Utrecht University

Master Thesis

Repeatability in simulation of large-scale agent-based social behavior

Author:
Kelvin J. M. Lubbertsen
5766613

Supervisor:
Dr. M. M. Dastani

Second examiner:
Dr. F. P. M. Dignum

Department of Information and Computing Sciences
Faculty of Science
Artificial Intelligence

August 16, 2017

Contents

List of Figures

Abstract

In large-scale agent-based social simulations situations are tested and compared based on measurements. These measurements measure the behavior of the agents in the simulation. Due to the concurrency within the simulation model of large-scale agent-based social simulations the ordering of some actions cannot be ensured which can lead to noise within the simulation model.

Noise within the simulation model means that the simulation is less accurate due to the fact that it no longer models the real world system it tries to represent. Given the same input one would expect a similar output, but this is not always the case in large-scale agent-based social simulations. Ensuring this means ensuring repeatability.

We propose synchronization techniques in multi-agent systems for ensuring repeatability in large-scale agent-based social simulations and show the correctness of these techniques in our attempt to ensure repeatability in large-scale agent-based social simulations.

Chapter 1

Introduction

In this chapter we will introduce the topic of ensuring repeatability in large-scale agent-based social simulations. For doing so we will first clarify the motivation for this topic, why it is relevant and we want to investigate it. Then we will give an outline of our research by defining our research question and we finish of with an outline of the rest of this thesis.

1.1 Motivation

For many years simulation has been one of the most applied scientific disciplines (Lane et al., 1993) and in the last decades needs have come along to model as agents with complex cognitive states (e.g. agent-based simulations in Law and Kelton (2014)) to describe more complex behavior. With the emergence of more complex demands from the industry, like simulating accurate human behavior instead of just procedural tasks like a production line, we see an important application for multi-agent systems. These systems are capable of modeling more complex situations by using BDI-agents (in Dastani (2008)) and communication like FIPA-ACL by FIPA (1999).

In recent years applications have been developed that require many agents, think for instance about modeling roads with many vehicles as agents. To allow such growth in processing power, multiple agents are simulated in parallel. This requires repeatability, a notion which says that given a similar input the output should be similar under certain conditions. This is needed for easy debugging for the programmer of the simulation but also because simulations tend to model real world systems and any behavior that is due to repeatability problems decreases the accuracy of the simulation. Therefore it is important to ensure repeatability in such simulations.

1.2 Research question

Having motivated why this is important, we can easily define the research question: how can repeatability in large-scale agent-based social simulations be ensured?

From this we identify three key parts of the problem: ensuring repeatability in the agents itself, ensuring repeatability in the multi-agent system and building such a large-scale agent-based social simulation which ensures repeatability.

Firstly it is important to look at repeatability from an agent perspective. Since agents are such an important part of the simulation it is important to identify what are common problems that make ensuring repeatability difficult and how to solve these problems.

But agents do not behave on their own in large-scale agent-based social simulations, they interact with the environment and each other. They together with other agents form a multi-agent system, so our second part about ensuring repeatability is about ensuring repeatability in large-scale multi-agent systems where interacting agents run concurrently.

Last but not least, one should be able to build such repeatable large-scale agent-based social simulations. This is what we therefore look at as well.

So combining all these parts, we have divided the main question into three sub-questions, namely:

- How can we ensure repeatability in complex reasoning agents such as BDI-agents?
- How to ensure repeatability in large-scale multi-agent systems where interactive agents run concurrently?
- How to build large-scale agent-based social simulations that ensure repeatability?

All these three questions together answer the main research question: how can repeatability in large-scale agent-based social simulations be ensured?

1.3 Outline

In this thesis we will try to answer the research question based on the sub-questions defined above. But to do this, first in chapter 2 we will describe the background material needed for this thesis, such as what an agent is and what a simulation is. This chapter finishes with an overview of the notion of repeatability and the related notion of reproducibility from the literature.

The notion of repeatability from the literature is then used to define our own notion of repeatability in chapter 3. In this chapter we also reason about different variations in the form of different equivalence relations.

Having a clear idea about our notion of repeatability we try to ensure it in complex reasoning agents and large-scale multi-agent systems by the means

of different synchronization techniques for different issues with repeatability. This is done in chapter 4. Here we try to answer both the first and second sub-question from section 1.2.

