Creating Reproducible Agent Based Models using Formal Methods

Joseph Kehoe

Dublin City University

Abstract. Reproducibility is a problem in Agent Based Modelling. It has proven difficult to replicate results obtained by other researchers so as to confirm their findings. A number of different solutions have been proposed to overcome this issue but the effectiveness of these approaches is still open to debate. Here we propose that formal methods, an approach developed by computer scientists for the production of high integrity systems, can be used to specify even complex Agent Based Models. In order to demonstrate the applicability of Formal Methods we specify Sugarscape, a well known Agent Based Social Simulation, using the Z Notation. Our specification uncovers many ambiguities in the original definition of Sugarscape thus demonstrating the effectiveness of this approach and providing a reference specification of Sugarscape for researchers to use.

Acknowledgements

The Institute of Technology Carlow provided partial support for this work.

1 Introduction

Although Agent Based Modelling (ABM) is becoming more popular across many disciplines there remain questions about its applicability [30]. One of the main issues is that of replication (or reproduction) of results. It is proving difficult for researchers to reproduce the results claimed by other researchers. This is known as the replication problem. Efforts are currently underway to tackle this issue by improving ABM specifications.

In Computer Science there is a similar problem, that of writing specifications precise enough to allow system developers to produce an implementation that they can be confident matches the specification. The most precise approach is based in mathematics and termed Formal Methods. A formal specification can be proven to have (or not have) certain (un)desirable properties. It is also possible to mathematically verify or prove that an implementation matches a specification. Formal methods have found a niche in high integrity software development where errors cannot be tolerated (due to, for example, the risk of loss of life).

We propose using formal methods as a way of writing reproducible ABM specifications. To this end we formally specified Sugarscape, a well known and large Agent Based Social Simulation (ABSS[1]), using the Z notation to see if this revealed any inconsistencies in the its original definition.

We found a number of ambiguities in the rule definitions and also discovered a range of missing information leading to incomplete rule definition. We have placed the entire specification online where each rule is formal specified and discussed in detail [26]. We have also used this specification to implement our own version of Sugarscape to test out different updating techniques and benchmark algorithm performance [27]. Here we give an overview of the process of formally specifying Sugarscape and provide conclusions on the suitability of this approach.

[1] We will use the terms ABM and ABSS interchangeably in this paper.

1.1 Outline of Paper

We start with a reminder of what the replication problem in ABM is. After this we survey the approaches currently proposed to help overcome this issue. Following on from this we give a brief overview of formal specification in general and Z in particular. Then we give a summary of the Agent Based Social Simulation we have specified, *Sugarscape*. We then demonstrate the efficacy of using formal methods to specify ABSS by listing the issues that the formal specification uncovered. We follow this section with a discussion of the merits of formally specifying ABMs and the possible weaknesses with this approach. Finally we give our conclusions on the efficacy of this approach.

2 The Replication Problem

The replication problem is simply stated: although simulations are developed in quantity and their results peer reviewed and published it has proven difficult for researchers to replicate these results [19,33,11].

Given the nature of ABM, where behaviour is defined in terms of simple agent interactions it might appear, at first glance, that reproducing an ABM defined by another researcher would be easy. However, in reality it has proven very difficult due to a number of factors, for example:

1. The simulation source code is not publicly available;
2. The rules of the model are not defined sufficiently precisely;
3. The updating method (synchronous or asynchronous) is not declared or clearly explained.

If we wish to check that the published results are accurate we must re-implement the simulation ourselves, seldom an easy task. Even then, if our results do not match the original we cannot be sure that it is our implementation that is correct. Modellers need to provide a clear and precise description of the model to work from and this is not always present. There have been a number of attempts to overcome these problems but none have been completely successful.

If the published results of a simulation cannot be replicated then we can have little trust in these results. Being able to repeat or reproduce scientific results is an essential part of science. This can be a particular issue for modellers in the natural sciences who may not be as familiar enough with the computer science required to ensure that their implementation is a correct interpretation of the model.

We posit a new approach, that of using formal methods. Although developed for proving correctness in safety critical systems we will demonstrate that they can be used to tackle the replication problem in ABM.

3 Related Work

3.1 Overview, Design concepts, and Details

Overview, Design concepts and Details (ODD) [16] is a protocol for specifying Individual Based Models (IBM) and/or ABM. ODD was introduced to overcome shortcomings in published IBM definitions, specifically the issue that IBM are not specified precisely enough to allow for replication of the simulations.

The ODD protocol consists of three blocks (Overview, Design concepts and Details) divided into seven sections:

1. Purpose;

2. State variables and Scales;
3. Process overview and scheduling;
4. Design Concepts;
5. Initialisation;
6. Inputs;
7. Submodels.

By enforcing a particular structure on simulation descriptions it (i) makes specifications easier to read and (ii) ensures that modellers do not forget to specify any aspect of the model. ODD was originally tested by 28 modellers within the field of ecology to test its usefulness. Since its introduction it has been used in over 50 publications [17]. The long term aims of ODD are stated as:

> "modellers could describe their IBM on paper using some kind of language that (1) people can understand intuitively, (2) is widely used throughout ecology, (3) provides 'shorthand' conventions that minimize the effort to describe the IBM rigorously and completely, and (4) can be converted directly into an executable simulator without the possibility of programming errors."

