\ 1-i & -i & -1 \end{pmatrix}$	AAAa	$\begin{pmatrix} -1 & 1+i & i \\ 1-i & 1 & 0 \\ -i & 0 & 1 \end{pmatrix}$
PPii	$\begin{pmatrix} -1 & 1+i & 1+i \\ 1-i & -1 & 1+i \\ 1-i & 1-i & 1 \end{pmatrix}$	BBBj	$\begin{pmatrix} -1 & 1+i & 0 \\ 1-i & -1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$
HHHpp	$\begin{pmatrix} 1 & 1+i & i \\ 1-i & 0 & i \\ -i & -i & 0 \end{pmatrix}$	SSkk	$\begin{pmatrix} -1 & 1+i & 1+i \\ 1-i & 1 & i \\ 1-i & -i & 1 \end{pmatrix}$
XXc	$\begin{pmatrix} -1 & 1+i & i \\ 1-i & 1 & i \\ -i & -i-1 & \end{pmatrix}$	Mf	$\begin{pmatrix} 1 & i & 0 \\ -i & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$
KKKss	$\begin{pmatrix} 1 & 1+i & i \\ 1-i & 0 & 0 \\ -i & 0 & 0 \end{pmatrix}$		

List of References

Acerbi, A. and Nolfi, S. (2007). Social learning and cultural evolution in embodied and situated agents. In *2007 IEEE Symposium on Artificial Life*, pages 333–340. ieeexplore.ieee.org.

Ackley, D. H. (2018). Digital protocells with dynamic size, position, and topology. *The 2018 Conference on Artificial Life: A Hybrid of the European Conference on Artificial Life (ECAL) and the International Conference on the Synthesis and Simulation of Living Systems (ALIFE)*, pages 83–90.

Adams, A., Zenil, H., Davies, P. C. W., and Walker, S. I. (2017). Formal definitions of unbounded evolution and innovation reveal universal mechanisms for Open-Ended evolution in dynamical systems. *Sci. Rep.*, 7(1):997.

Alatas, B. (2011). ACROA: Artificial chemical reaction optimization algorithm for global optimization. *Expert Syst. Appl.*, 38(10):13170–13180.

Allen, C., Smit, I., and Wallach, W. (2005). Artificial morality: Top-down, bottom-up, and hybrid approaches. *Ethics Inf. Technol.*, 7(3):149–155.

Andrews, P. S. and Stepney, S. (2014). Using CoSMoS to reverse engineer a domain model for aevol. In *Proceedings of the 2014 Workshop on Complex Systems Modelling and Simulation, New York, USA*, pages 61–79. www-users.cs.york.ac.uk.

Andrews, S. S. and Bray, D. (2004). Stochastic simulation of chemical reactions with spatial resolution and single molecule detail. *Phys. Biol.*, 1(3-4):137–151.

Anet, F. A. L. (2004). The place of metabolism in the origin of life. *Curr. Opin. Chem. Biol.*

Banzhaf, W. (1993). Self-replicating sequences of binary numbers. foundations II: Strings of length n=4. *Biol. Cybern.*, 69(4):275–281.

Banzhaf, W., Baumgaertner, B., Beslon, G., Doursat, R., Foster, J. A., McMullin, B., de Melo, V. V., Miconi, T., Spector, L., Stepney, S., and White, R. (2016). Defining and simulating open-ended novelty: requirements, guidelines, and challenges. *Theory Biosci.*, 135(3):131–161.

Barr, M. and Wells, C. (1990). *Category theory for computing science*, volume 49. Prentice Hall New York.

Bedau, M. A., McCaskill, J. S., Packard, N. H., Rasmussen, S., Adami, C., Green, D. G., Ikegami, T., Kaneko, K., and Ray, T. S. (2000). Open problems in artificial life. *Artif. Life*, 6(4):363–376.

Bistarelli, S., Montanari, U., and Rossi, F. (1997). Semiring-based constraint satisfaction and optimization. *J. ACM*, 44(2):201–236.

Branke, J., Kaussler, T., Smidt, C., and Schmeck, H. (2000). A multi-population approach to dynamic optimization problems. In *Evolutionary Design and Manufacture*, pages 299–307. Springer London.

Carter, R. J., Wiesner, K., and Mann, S. (2018). Emergence and dynamics of self-producing information niches as a step towards pre-evolutionary organization. *J. R. Soc. Interface*, 15(138).