Finally, in chapter 5, we implement based on a few examples the synchronization techniques from chapter 4. We do this to show how to build large-scale agent-based social simulations that ensure repeatability and therefore show that our synchronization techniques ensure different types of repeatability problems in large-scale multi-agent systems.

At last we will answer the research question, conclude the thesis and discuss some future work in chapter 6.

Chapter 2

Background

In this chapter background information from other sources is provided to explain the relevant notions on which this thesis is built.

The outline of this chapter is as follows. First the concepts of process algebra are described in section 2.1, then the concepts of agents and multi-agent systems in section 2.2. We continue with the implementation of cognitive agents and multi-agent systems in 2APL and OO2APL in section 2.3. And to continue with large-scale social simulations 2.4 and the simulation tooling DSOL section 2.5 to finish with the notions of repeatability and reproducibility in the literature in section 2.6.

2.1 Process algebra

For describing processes, algebras such as partially ordered sets by Dushnik and Miller (1941) and Petri nets by Petri (1962) have emerged. In this section we talk about these process algebras.

Process algebras are used for describing the flow of processes, or the behavior. Computer programs can be modeled as processes and their behavior can be analyzed using these process algebra. Therefore we describe them here.

2.1.1 Partially ordered sets

Partially ordered sets are sets that define some sort of order between its elements. This order is what turns it into a process. A partially ordered set is defined as a tuple $(X, \prec)$, where X is the set of elements and $\prec$ is the irreflexive and transitive relation of the elements of X (see Best and Fernandez (1988)).

The relation $\prec$ for two elements in X (i.e. $x, y \in X : x \prec y$) states that x happened before y. This relation $\prec$ is transitive, irreflexive and non-symmetric, creating an acyclic order. Using this relation, complex processes can be modeled which contain forks (i.e. the process splits into two processes) and joins (i.e. two processes merge into one process). This is possible because multiple elements

can run in parallel, which means that two elements (i.e. $a, b \in X$) have no ordering, meaning that neither $a \prec b$ nor $b \prec a$. From this parallelism comes the name 'partially ordered', because of the lack of ordering between two elements like a and b.

2.1.2 Petri nets

Petri nets are a different type of process algebra. Compared to partially ordered sets it has one big advantage: it can model the flow through the process explicitly. This is a special type of petri net, called a marked petri net. Overall petri nets are more specific when modeling processes. They contain separate notions of states and transitions. States, also known as S-elements, they represent the elements we have seen already in partially ordered sets. The transitions, also known as T-elements,define the relation between states and add some transition logic that can be used to describe the flow.

Formally a petri net N is a triple $N = (S, T; F)$ with the requirements that $S \cup T \neq \emptyset$ and $S \cap T = \emptyset$. In this S is the set of S-elements, T the set of T-elements and F the flow relation between the transitions and the states. The set of states and transitions is called $X = S \cup T$. Formally speaking F is (by Nielsen and Thiagarajan (1984)):

$$F \subseteq (S \times T) \cup (T \times S) \text{ such that } dom(F) \cup range(F) = S \cup T \tag{2.1}$$

The above requirements about F state that a flow relation is a relation from a S-element to a T-element or from a T-element to a S-element. Furthermore it states that the flow relations of F should cover all S-elements and T-elements.

Furthermore, the notions of pre-sets and post-sets exist. A pre-set of an element $x \in X$, denoted ${}^*x$ denotes all the elements that come directly before this set, whereas the the post-set (i.e. x^*) denotes the elements that come directly after it.

To describe the flow of processes, markings exist. Markings are sets of tokens that are placed on states. A marking fires whenever every state in the pre-set of a transition contains at least one token. These tokens are then placed on the states in the post-set of that same transition. For this, the M, which denotes the marking is added to the tuple of a petri net. A petri net which contains a marking is called a marked petri net.

Two important characteristics of petri nets exist: the notion of live marked petri nets and that of safe marked petri nets. A live marked petri nets is a petri net where there will always be a transition that can be fired, meaning that there is always a next marking, it never stops. A marked petri net is safe if by the structure of the petri net it is impossible that at one time, more than one token exists on a single state.