These aims seem quite naive from a computer science perspective where similar goals have been attempted for many years without success [5].

ODD has been criticised for not being formal enough [14] and the chasm between the current approach to specifications taken in IBM and those more formal models used in computer science is clear. For example it is stated that it is often the case that the specifications produced by each role in the simulation development process (four are identified: *Thematician, Modeller, Computer Scientist* and *Programmer*) might *stay in the realm of mental models, and never reach materialisation*!

ODD has been extended by adding an algorithmic model to specify the behaviours in the IBM [18]. Although interesting, it is lacking any of the modularisation features used by computing specification languages such as Z [36].

3.2 Model Alignment

An early paper to point out and address this problem [11] concluded that we cannot trust simulation results that have not been replicated. Simulations should be treated like experiments in this regard. They illustrate this by taking a published simulation model and producing two independent implementations of it. They then compared the results of their two independent implementations against the original to find differences in the outcomes. They state that producing two different implementations gave them confidence in their results, where they differed with the original results, that they would not have had with only one implementation.

The process of re-implementing a model they called *model alignment* and they suggest that, as most simulations are not amenable to formal analysis, experimentation is the only route to verification.

While we agree that experimentation is useful it does not solve the problem of specifications that are vague or are missing important information. If we do not agree the rules of the ABM with which we are experimenting it will not help.

3.3 A Model Driven Approach

Following from the conclusion of [11] that experimentation is the only route to verification, [33] list two closely related approaches:

Re-implementation The model is rewritten following the original authors instructions;
Alignment or Docking We implement a conceptually identical model *but* using different tools and different methodologies.

They favour alignment as the better approach and propose using an independent modelling process that can be automatically implemented using a number of different ABM toolkits. This has the advantages that it becomes easier for modellers to provide multiple implementations and does not require in depth programming skills from them.

Such an approach would be useful in that a number of implementations can be compared but it still only gives replication, not reproducibility. We can replicate (copy) their results but it does not allow us to reproduce their results. In other words if there is some error in their approach then *Alignment* will not find that error, only repeat it.

3.4 Executable Papers

Hinsen [21,20,19] identified the problem as belonging to the entire computational science field. His proposed solution is to produce *executable papers*, termed *ActivePapers*. That is, published papers will, as well as containing the usual text, also contain the full simulation model, executable code, source code, etc. all in the correct formats in a single HDF[2] format. Then from this executable paper one could extract all the necessary information required to replicate the simulation.

While he recognises and accepts the difference between replicability and reproducibility in science he states that the current state of affairs is unsatisfactory and a symptom of a lack of exchange between the natural sciences and computer science.

3.5 Data Sharing

A similar proposal to *ActivePapers* (above) is made by [28] for embedding data with existing formats (pdf, etc). Here the suggestion is that no new formats are required and what is missing is only the toolset to allow easy insertion of different data attachments to existing formats. The authors have developed some tools to this end.

All these approaches are useful and give more information to experimenters but all have the same weakness. They only allow us to repeat an experiment exactly as it was done originally. Therefore such approaches do not deal with errors that are hidden in the original experiment. For example, if the updating approach used is the source of the problem then simply copying this approach will give us the same erroneous results.

What is needed is a higher level approach to replication that allows us the freedom to interrogate the original ABM and reproduce the model independently in a verifiable manner.

4 Formal Methods

4.1 Overview

Computer scientists have been trying to solve a problem analogous to replication. The problem is how to be sure that the software being produced matches the specification of the client. We can have confidence in this only if the problem is specified in a precise and unambiguous manner. It has become clear that specifying a problem in a natural language[3] is inappropriate. Natural languages are neither precise nor unambiguous. To overcome these limitations computer scientists

[2] http://www.hdfgroup.org/
[3] e.g. English, Mandarin, German, etc.

sought to base their specifications in mathematics. The process of producing mathematically based specifications is known as formal methods [4,37].

Formal methods are based on strong mathematical foundations usually involving one or more of Logical Calculus, Set and Type theory, Formal Languages, Automata Theory and Program Semantics. The two more popular mathematical approaches are model based, e.g. VDM [25], and process based, e.g. CSP [1].

These approaches place the emphasis on correctness and give benefits such as precision of the system specification and provability of the correctness of the specification and/or implementation. Their main benefit may rest in the discipline they impose on their users, forcing them to explicitly define and think carefully about every part of the system being specified. There are three phases to formally specifying a system, each of which is more difficult than the last:

Specification Precisely define the system in the underlying mathematical formalism;
Refinement Mathematically refine the specification from a high level specification to implementation;
Verification Use the methods proof rules to prove that the implementation correctly interprets the specification and has all the properties we expect (e.g. the safety property[4]).

The most common approach is known as *Formal Methods-lite* where we only proceed with the first step, *Specification*. This still brings many of the benefits of formal specification without the higher overheads (in terms of extra time and mathematical knowledge required) associated with the other two phases. We believe that formal methods-lite is sufficient to solve the replication problem.

4.2 The Z Notation

We chose the Z notation [36] as our specification language. The Z notation is a formal specification language for describing and modelling computer systems. It was first proposed in 1977 by Jean-Raymond Abrial and gained ISO standard accreditation in 2002. It is based on axiomatic set theory, lambda calculus and first order predicate logic. All expressions in Z are typed.