Clark, E. B., Hickinbotham, S. J., and Stepney, S. (2017). Semantic closure demonstrated by the evolution of a universal constructor architecture in an artificial chemistry. *J. R. Soc. Interface*, 14(130).

Dittrich, P., Ziegler, J., and Banzhaf, W. (2001). Artificial Chemistries—A review. *Artif. Life*, 7(3):225–275.

Ehrig, H., Rensink, A., Rozenberg, G., and Schürr, A. (2010). Graph transformations. *Lect. Notes Comput. Sci.*, 6372.

Elena, S. F. and Lenski, R. E. (2003). Evolution experiments with microorganisms: the dynamics and genetic bases of adaptation. *Nat. Rev. Genet.*, 4(6):457–469.

Etherington, I. M. H. (1941). II.—Non-Associative algebra and the symbolism of genetics. *Proceedings of the Royal Society of Edinburgh, Section B: Biological Sciences*, 61(1):24–42.

Faulconbridge, A. (2011). *RBN-World: Sub-Symbolic Artificial Chemistry for Artificial Life.* PhD thesis, University of York.

Faulconbridge, A., Stepney, S., Miller, J. F., and Caves, L. S. D. (2009). RBN-World. In *Advances in Artificial Life. Darwin Meets von Neumann*, Lecture Notes in Computer Science, pages 377–384. Springer Berlin Heidelberg.

Faulconbridge, A., Stepney, S., Miller, J. F., and Caves, L. S. D. (2010). RBN-World-The hunt for a rich AChem. In *ALife*, pages 261–268.

Faulkner, P., Krastev, M., Sebald, A., and Stepney, S. (2018). Sub-Symbolic artificial chemistries. In Stepney, S. and Adamatzky, A., editors, *Inspired by Nature*, pages 287–322. Springer International Publishing, Cham.

Faulkner, P., Sebald, A., and Stepney, S. (2016). Jordan algebra AChems: Exploiting mathematical richness for open ended design. In *Proceedings of the Artificial Life Conference 2016*, pages 582–589. MIT Press.

Faulkner, P., Sebald, A., and Stepney, S. (2017). Tuning Jordan algebra artificial chemistries with probability spawning functions. In *European Conference on Artificial Life 2017*, pages 497–504. MIT Press.

Froese, T., Virgo, N., and Ikegami, T. (2014). Motility at the origin of life: its characterization and a model. *Artif. Life*, 20(1):55–76.

Gillespie, D. T. (1977). Exact stochastic simulation of coupled chemical reactions. *J. Phys. Chem.*, 81(25):2340–2361.

Głazek, K. (2002). *A Guide to the Literature on Semirings and their Applications in Mathematics and Information Sciences: With Complete Bibliography.* Springer Netherlands.

Goldsby, H. J., Dornhaus, A., Kerr, B., and Ofria, C. (2012). Task-switching costs promote the evolution of division of labor and shifts in individuality. *Proceedings of the National Academy of Sciences*, 109(34):13686–13691.

Gonzalez, S. and Martinez, C. (2003). Nonassociative algebras: Some applications. *Rev. Mat. Iberoamericana.*

Haag, R. and Kastler, D. (1964). An algebraic approach to quantum field theory. *J. Math. Phys.*, 5(7):848–861.

Hayes-Roth, B. (1988). A blackboard architecture for control. In Bond, A. H. and Gasser, L., editors, *Readings in Distributed Artificial Intelligence*, pages 505–540. Morgan Kaufmann.

Hickinbotham, S., Clark, E., Nellis, A., Stepney, S., Clarke, T., and Young, P. (2016). Maximizing the adjacent possible in automata chemistries. *Artif. Life*, 22(1):49–75.

Hickinbotham, S., Clark, E., Stepney, S., Clarke, T., Nellis, A., Pay, M., and Young, P. (2010). Specification of the stringmol chemical programming language version 0.2. Technical report, Technical Report YCS-2010-458, Univ. of York.

Higgs, P. G. and Lehman, N. (2015). The RNA world: molecular cooperation at the origins of life. *Nat. Rev. Genet.*, 16(1):7–17.

Hoverd, T. and Stepney, S. (2009). Environment orientation: an architecture for simulating complex systems. In *Proceedings of the 2009 Workshop on Complex Systems Modelling and Simulation*, pages 67–82. pdfs.semanticscholar.org.