One of the main reasons why petri nets have become so popular is their ability to easily write them graphically. For demonstration purposes of this, suppose a marked net Σ, described in Figure 2.1. In this figure, the S-elements

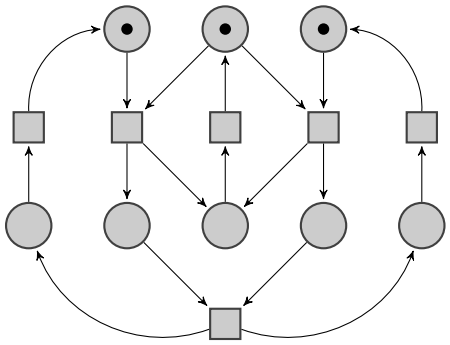


Figure 2.1: Example from Nielsen and Thiagarajan (1984)[Figure 1], describing a marked petri net graphically

Figure 2.2: Example of a standalone action a in BPA

are circles and the T-elements rectangles. This is a common convention for petri nets according to Nielsen and Thiagarajan (1984). This petri net shows a marking of 1 for the top three states and is a petri net that is safe and live.

2.1.3 Basic process algebra

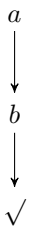
Another type of process algebra is called basic process algebra (BPA) and defined by Fokkink (2007). Advantage of this process algebra is the existence of logical axioms (i.e. transition rules) and proof structure. In BPA a process consists of a nonempty finite set of actions named A. Every action $a \in A$ is a standalone action that can be executed alone. A process terminates by a transition to a final state denoted as $\surd$ (see Figure 2.2).

As can be seen from Figure 2.2, these processes can easily be represented graphically. Written formally, the example of Figure 2.2 is denoted as:

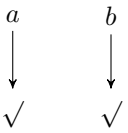
$$a \xrightarrow{a} \surd \tag{2.2}$$

Note here that a actually is 'consumed' by applying it over the transition. This may not seem like a big deal yet, however with multiple actions and the use of operators it will become clear why this is important.

Transitions and concurrency between actions is denoted using the operators $+$ and $\cdot$ (see Figure 2.3). By conventions the $\cdot$ operator is left out when it is clear from the context, meaning that $a \cdot b \cdot c$ can be written as abc. This makes it easier to read because it requires less brackets.



(a) Example for $a \cdot b$



(b) Example for $a + b$

Figure 2.3: Examples showing use of operators + and · for transitions in BPA

An example of a process is denoted in BPA is:

$$a \cdot (a + b) \cdot b \tag{2.3}$$

Note from this example that actions can occur more than once (i.e. they are not unique) and that (by convention) the termination ($\surd$) is left out when it is clear from the context and serves no further use.

The main advantage of BPA is the existence of a sound and complete set of axioms and transition rules that makes it easy to proof things about processes. Examples are for instance:

$$\frac{}{x \xrightarrow{x} \surd} \tag{2.4}$$

Or:

$$\frac{x \xrightarrow{x} \surd}{x \cdot y \xrightarrow{x} y} \tag{2.5}$$

2.2 Agents

In short two types of agents exist, as explained in Shoham (1993) and Kabanza et al. (1997). Before describing these types, it is first important to describe what both types have in common: being an agent. This definition is used throughout science, for Artificial Intelligence the definition is given by Shoham. He defines an agent as:

> An agent is an entity whose state is viewed as consisting of mental components such as beliefs, capabilities, choices, and commitments. These components are defined in a precise fashion, and stand in rough correspondence to their common sense counterparts. (Shoham (1993) , p.52)

Shoham adds to this that agents should be 'autonomous'. About this he states:

> The sense of autonomy is not precise, but the term is taken to mean that the agents' activities do not require constant human guidance or intervention. (Shoham (1993) , p.52)

2.2.1 Proactive agents

As already mentioned in the definition of Shoham, agents consist of beliefs, capabilities, choices and commitments. This concept was already formalized by Rao and Georgeff (1991) when this concept was formed by Shoham. In their work Rao and Georgeff use modal logic to create a framework in which they describe agents to have beliefs, desires and intentions.

The framework of Rao and Georgeff uses branching time temporal logic and extends it with three modal operators, not surprisingly for the three notions beliefs, desires and intentions. Though the modal operator for desires is not called 'desire', but 'goal'.