Specification in Z is state-based, that is it is based on a system of states with operations defined in terms of before and after states and the relationship between them. A specification is defined as a series of sequential steps with each step considered to be discrete in nature. This makes Z a good fit for ABMs where simulations are defined as a sequence of atomic steps.

A high level Z specification will be independent of implementation issues and although it is possible to formally refine a Z specification down into computer code this is not necessary. The high level specification only states the before and after states of each operation and any implementation that satisfies these constraints is allowable.

Because Z has been around so long it has evolved into a mature and standardised system that is widely understood within the Formal Methods community. It has a range of software based tools [13] readily available for specifiers that aid with the production of specifications and proving properties of those specifications.

This availability of tools combined with the widespread recognition and maturity of Z alongside its state-based approach makes it a good choice for specifying ABMs. In particular, we have found that the specification of rules in a manner that is completely agnostic as to the implementation approach is a boon. Specifically it allowed us to place no restrictions on what forms of conflict resolution rules are used. It only states the *before* and *after* state of the simulation on application of each rule, not the particular strategy used to get from the before state to the after state.

[4] Safety properties informally require that "something bad will never happen"

5 Sugarscape

Sugarscape [12] is the simulation that demonstrated how ABM could be applied to the social sciences. It remains influential today and almost every major simulation toolkit (Swarm, Repast, Mason and NetLogo) [31,2,23] comes with a partial implementation of Sugarscape that demonstrates that toolkit's approach to simulation.

Currently, social science simulations are starting to embrace concurrency in an effort to allow for bigger, more complete and faster implementations of ABMs. Different concurrency researchers [29,10] have used partial implementations of the Sugarscape model as a testbed for benchmarking different approaches to parallelising ABMs. However although the rules of Sugarscape have been defined in [12] there is no general agreement on their exact meaning [3,15]. These difficulties hamper the ability of researchers to properly compare their approaches, provide complete implementations of Sugarscape or replicate the results of other researchers.

It is worth noting that each of the thirteen rules of Sugarscape can be defined in a paragraph of text and appear at first glance to be simple. This shows how deceptive natural language based specifications are, they appear clear but when different researchers try to implement them cannot agree on their exact meaning.

Originally the rules were stated with an explicit assumption that the underlying implementation would be sequential. Concurrency was simulated through randomisation of the order of each rule application on the individual agents. Models that follow this regime are termed *asynchronous*.

Most of the rules require some form of conflict resolution. We have specified the rules in a manner consistent with the original intention (agents acting concurrently) but independent of any particular approach to how this concurrency is implemented. That is, we have refrained from imposing any specific conflict resolution rules or a specific updating approach.

By formalising the simulation and providing a single precisely defined reference for the rules we can produce a standard definition of Sugarscape. Compliance with this single reference allows proper comparisons to be made between different approaches. It also leaves it open to the implementer to decide what approach to conflict resolution they wish to take. We detected ambiguities present in the current rule definitions, provided precise interpretations, where possible, and flagged unresolvable problems where not.

6 Results of the Specification

We produced a formal specification of sugarscape in Z and made this available as a reference [26]. This specification covers the entire Sugarscape definition with all 13 rules and two resource types (sugar and spice).

We made the decision to restrict the initial specification to one pollution type and one resource type in an effort to guarantee clarity. While the original rules were designed so that they could be extended to arbitrary numbers of resources and pollutants, explicitly specifying an arbitrary number of resources and pollutants would make the specification more difficult to understand and thus more likely to either contain or cause mistakes. This specification extends to about 46 pages in length including embedded explanatory text.

Once we produced a specification for the single resource scenario we extended the specification to a two resource situation (where the resources are known, respectively, as *sugar* and *spice*). This allowed us to specify the final rule, $Trade$, as that rule requires two resources to function.

This allowed for:

1. A simpler and easier to understand specification of the rules that use only one resource (trading clarity against completeness);

2. A complete (but separate) specification for simulations that use two resources.

The downside of this is that each rule has to be specified twice, once for the single resource scenario and once for the two resource scenario. Fortunately a modeller will only need to read one version of the rule depending on the number of resources in the simulation. We do not provide specifications for multiple pollutants as multiple pollutants were never actually implemented in Sugarscape. While specifying multiple pollutants would make for a more general specification there is a trade off in terms of the complexity this would add. If the specification becomes too complex it can make it more difficult to reason about. As multiple pollutants are never used we do not feel this is too much of a limitation.

Similarly we did not provide a specification for more than two resources as we deem the benefits of doing so counterbalanced by both the complexity of the resulting specification and the lack of any requirement to use such a complex simulation for benchmarking purposes. Sugarscape has only ever been implemented with two resource types. Anyone wishing to extend Sugarscape further can use the two resource specification for guidance.

6.1 The Specification Process

A Z specification will generally be broken down into the following steps. First we define any necessary constants. Alongside these constants we define new types, outside of the built-in types provided by Z, that are required for the specification. Following this we specify the state of the system being specified. This state definition will include all the information held by the system alongside any *invariants* or properties that hold for the lifetime of the system. Once the state has been defined we define the initial state of the system which assigns the appropriate initial values that the system will have on startup. With this in place all of the operations that can occur are individually defined. Each operation is defined in terms of its effect on the state. These effects are expressed by showing the relationship between the state before the operation and the state after the operation has completed. A good high level specification does not say how this transformation occurs, only what the end result will be. How precisely we convert the before state to the after state is an implementation issue and there may be many different ways of effecting this transformation. It is considered bad practice to enforce one particular implementation approach.