Hutton, T. J. (2002). Evolvable self-replicating molecules in an artificial chemistry. *Artif. Life*, 8(4):341–356.

Hutton, T. J. (2003). Information-replicating molecules with programmable enzymes. In *Proceedings of Sixth International Conference on Humans and Computers*, pages 170–175. sq3.org.uk.

Hutton, T. J. (2007). Evolvable self-reproducing cells in a two-dimensional artificial chemistry. *Artif. Life*, 13(1):11–30.

Keef, T. and Twarock, R. (2009). Affine extensions of the icosahedral group with applications to the three-dimensional organisation of simple viruses. *J. Math. Biol.*, 59(3):287–313.

Knibbe, C., Mazet, O., Chaudier, F., Fayard, J.-M., and Beslon, G. (2007). Evolutionary coupling between the deleteriousness of gene mutations and the amount of non-coding sequences. *J. Theor. Biol.*, 244(4):621–630.

Krastev, M., Sebald, A., and Stepney, S. (2016). Emergent bonding properties in the spiky RBN AChem. *ALife 2016, Cancun*, pages 600–607.

Krastev, M., Sebald, A., and Stepney, S. (2017). Functional grouping analysis of varying reactor types in the Spiky-RBN AChem. In *European Conference on Artificial Life 2017*, pages 247–254. MIT Press.

Liu, J. and Hu, H. (2010). Biological inspiration: From carangiform fish to multi-joint robotic fish. *J. Bionic Eng.*, 7(1):35–48.

Liu, Y. (2018). The artificial ecosystem: number soup (part II). *arXiv preprint arXiv:1801.04916*.

Lucht, M. W. (2012). Size selection and adaptive evolution in an artificial chemistry. *Artif. Life*, 18(2):143–163.

Madina, D., Ono, N., and Ikegami, T. (2003). Cellular evolution in a 3D lattice artificial chemistry. In *Advances in Artificial Life*, Lecture Notes in Computer Science, pages 59–68. Springer Berlin Heidelberg.

Maley, C. C. (1999). Four steps toward open-ended evolution. In *Proceedings of the 1st Annual Conference on Genetic and Evolutionary Computation - Volume 2*, GECCO'99, pages 1336–1343, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

McCrimmon, K. (1978). Jordan algebras and their applications. *Bull. Am. Math. Soc.*, 84(4):612–627.

McCrimmon, K. (2006). *A taste of Jordan algebras.* Springer Science & Business Media.

Michalski, J. R., Onstott, T. C., Mojzsis, S. J., Mustard, J., Chan, Q. H. S., Niles, P. B., and Johnson, S. S. (2018). The martian subsurface as a potential window into the origin of life. *Nat. Geosci.*, 11(1):21–26.

Muller, P. (1994). Glossary of terms used in physical organic chemistry (IUPAC recommendations 1994). *J. Macromol. Sci. Part A Pure Appl. Chem.*, 66(5).

Nguyen, T. T., Yang, S., and Branke, J. (2012). Evolutionary dynamic optimization: A survey of the state of the art. *Swarm and Evolutionary Computation*, 6:1–24.

Nitash, C. G., LaBar, T., Hintze, A., and Adami, C. (2017). Origin of life in a digital microcosm. *Phil. Trans. R. Soc. A*.

Ofria, C. and Wilke, C. O. (2004). Avida: a software platform for research in computational evolutionary biology. *Artif. Life*, 10(2):191–229.

Ono, N. and Ikegami, T. (2001). Artificial chemistry: Computational studies on the emergence of Self-Reproducing units. In *Advances in Artificial Life*, Lecture Notes in Computer Science, pages 186–195. Springer Berlin Heidelberg.

ProofWiki (2011). Proof that Minimum and Maximum are Distributive. proofwiki.org/wiki/Max_and_Min_Distributive.

Pross, A. (2004). Causation and the origin of life. metabolism or replication first? *Orig. Life Evol. Biosph.*, 34(3):307–321.

Rainford, P. F., Sebald, A., and Stepney, S. (2018). Modular combinations of artificial chemistries. *The 2018 Conference on Artificial Life: A Hybrid of the European Conference on Artificial Life (ECAL) and the International Conference on the Synthesis and Simulation of Living Systems (ALIFE)*, pages 361–367.