Beliefs are the things the agent thinks are true. Though it should be mentioned that this does not necessary mean that this is the case. The agent can be wrong, but based on the information it got at the time it decided this, it thought it was the case. However, because the agent is considered consistent, it cannot be the case that if the agent beliefs ϕ, that it also beliefs $\neg\phi$.

The second notion, that of desires is defined by the modal operator 'goal'. An agent that actively tries to achieve a certain desire is said to have that as a goal. Think for instance about an example in which a cleaning robot (i.e. the agent) cleans a floor in a room. In this case according to the framework, the agent has a goal 'clean the floor of the room'.

The last notion is the most abstract, namely that of intentions. An agent is said to intend a certain state in time, if the agent has a plan (i.e. a path through time) of which it has already performed (or is performing now) one step. Getting back to the cleaning robot example, the agent can have the intention 'clean the house', which exists of cleaning the floors of multiple rooms and the windows of the house. Since it already started cleaning the floor of the first room, the agent is said to intend to clean the house.

Proactive agents are for instance implemented in programming languages like 2APL (Dastani, 2008). In Dastani (2008) an agent can have goals, which help the agent plan in advance and therefore behave proactive.

2.2.2 Reactive agents

Different from proactive agents are reactive agents. As the definition already mentions, these types of agents only respond to inputs it receives, think for instance about messages, events that occur in the world or internally in an agent. Internal events seem at first a bit weird, since that would mean that agents have some sort of proactiveness, however this is primarily a case of defining the scope of the trigger. In many agent-oriented programming languages or frameworks like Dastani and Testerink (2014) there exist different types of triggers, namely

internal and external ones. External triggers are for instance triggers from the environment of the agent.

For internal triggers it is key to have a clear example in mind. On such example can be the case in which an agent receives an external trigger from the environment and then sets a timer. When this timer finishes, this is an internal trigger, due to the scope of the timer.

Messages are a third type of trigger of an agent in many frameworks and languages like in Dastani and Testerink (2014); Dastani (2008); Bellifemine et al. (2007). These are commonly separated from external triggers (i.e. from the environment) to make implementing agent communication languages like FIPA (1999); Finin et al. (1994) easier.

2.2.3 Rationality

Agents in most implementations (like Dastani and Testerink (2014); Dastani (2008); Bellifemine et al. (2007)) are defined to be rational. According to Rao and Wooldridge (1999), rationality in agents is defined as:

> Rational agents are software entities that perceive their physical or software environment through appropriate sensors; have a model and can reason about the environment that they inhabit; and base their own mental state take actions that change their environment. (Rao and Wooldridge (1999) , p.1)

Furthermore Rao and Wooldridge (1999)[p.1] describe rational agents by three key aspects:

- Balancing reactive and proactive behavior
- Balancing perception, deliberation and action
- Balancing self-interest and community interest

Reactive and proactive behavior is already described in sections 2.2.1 and 2.2.2, finding a balance in this is important. Think about a case in which an agent performs a certain task to achieve a long term goal (i.e. proactive behavior) in which it suddenly realizes it is in danger and should flee, then it is desired to balance the agent in such a way that it prioritizes this reactive behavior above the proactive behavior. The other way around, when an agent is performing a task for a long term goal, if it gets distracted all the time it is of no use either. This is meant by the correct balance of behavior, which rational agents do.

The balance in perception, deliberation and action is all about balancing the input and which ones to take most serious. For instance Van der Hoek et al. (1999) embeds different types of beliefs (i.e. observation, communication and default) in their framework. These types of beliefs are prioritized, this framework prioritizes observations above communication and communication above default beliefs.

The third type of balance is that of balancing their interests. Think for instance about balancing between the interest of the group and that of an agent itself. Such a dilemma is the multi-agent social dilemma (Stimpson and Goodrich, 2003). The analysis in this dilemma is that it is the dominant strategy to invest in the self-interest, however the community interest eventually leads to a bigger reward if all agents pursue it. The experiments by Stimpson and Goodrich (2003) show that rational agents can see the interest of the community interest and perform accordingly but also that one can tweak the balance such that agents are only interested in their self-interested goal. This example shows that rational agents can balance their interests.