We now show how the specification of Sugarscape proceeded before discussing how effective this process was. The complete specification is not shown here due to its length but is available in full [26].

Types and Constants Constants are simply defined by naming the constant and defining its type. Then we state what its specific value is and any invariant properties that it must satisfy. These invariants may relate its value to some other known constants. In some cases we may not wish to state what its value is and only state that it is a constant. For example, the size of the lattice in Sugarscape is a constant size throughout any given simulation run but different values may be used for any one particular run. For example the simulation size may be chosen based on the amount of available processing power we have available. In this case we state that it is a constant and leave its actual value unstated in the specification. Anyone using the specification must then decide themselves what size grid they are using and state what its value is. In this case the grid size is passed in as a parameter by the simulation implementer. These constants act as placeholders that must be given values before the simulation can proceed. By identifying these constants that have not been assigned specific values we have identified (possibly deliberate) ambiguities in the definition of Sugarscape.

$$
\begin{array}{l}
CULTURECOUNT : \mathbb{N}_1 \quad (1a) \\
MINMETABOLISM, MAXMETABOLISM : \mathbb{N} \quad (2a) \\
M : \mathbb{N}_1 \quad (3) \\
\hline
CULTURECOUNT \bmod 2 = 1 \quad (1b) \\
MINMETABOLISM < MAXMETABOLISM \quad (2b)
\end{array}
$$

1. $CULTURECOUNT$ (1a) defines the size of the culture string. Although we do not have a specific size for $CULTURECOUNT$ we do know that it must be an odd number as the culture string is defined to contain an odd number of bits (1b);
2. Similarly although we know that $MINMETABOLISM$ and $MAXMETABOLISM$ (2a) values are not explicitly stated we are told that $MINMETABOLISM$ must be less than $MAXMETABOLISM$ (2b);
3. M is the dimension of the grid

When the simulation is run specific values *must* be assigned to these constants that satisfy the invariants we have given them. We leave these specific values to the implementer to assign but we have flagged what the constants that define any particular instance of Sugarscape are and forced the implementer to give them values within the given constraints.

We often need to define new types in a specification. These new types are generally used to make the specification easier to read and understand as the examples below clearly demonstrate.

$$
\begin{array}{ll}
[AGENT] & (1) \\
POSITION == 0 \mathinner{..} M-1 \times 0 \mathinner{..} M-1 & (2) \\
SEX ::= male \mid female & (3) \\
BIT ::= 0 \mid 1 & (4) \\
affiliation ::= red \mid blue & (5)
\end{array}
$$

1. $AGENT$ is used as a unique identifier for agents. We could just assign each agent a unique natural number but our approach make our intentions easier to understand and the specification easier to read;
2. $POSITION$ is also used to make specifying the 2D indices within the grid easier to read and more compact;
3. All agents have a sex attribute that can only take one out of two values;
4. BITs are used to encode both culture preferences and diseases of agents;
5. Every agent has a cultural affiliation defined as either belonging to the blue tribe or red tribe.

State In Z modularisation is achieved by dividing the specification into schemas. In this case we divided the state into two main separate parts or schemas. The first schema defined the information held by the lattice component of the simulation and the second schema defined the information held by the agents in the simulation.

Although Lattice locations can be viewed as a type of agent this division makes the state easier to comprehend and serves a useful purpose as some operations (known as *rules* in Sugarscape terminology) act only on one of these schemas (for example, the *Growback* rule/operation only affects the lattice and not the agents).

The Lattice is an $M \times M$ grid or matrix of locations where each location contains amounts of sugar[5] and pollution. Each location can hold up to a maximum amount of sugar where this maximum amount can vary from location to location.

[5] We ignore spice here for simplicity. The complete full specification also includes spice.

Lattice

$$sugar : POSITION \pfun \mathbb{N} \quad (1)$$
$$maxSugar : POSITION \pfun \mathbb{N} \quad (2)$$
$$pollution : POSITION \pfun \mathbb{N} \quad (3)$$

$$\mathrm{dom}\, sugar = \mathrm{dom}\, maxSugar = \mathrm{dom}\, pollution = POSITION \quad (4)$$
$$\forall x : POSITION \bullet sugar(x) \leq maxSugar(x) \leq MAXSUGAR \quad (5)$$

Taking each part of the schema in turn:

1. *sugar* is a mapping that stores the amount of sugar stored at each position in the lattice;
2. *maxSugar* is a mapping that records the maximum amount of sugar that can be stored in (carried by) each position;
3. *pollution* records the amount of pollution at each location;
4. Every position has a sugar level, a maximum allowed sugar level (or carrying load) and a pollution level;
5. Every position's sugar level is less than or equal to the maximum allowed amount for that position which is in turn less than or equal to the $MAXSUGAR$ constant;

We need to track the number of turns that have occurred in the simulation. Each turn consists of the application of all rules that form part of the simulation. This is specified in the simple *Step* schema below:

Step

$$step : \mathbb{N}$$

Operations Each rule in Sugarscape is defined as a separate operation in Z. To show how this proceeds we will show the specification for the *PollutionDiffusion* rule as it is one of the simpler rules. First we show the original rule definition as stated in [12]

Pollution Diffusion D_α

- Each α time periods and at each site, compute the pollution flux the average pollution level over all its von Neumann neighbouring sites;
- Each site's flux becomes its new pollution level.