Reed, M. (1997). Algebraic structure of genetic inheritance. *Bull. Am. Math. Soc.*, 34(2):107–130.

Reynolds, C. W. (1987). Flocks, herds and schools: A distributed behavioral model. In *Proceedings of the 14th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '87, pages 25–34, New York, NY, USA. ACM.

Ruiz-Mirazo, K., Briones, C., and de la Escosura, A. (2017). Chemical roots of biological evolution: the origins of life as a process of development of autonomous functional systems. *Open Biol.*, 7(4).

Sabinin, L., Sbitneva, L., and Shestakov, I. (2006). *Non-associative algebra and its applications.* taylorfrancis.com.

Sayama, H. (2009). Swarm chemistry. *Artif. Life*, 15(1):105–114.

Sayama, H. (2010a). Robust morphogenesis of robotic swarms [application notes]. *IEEE Comput. Intell. Mag.*, 5(3):43–49.

Sayama, H. (2010b). Swarm chemistry evolving. In *ALIFE*, pages 32–36. pdfs.semanticscholar.org.

Sayama, H. (2011). Seeking open-ended evolution in swarm chemistry. In *2011 IEEE Symposium on Artificial Life (ALIFE)*, pages 186–193.

Sayama, H. (2012). Evolutionary swarm chemistry in three-dimensions. *Artif. Life*, 13:576–577.

Sayama, H. (2018a). Cardinality leap for Open-Ended evolution: Theoretical consideration and demonstration by "hash chemistry". *arXiv preprint arXiv:1806.06628*.

Sayama, H. (2018b). Complexity, development, and evolution in morphogenetic collective systems. *arXiv preprint arXiv:1801.02086*.

Sayama, H. (2018c). Seeking Open-Ended evolution in swarm chemistry II: Analyzing Long-Term dynamics via automated object harvesting. *arXiv preprint arXiv:1804.03304*.

Shannon, C. E. (1940). *An algebra for theoretical genetics*. PhD thesis, Massachusetts Institute of Technology.

Soros, L. (2018). *Necessary Conditions for Open-Ended Evolution*. PhD thesis, University of Central Florida.

Soula, H. A. (2016). Generalized stochastic simulation algorithm for artificial chemistry. *Artificial Life Conference 2016*.

Stewart, J. (1997). Evolutionary transitions and artificial life. *Artif. Life*, 3(2):101–120.

Suzuki, H., Ono, N., and Yuta, K. (2003). Several necessary conditions for the evolution of complex forms of life in an artificial environment. *Artif. Life*, 9(2):153–174.

Szostak, J. W. (2017). The origin of life on earth and the design of alternative life forms. *Mol. Front. J.*, 01(02):121–131.

Turgut, A. E., Çelikkanat, H., Gökçe, F., and Şahin, E. (2008). Self-organized flocking in mobile robot swarms. *Swarm Intell*, 2(2-4):97–120.

Vargha, A. and Delaney, H. D. (2000). A critique and improvement of the "CL" common language effect size statistics of McGraw and wong. *J. Educ. Behav. Stat.*, 25(2):101–132.

Vreman, A. W. (2004). An eddy-viscosity subgrid-scale model for turbulent shear flow: Algebraic theory and applications. *Phys. Fluids*, 16(10):3670–3681.

Walde, P. (2010). Building artificial cells and protocell models: experimental approaches with lipid vesicles. *Bioessays*, 32(4):296–303.

Yamamoto, L. (2010). Evaluation of a catalytic search algorithm. In *Nature Inspired Cooperative Strategies for Optimization (NICSO 2010)*, Studies in Computational Intelligence, pages 75–87. Springer Berlin Heidelberg.

Yampolskiy, R. V. (2016). On the origin of synthetic life: attribution of output to a particular algorithm. *Physica Scripta*, 92(1):013002.

Young, T. J. and Neshatian, K. (2013). A constructive artificial chemistry to explore Open-Ended evolution. In *AI 2013: Advances in Artificial Intelligence*, Lecture Notes in Computer Science, pages 228–233. Springer International Publishing.

Young, T. J. and Neshatian, K. (2015). The effect of reactant and product selection strategies on cycle evolution in an artificial chemistry. In *Artificial Life and Computational Intelligence*, Lecture Notes in Computer Science, pages 310–322. Springer International Publishing.