2.2.4 Multi-agent systems

Just like humans, agents often do not live alone. In the sense of agents, we then talk about multi-agent systems. According to Wooldridge (2009) a multi-agent system consists of a few things, namely: (1) multiple agents, (2) communication and (3) an environment. For instance in Dastani (2008) the possible existence of multiple environments is explicitly mentioned, which makes it possible to have more complex environments.

Furthermore, based on the notion of an agent Dastani states that agents can communicate. It namely includes a set of messages in the agent configuration.

2.3 2APL and OO2APL

Now let us focus on practical implementations of agents. The focus here will lay on the programming language 2APL introduced in Dastani (2008) and more specifically the recently added framework from Dastani and Testerink (2014) called OO2APL. Reason for this is that 2APL has a unique property over most other programming languages: its modularity introduced by Dastani and Testerink (2014) makes it easy to read and adapt the internal workings of 2APL. Also it allows different planning, repairing and executing constructions for beliefs, goals and events (i.e. internal, external or messages) making it very suitable for concurrent execution.

2.3.1 Plans and triggers

The notion of plans was already discussed briefly, however this notion is key to understanding the concepts of most agent-oriented programming languages and frameworks. Sometimes, like in Pollock (1999), the notion of plans is coupled to the notion of 'intention' from Shoham (1993). In agent-oriented programming a plan is a procedure that is executed to enforce (partially) a certain next state. Think for instance about an cleaning robot. To ensure the goal 'clean the floor of the room', the robot can have multiple plans, like:

1. Clean the dirt at a location

2. Go to the dirt
3. Scan surroundings (to check if the entire floor is clean or to identify the next bit of dirt)

The above three plans are all three components of the procedure of the goal 'clean the floor of the room'. To ensure the correct plan is executed, these plans have conditions that check which plan should be executed. These conditions are built based on the beliefs. Think for instance about conditions whether the room still contains dirt or whether there is dirt at a certain location.

Such conditions are conditions over the beliefs, which is important to keep in mind. It can for instance be the case that the agent believes that there is no dirt in the room (i.e. in the case of the plan 'scan surroundings'), but that he just has not noticed yet that there is some dirt at a location he has not scanned yet. By scanning the floor he then identifies new places of dirt, which he will then clean.

As can be noticed from the example above, plans are triggered when a certain condition holds. The combination of the condition, information about the trigger and the plan itself is called a plan scheme in OO2APL. In total, four types of plan schemes exist in OO2APL (Dastani and Testerink, 2014), plan schemes that handle:

- Internal triggers (e.g. triggers sent by other plans from the same agent)
- External triggers (e.g. triggers sent by the environment)
- Messages
- Goals

From OO2APL it can be noted that every trigger is analyzed and based on that a plan is selected that should be executed. Therefore every agent consists of four sets of triggers that are analyzed by the corresponding set of plan schemes from which plans are selected.

The plan schemes define the agent at the moment the agent is created. During the life of an agent its behavior can be changed using the notion of trigger interceptors. Interceptors are plan schemes that can be added to the agent's plan schemes at runtime, they have the characteristic to be destroyed when triggered by default, however this can be changed in the definition of the interceptor.

For initialization the agent can define a special type of plan, called an initial plan. This type of plan can for instance be used to adopt goals or send initial messages. This type of plan in OO2APL corresponds to the initial goal base of Dastani (2008)'s 2APL.

Interceptors are prioritized by default above all other plan schemes of the same type. Meaning that for instance all goals are first checked against all interceptor goal plan schemes before all other goal plan schemes are tried. This

is done because it makes sure that one can handle these earlier than the default behavior and when necessary block the default behavior.

Whenever a plan is triggered at some time it can in OO2APL continue to exist for a long period of time. These plans are called persistent plans. This is especially useful for interceptor plan schemes where the plan scheme is removed after it successfully consumes a trigger, this way the plan can continue in the future. Note however that whenever a plan is not finished by default and the trigger remains to exist, multiple instances of the same plan exist which can lead to undesired behavior.

2.3.2 Deliberation cycle

One run in which all different types of triggers (i.e. internal triggers, external triggers, messages and goals) are tested against all plan schemes and all plans are executed is called a deliberation cycle in 2APL. A deliberation cycle consists of the following steps according to Dastani (2008) and updated by Dastani and Testerink (2014):

1. Apply all goals to all goal plan schemes
2. Apply all external triggers to all external trigger plan schemes
3. Apply all internal triggers to all internal trigger plan schemes
4. Apply all received messages to all message plan schemes
5. Execute all plans

In the above list, the interceptors are phases in the applications of the plan schemes for the corresponding type. The agent designer gets, after creating an agent on the platform, an interface from which it can send external triggers to the agent it has just created. Internal triggers can, just like messages and goals, be created in every plan.