This rule determines how pollution diffuses over grid. Pollution diffusion is calculated every α turns and is computed as the average pollution level of all a location's von Neumann neighbours. We rename the constant α as $POLLUTIONRATE$ for clarity as α is used in many of the rules with different meanings.

The von Neumann neighbours of a location are those immediately above, below, left and right of the current locations (aka north, south, east and west). We define the four cardinal directions taking into account the fact that the grid wraps around at its edges (i.e. it is a torus).

$$
\begin{array}{l}
north : POSITION \rightarrowtail\!\!\!\!\!\!\rightarrow POSITION \\
south : POSITION \rightarrowtail\!\!\!\!\!\!\rightarrow POSITION \\
east : POSITION \rightarrowtail\!\!\!\!\!\!\rightarrow POSITION \\
west : POSITION \rightarrowtail\!\!\!\!\!\!\rightarrow POSITION \\
\hline
\forall x, y : \mathbb{N} \bullet \\
\quad west((x, y)) = ((x - 1) \bmod M, y) \\
\quad east((x, y)) = ((x + 1) \bmod M, y) \\
\quad south((x, y)) = (x, (y - 1) \bmod M) \\
\quad north((x, y)) = (x, (y + 1) \bmod M)
\end{array}
$$

PollutionDiffusion

$$
\begin{array}{l}
\Delta Lattice \\
\Xi Step \\
\hline
maxSugar' = maxSugar \\
sugar' = sugar \\
(step \bmod POLLUTIONRATE \neq 0) \Rightarrow pollution' = pollution \\
(step \bmod POLLUTIONRATE = 0) \Rightarrow pollution' = \\
\quad \{\forall l : POSITION \bullet l \mapsto (pollution(north(l)) + pollution(south(l)) \\
\qquad + pollution(east(l)) + pollution(west(l))) \operatorname{div} 4\}
\end{array}
$$

We note here the first appearance of the notion of *after-state*. Z specifies operations in terms of how the state after the operation relates to the state before the operation. For any particular variable X in a schema, X refers to the value held by that variable before the operation starts while X' (called X prime) refers to the value it holds after the operation has finished. Thus to state that an operation increments X by one we would write $X' = X + 1$. We note that "=" is the equality predicate and not an assignment operator[6].

Ξ before a schema name indicates the inclusion of the before and the after states with the added invariant that all the after states are equal to their before states, that is the schema does not change the state values. Δ also indicates the inclusion of the before and the after states but does not state the relationship between them. If we use Δ then we are required to state in the schema how the after state values relates to the before state values. Thus in this schema we do not need to state the after state of the *step* variable - it must be equal to the before state.

The schema tells us that both *maxSugar* and *Sugar* values remain unchanged. This is stated explicitly by saying that the before and after values are equal. Pollution diffusion occurs only once every $POLLUTIONRATE$ steps. Our schema enforces this by only calculating new pollution levels when *step* (a counter telling us the number of this tick) is evenly divided by $POLLUTIONRATE$. When it is updated the new pollution level at a location is the average of its neighbour's pollution, specifically their pollution level before the operation occurs.

6.2 Results of the Specification

Formally specifying Sugarscape allowed us to identify a number of issues with its definition. We have grouped these issues into three main categories: *Lack of Clarity, Missing Information* and *Sequential biases*.

[6] So, for example, $X = X' + 1$ is a legal way of stating that X is decremented by one

Lack of Clarity The rules, although simply stated, lack clarity in their definition. Only one version of each rule is presented even when many variations are referred to in the text. The variations presented cannot always be used together, for example the *Movement* rule defined in the appendix is not the variant required if the pollution rule is also used. Our specification brings all the variants together in one place for ease of reference. We also identified the combinations of rules that are allowable within any particular simulation setup.

Missing Information Missing or incomplete information is the biggest cause for concern. In many cases we can work out the most likely answer based on context but in some cases there is not one definitive correct answer. If there was more than one arguably correct solution we choose the simplest. How we fill in these blanks can have a big effect on how the simulation proceeds. These effects may be important if we are trying to compare different implementations of Sugarscape. For example, there is no mention of any minimum amount required by agents to have children in the *Mating* rule but such an amount is referenced in the *Credit* rule.

By replacing each ambiguous interpretation with one simple and precise interpretation we allow different developers to replicate and benchmark their results against each other. All hidden assumptions that could serve to advantage one implementation over another are excised.

Sequential Biases Sugarscape is based on the assumption that it will be implemented sequentially. While this may have been a good assumption at the time it was written it is not now necessarily the case. Improvements in processing speed have recently been attained mainly through the introduction of concurrency. Simulations are now almost always run on multicore or even multiprocessor machines. Building assumptions about the processor architecture into the simulation definition is not good practice. A properly defined simulation should be independent of such implementation concerns.