All goals, triggers (i.e. internal and external), messages and plans are stored in a separate object that is known to the platform that runs the agents. This object consists of:

- A set of goals
- A set of internal triggers
- A set of external triggers
- A set of messages
- A set of plans
- A message client (for sending messages to other agents)
- The belief base (see section 2.3.4)

- A set of interceptors for goals
- A set of interceptors for internal triggers
- A set of interceptors for external triggers
- A set of interceptors for messages
- An unique identifier that identifies the agent
- A set of death listeners

Most items in the list above are self-explaining, like the sets of goals, triggers and interceptors for instance. However some are not, like the message client for instance. This is the system in the platform that serves as the router making sure that the messages arrive at the correct agent. By default, the agent can only send messages to one agent at a time, but since the message client can easily be changed, one can think of applications where groups of agents receive the same message. For this purpose the message client gets notified when a new agent is created, such that it can register this agent.

The death listeners are the second item in the list that requires some additional description. When agents get removed from the platform (i.e. they die), some things need to be notified. Think for instance about the message client, you cannot sent messages to agents that do not exist anymore. Or the user interface that might want to render the agent on the screen, which is not needed anymore, since the agent does not exist anymore.

Based on this deliberation cycle one can clearly see that this is a cycle of a pro-active agent, since it has goals. Whenever the set of goals is empty, it is called a reactive agent.

2.3.3 Goals

Goals are in Dastani and Testerink (2014)'s framework a special type of trigger which add another condition: the condition of whether the goal is achieved or not. The condition reasons about the agent's belief base (see section 2.3.4). Achieved goals are removed from the set of active goal triggers.

When a new goal is added, this is called 'adopting' and can be done in plans in Dastani and Testerink (2014)'s framework. But only if the goal is not already adopted. When a goal is removed, which is called 'dropping', two cases are possible:

1. The goal is achieved
2. The goal is dropped intentionally by a plan

The check of whether a goal is achieved is embedded into the abstract goal class in OO2APL and checked every deliberation cycle. Plans can drop a goal intentionally when this is desired by the agent designer.

For preventing that goals that cannot do anything at a deliberation cycle, an agent can suspend goals. Think for instance about an agent that has to analyze a certain property of the environment without the use of an external trigger, then one can suspend a goal until it is possible for that agent to analyze the property.

Furthermore, when no active goals or triggers exist (including internal and external triggers and messages), the whole agent is set to a sleep-mode and reactivated when needed. One can suspend goals up until a certain type of trigger occurs or when a certain condition is met.

Instead of creating a plan every deliberation cycle and executing it accordingly for a goal that does practically nothing (i.e. no action follows from the plan), this concept improves on this by checking only if a condition holds. These conditions are:

- One of the trigger sets (i.e. goals, internal triggers, external triggers, messages) consists of a trigger of type X
- A certain condition on the trigger sets holds

2.3.4 Belief bases

Up until now, the focus has been on goals and plans and how they are designed and implemented in Dastani and Testerink (2014). However there is another important notion: that of beliefs. A quick recall: beliefs are the things the agent thinks are true (but not necessary are) about the environment, itself and other agents.

Beliefs in Dastani and Testerink (2014)'s framework are called 'contexts'. They are added when the agent is created, which does not mean that at that time all the beliefs are known (they will inevitably change), but more on this later. As an agent can contain multiple contexts, they together are called the agent's belief base as in Dastani (2008). Plans and goals can access the agent's belief base at every time and reason about it.

In Dastani and Testerink (2014), object-oriented programming is combined with 2APL, which adds an additional layer of software engineering to the 2APL definition. Key in this are the notions of classes and objects from object-oriented programming paradigm. Classes are definitions of entities which contain properties and objects are instantiations of these classes that 'live' at runtime. Every agent's belief base can only contain one object of every class that it can request by giving the type of class.

Note here that this means classes of belief bases, one belief base class can for instance be the belief base of a type of agent. This belief base class can off course contain lists which contain multiple objects of the same instance.

It can also happen that two agents share the same object of the same class in their belief base. Examples of cases where this is the case is for instance the 'blockworld' or 'itemdatabase' environments used in Dastani (2008).

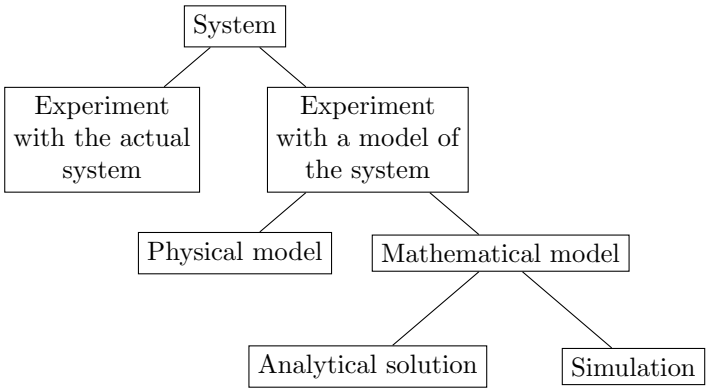


Figure 2.4: Ways to study a system (from Law and Kelton (2014)[p.4]

2.4 Large-scale agent-based social simulations

The subject under investigation in this thesis is a so called large-scale agent-based social simulation. Based on the literature one can analyze what this notion means, which is exactly the purpose of this section.

2.4.1 Simulation

Let us first look at what a simulation is. To do this, let us use Figure 2.4. It can be derived from this figure that we first need a system. Examples of such a system might be a forest fire (Filippi et al., 2010), a road construction situation (Lu, 2003) or hospital management (Glowacka et al., 2009). One can experiment with these systems in real world, however one can imagine that it is not always a good idea. For instance it might be too expensive, unsafe or the real world system does not exist yet.

When for some reason testing with a real system is impossible or impractical, this system is represented as a model. A model is a representation of the real system, however only with the components that have a correlation with the aim of the simulation. Think for instance about a simulation of a service desk. Here it might not be needed to model the actual locations of customers waiting in the queue, only their ordering of the customers might be important.

Some elements do not have to be modeled in the same way as they exist in the real world, as long as their behavior is similar enough such that the model still represents the real world system. Examples of this might include modeling a tram line, where you do not explicitly model the trams itself (i.e. where on the track it is), but only their arrival at and departure from different stops.

As already briefly mentioned, simulations have a certain aim. They try to measure a certain situation based on a set of criteria. These measurements are numeric or boolean measurements that quantify this goal defined by the designer of the situation. These measurements are therefore a set of formula. However,

there is a difference between an analytical solution of a mathematical model and a simulation. An analytical solution can, through mathematics, find the best possible value in certain scenario. But in many cases this is not possible, because the problem is for instance NP-hard (van Leeuwen, 1990; Kirkpatrick et al., 1983; Law and Kelton, 2014).

Building process

For better understanding simulations and their equivalence properties, let us look at the building process according to Law and Kelton (2014). The designer of the simulation has a few tasks. These tasks are:

- Defining the goal of the simulation
- Defining the scenarios for the simulation
- Defining the measurements for the simulation

The above tasks are required for defining the scope of the simulation model. Based on this, the model is built. This model can use different simulation techniques such as discrete event simulation (Varga, 2001) or continuous simulation (Mitchell and Gauthier, 1976).

Discrete event simulation creates a model where every change of the state is defined by an event. These events are stored in a priority queue where the earliest event is picked and executed. Time advance therefore is not continuous: if event x is scheduled at time t_x and event y at time t_y, then instead of doing nothing for the times between $t_x + 1$ until $t_y - 1$, it simply advances from t_x to t_y. Advantage of this approach is that it is very efficient for simulations with a sparse time line (i.e. not many events per time unit and a big time gap between two consecutive events).

Continuous simulation on the other hand has no time advance mechanism. However, this does not make it useless, because discrete event simulation often fails to produce analytically useful results for things like convergence to some equilibrium (Özgün and Barlas, 2009). Here continuous simulations are more useful. Even more, think about nonterminating scenarios in which (optionally after a warm-up phase) a certain behavior is simulated until the model decision is made to perform a measurement to describe the 'end' of the simulation (there is little agreement about how to do this correctly, see Haddock (1987)).