The Z specification is free from all sequential assumptions. It achieves this without having to specify or constrain in any way what conflict resolution or avoidance strategies are employed. This leaves developers the freedom to try out different approaches as suits their implementation platform. However modellers still need to explicitly and clearly state what particular resolution strategies they are employing as these can alter the simulation outcomes.

Although the original Sugarscape definition explicitly assumes that the implementation will be a sequential one we have chosen not to make this assumption in our specification. To be sure this assumption can be included in the specification and we have produced an alternative explicitly sequential specification of *Combat* for comparison. We note that explicitly demanding that asynchronous updating be used does have a large affect on the structure of the specification. This is not unexpected as it is well known that asynchronous and synchronous updating give different outcomes in simulations [22,34,6,7].

6.3 Conclusions

Of the thirteen rules of Sugarscape our specification process found issues in all but four of the rules (Growback, Pollution Formation, Culture and Trade). It is interesting to note that although *Trade* is probably the most complex rule there were no issues with its definition. However, it is a self-contained rule unaffected by the other rules. The more interesting issues were due to the unforeseen interactions that can occur between rules. For example, when *Credit* and *Inheritance* are used together in a simulation it is unclear how we deal with loans of dead agents. Are they inherited by the children or discarded? Similarly *Credit* mentions using the minimum amount required by agents to have children to help calculate loan amounts but this minimum amount

is not mentioned in the *Mating* rule. The ambiguities and missing information within the other nine rules could have a significant effect on the outcome of a simulation run.

Further work remains to be done in getting agreement from the ABM community on the decisions made in producing this interpretation of Sugarscape. Any incorrect assumptions made in the course of producing this specification need to be identified, agreed upon and corrected. A more complete list of the issues identified during the specification process can be found in [26].

Sugarscape can now be used as a benchmark (or rather set of benchmarks) for modellers. This is particularly useful for those proposing alternative approaches to simulation such as synchronous updating or, for example, comparing the performance gains under different processor architectures, for example, Multicore versus Graphics Processor Units (GPU).

7 Issues with Formal Specification

We have demonstrated that formal methods can be used to specify even a complex ABM such as Sugarscape. The fact that it identified many ambiguities shows that it fulfils its purpose. Formal methods have many years of solid research and a strong theoretical grounding behind them. They are designed to make specification of large and complex systems as easy as possible by building in modularity.

Our Sugarscape specification shows that we can model an ABM at a very high level while still remaining very precise. This is a key issue as we need a way of specifying our simulations that allows not just replicability but more importantly reproducibility. The benefit is that the specification does not force us to adopt one particular set of parameters but it does force us to state precisely what values we have attached to those parameters.

There are, of course, downsides to the use of formal specification. The price we pay for their precision is a requirement for mathematical formalisms that some modellers may find too difficult or time consuming. To quote a well known paper in computer science "there is no silver bullet" [5]. The quest for a simple and intuitive natural language-based specification technique has proven futile in computer science and will similarly prove futile here. If we want reproducibility then we need the rigour of mathematics. The size of the formal specification produced for Sugarscape may seem large but we must remember that (i) it is an entire family of simulations with very complex interactions and (ii) we have specified it twice, once with a single resource (Sugar) and once with two resources (Sugar and Spice). Most other ABSS are smaller and simpler in scope and would require less work.

While the Z notation allowed us to produce an acceptable specification of Sugarscape there are other more recently developed specification languages such as Alloy [24] or Object-Z [35] that may prove even more effective and produce simpler specifications. Translating from Z to these other notations would allow for a fair comparison to be made between the different specification techniques. Formal specification deserves serious consideration as an approach to solving the replicability issue in ABM. While some may feel it is "overkill" for normal specification there is a stronger case for its use in specifications used as benchmarks, comparisons of updating techniques (synchronous vs asynchronous) or implementation techniques on GPU/multicore architectures.

Formal methods are difficult and time consuming to read and/or write. Both tasks require a familiarity of the underlying mathematics and the available tools. While reading and understanding an existing specification is not as onerous as writing one some math sophistication is still required. It may be the case that computer scientists will be needed to produce these specifications for modellers and also to aid in implementing them until modellers become more familiar with the mathematics underlying formal specification techniques. This should come as no surprise: software development is a highly skilled task. Formal methods are not an easy solution to the problem but then decades of research in computer science indicate that there is no

easy solution (no "silver bullet") to this problem. Formal methods do not guarantee correctness but they do reduce the ambiguity of the ABM description and make errors easier to spot and verify. Any attempted solution to this problem will require hard work and a firm mathematical foundation.

8 Conclusion

While it is recognised that there is an issue with replication in ABM, it has been pointed out [30,9] that replication is not the same as reproduction. Simply forcing authors to publish their model's implementation will not give us reproducibility. If their models are incorrect then just replicating them will just replicate their errors. What is required is more along the lines of the model-driven approach [11]. The inclusion of extra data alongside the published results will obviously help but it is not enough. We should be able to reproduce their results, based on their model, using our own implementations.

What is missing is the ability to specify the model in a clear and precise manner such that independent replication of an implementation becomes possible and ambiguities and errors in the model can be revealed. One approach in computer science that fits this criteria is formal methods where the model is specified with mathematical rigour and can, if necessary, be formally verified.

Although [11] states that experimentation is the only way to verify a model this does not preclude but rather requires some precise specification of the model so we can prove that an implementation of the model is a correct interpretation of this model.

We have produced a formal specification of the Sugarscape family of simulations [26]. It is, to the best of our knowledge, the first formal specification of the entire Sugarscape simulation family. The purpose of the specification is to provide a clear, unambiguous and precise definition of Sugarscape and demonstrate the applicability of formal methods to helping solve the replication problem. The specification has identified many ambiguities and/or missing information in the original rule definitions. Where there was an obvious way of removing these ambiguities we have done so. If there was more than one possible solution we chose the most likely one.

Because our specification is high level and only defines the before and after state of each rule it makes no assumptions as to how any rule will be implemented. All inherent biases towards, for example, a sequential implementation have been removed. Implementers have complete freedom as to what programming model they employ (Object-oriented, imperative, functional, or any concurrent approach). Any simulations that adhere to the specification can be properly compared in terms of performance or patterns of behaviour. This will put on a firmer foundation any claims made by researchers about their implementations.

Specifying Sugarscape has made a huge difference to our understanding of the simulation and although it took some time to write we feel that it has, in the end, saved time by identifying issues in advance of implementation and giving the simulation a clarity that enables it to be used by others. While we could have implemented our synchronous Sugarscape framework without first formally specifying it the specification process ensured that we had a fuller understanding of the simulation. Any problems or ambiguities were found and clarified before we started coding thus making the code cleaner and simpler.

While we believe that the Z notation is not the best formal method to use, it is well known and many newer methods more suited to ABM are based on it [24,35]. Our Z specification can be translated into any one of the newer methods by their advocates without much difficulty.

The specification itself is complex. Specifying the single resource scenario took 46 pages of Z with embedded explanatory text. Researchers not familiar with Z or its underlying mathematics may find it difficult to understand. It is an open question as to whether the benefits of formally specifying an ABM is sufficient to repay the amount of effort required to produce it.

Specifying systems precisely is difficult. Computer science has been engaged in this issue for decades and is still trying to solve the problem. There is no easy solution. In terms of precision formal specifications are unbeatable as they are embedded in a firm mathematical foundation. Formally specifying an ABM is no small undertaking and requires a familiarity with mathematical topics that some modellers may not presently be familiar with.

If modellers want reproducibility then they must treat their ABM specifications as first class citizens and demand the same precision from them as they do from other aspects of their models. We have shown that formal methods, even *formal methods-lite* can be used to uncover problems with ABMs.

8.1 Further Work

This specification has been made available [26] for anyone who wishes to use it and provides a standard reference that researchers can use when producing their own implementation.

It will prove invaluable, for example, to researchers who advocate the use of a GPU [8,29,32], containing hundreds to thousands of individual processors, for improved performance. The more complex rules in Sugarscape (such as Combat, Inheritance and Mating) are not easily parallelized and provide a better test of how well parallelisation algorithms work than other simpler rules. By providing a precise and full set of these rules it is now possible for researchers to properly compare how different models cope with the more complex and realistic ABMs.

We are currently using the specification to provide a reference implementation of Sugarscape [27] that employs synchronous updating. This implementation is being used to benchmark the concurrent performance of novel synchronous updating algorithms for ABM on multicore and GPU architectures.

References

1. Ali E. Abdallah, Cliff B. Jones, and Jeff W. Sanders, editors. *Communicating Sequential Processes: the First 25 years*, volume 3525 of *LNCS*. Springer, 2005.
2. Matthew Berryman. Review of software platforms for agent based models. Technical report, DTIC Document, 2008.
3. Anthony Bigbee, Claudio Cioffi-Revilla, and Sean Luke. *Replication of Sugarscape using MASON*. Springer, 2007.
4. Dines Bjørner and Klaus Havelund. 40 years of formal methods - some obstacles and some possibilities? In *FM 2014: Formal Methods - 19th International Symposium, Singapore, May 12-16, 2014. Proceedings*, pages 42–61, 2014.
5. Frederick P. Brooks, Jr. No silver bullet essence and accidents of software engineering. *Computer*, 20(4):10–19, April 1987.
6. Geoffrey Caron-Lormier, Roger W. Humphry, David A. Bohan, Cathy Hawes, and Pernille Thorbek. Asynchronous and synchronous updating in individual-based models. *Ecological Modelling*, 212(3 4):522 – 527, 2008.
7. David Cornforth, David G. Green, and David Newth. Ordered asynchronous processes in multi-agent systems. *Physica D: Nonlinear Phenomena*, 204(1 2):70 – 82, 2005.
8. Christophe Deissenberg, Sander van der Hoog, and Herbert Dawid. EURACE: A massively parallel agent-based model of the European economy. *Applied Mathematics and Computation*, 204(2):541 – 552, 2008. Special Issue on New Approaches in Dynamic Optimization to Assessment of Economic and Environmental Systems.
9. Chris Drummond. Replicability is not reproducibility: Nor is it good science, 2009.
10. R. M. D'Souza, M. Lysenko, and K. Rahmani. SugarScape on steroids: simulating over a million agents at interactive rates. *Proceedings of Agent2007*, 2007.

11. Bruce Edmonds and David Hales. Replication, replication and replication: Some hard lessons from model alignment. *Journal of Artificial Societies and Social Simulation*, 6(4), 2003.
12. Joshua M. Epstein and Robert Axtell. *Growing Artificial Societies: Social Science from the Bottom Up*. The Brookings Institution, Washington, DC, USA, 1996.
13. Andrew Martin et al. CZT: Community z tools. http://czt.sourceforge.net/. Accessed: 2015-09-30.
14. Jose Manuel Galan, Luis R. Izquierdo, Segismundo S. Izquierdo, Jose Ignacio Santos, Ricardo Del Olmo, Adolfo Lopez-Paredes, and Bruce Edmonds. Errors and artefacts in agent-based modelling, 2009.
15. Nigel Gilbert. private communication, March 2014.
16. Volker Grimm, Uta Berger, Finn Bastiansen, Sigrunn Eliassen, Vincent Ginot, Jarl Giske, John Goss-Custard, Tamara Grand, Simone K. Heinz, Geir Huse, Andreas Huth, Jane U. Jepsen, Christian J¿rgensen, Wolf M. Mooij, Birgit M ller, Guy Pe er, Cyril Piou, Steven F. Railsback, Andrew M. Robbins, Martha M. Robbins, Eva Rossmanith, Nadja R ger, Espen Strand, Sami Souissi, Richard A. Stillman, Rune Vab¿, Ute Visser, and Donald L. DeAngelis. A standard protocol for describing individual-based and agent-based models. *Ecological Modelling*, 198(1 2):115 – 126, 2006.
17. Volker Grimm, Uta Berger, Donald L. DeAngelis, J. Gary Polhill, Jarl Giske, and Steven F. Railsback. The {ODD} protocol: A review and first update. *Ecological Modelling*, 221(23):2760 – 2768, 2010.
18. Franziska Hinkelmann, David Murrugarra, AbdulSalam Jarrah, and Reinhard Laubenbacher. A mathematical framework for agent based models of complex biological networks. *Bulletin of Mathematical Biology*, 73(7):1583–1602, 2011.
19. Konrad Hinsen. A data and code model for reproducible research and executable papers. *Procedia Computer Science*, 4(0):579 – 588, 2011. Proceedings of the International Conference on Computational Science, {ICCS} 2011.
20. Konrad Hinsen. Computational science: shifting the focus from tools to models, 2014.
21. Konrad Hinsen. Activepapers: a platform for publishing and archiving computer-aided research. *F1000Research*, 3, 2015.
22. Bernardo A Huberman and Natalie S Glance. Evolutionary games and computer simulations. *Proceedings of the National Academy of Sciences*, 90(16):7716–7718, 1993.
23. Mario E. Inchiosa and Miles T. Parker. Overcoming design and development challenges in agent-based modeling using ascape. *Proceedings of the National Academy of Sciences*, 99(suppl 3):7304–7308, 2002.
24. Daniel Jackson. *Software Abstractions: Logic, Language, and Analysis*. The MIT Press, 2006.
25. C. B. Jones. *Systematic Software Development using VDM*. Prentice Hall International, second edition, 1990.
26. Joseph Kehoe. The specification of sugarscape. http://arxiv.org/abs/1505.06012, 2015.
27. Joseph Kehoe. Synchronous sugarscape: A reference implementation. https://github.com/josephkehoe/Sugarscape, 2015. Accessed: 2015-12-30.
28. John R Kitchin. Examples of effective data sharing in scientific publishing. *ACS Catalysis*, 2015.
29. Mikola Lysenko and Roshan D'Souza. A framework for megascale agent based model simulations on graphics processing units. *J. Artificial Societies and Social Simulation*, 11(4), 2008.
30. Roger D Peng. Reproducible research in computational science. *Science (New York, Ny)*, 334(6060):1226–1227, 2011.
31. Steven F. Railsback, Steven L. Lytinen, and Stephen K. Jackson. Agent-based simulation platforms: Review and development recommendations. *Simulation*, 82(9):609–623, September 2006.
32. Paul Richmond, Simon Coakley, and Daniela M. Romano. A high performance agent based modelling framework on graphics card hardware with CUDA. In *Proceedings of The 8th International Conference on Autonomous Agents and Multiagent Systems - Volume 2*, AAMAS '09, pages 1125–1126, Richland, SC, 2009. International Foundation for Autonomous Agents and Multiagent Systems.
33. Candelaria Sansores and Juan Pav n. Agent-based simulation replication: A model driven architecture approach. In Alexander F. Gelbukh, Alvaro de Albornoz, and Hugo Terashima-Mar n, editors, *MICAI*, volume 3789 of *Lecture Notes in Computer Science*, pages 244–253. Springer, 2005.
34. Birgitt Schönfisch and André de Roos. Synchronous and asynchronous updating in cellular automata. *Biosystems*, 51(3):123 – 143, 1999.

35. Graeme Smith. *The Object-Z specification language*, volume 1. Springer Science & Business Media, 2012.
36. J. M. Spivey. *The Z Notation: A Reference Manual*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1989.
37. Jim Woodcock, Peter Gorm Larsen, Juan Bicarregui, and John Fitzgerald. Formal methods: Practice and experience. *ACM Comput. Surv.*, 41(4):19:1–19:36, October 2009.