Now that the model is built, in the following phase, three tasks are defined (by (Law and Kelton, 2014)) to define the quality of the simulation model. These are:

- Validation
- Input analysis
- Verification

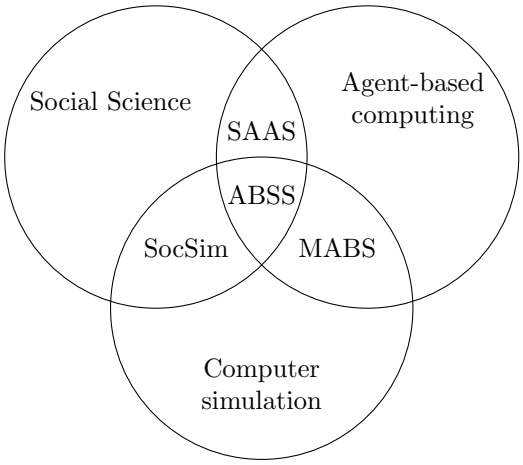


Figure 2.5: The intersections of the three areas defining ABSS by Davidsson (2002)[Figure 2]

First let us look at the validation task. This is about determining the accuracy of the model compared to the real world system, but only for the purposed goal of the simulation. It can therefore be the case that a model for one purpose is valid, but not for another purpose. The validity of the model is measured as the error between the real system and the model. This error is domain specific, but think for instance about the statistical difference of passenger arrivals generated by a tram simulation and that of a real world system.

When having built a model, one must gather the needed input for the system. Think in this case about things like serving times of a service desk, arrival times of passengers, etc. Often one is not interested in the true values recorded, but more importantly about the shape of the probability distribution from which the values are samples. Different scenarios defined in the earlier stage may define the mean of the probability distribution. Furthermore, one might have to alter the recorded data and remove outliers and other impossible cases for the purpose of this simulation. This process is called input analysis.

For verification we need to talk about the assumption document of the model. When modeling a certain situation, you cannot model everything connected to the real world system and therefore have to make assumptions. Think for instance again about the assumption of leaving out traffic lights in the model. In this very simple example, verification is checking whether there are indeed no traffic lights in the design of the new tram line. More complex verification steps might include checking assumptions by modeling them as well to see that the claim made by the assumption is correct.

2.4.2 Agent-based social simulations

Two special fields of simulation are that of social simulation and agent-based simulations. Let us analyze these subfields and their intersection (see Figure 2.5) to describe their usage and further understand their usage in ensuring repeatability. To do so, we must understand these fields of simulation one by one.

Starting with the already known field of simulations (see section 2.4.1), let us first add the social sciences, leading to social simulations (i.e. SocSim in Figure 2.5). These types of simulations analyze things like management, policy and social psychology using computer simulations.

Secondly we have the combined fields of agent-based computing and computer simulations (i.e. MABS in Figure 2.5). These lead to multi-agent based simulations (Law and Kelton, 2014). This type of simulations make use of the complex cognitive states in agents (e.g. beliefs, desires and intentions). With the increase of computational power, we see more and more of these applications (e.g. Cetin et al. (2003); Balmer et al. (2008); Tumer and Agogino (2007); Weiss (2011)). In this multiple agents live together, but the do not interact in a social way (i.e. communicate or coordinate) according to Law and Kelton (2014). In Law and Kelton (2014) this subfield is said to be a special type of discrete event simulation.

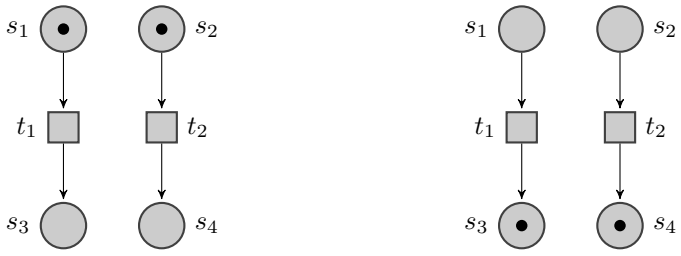
The third possible combination (see Figure 2.5) is that of the combination of social science and agent-based computing. This subfield is mentioned in Figure 2.5 as 'SAAS', which stands for Social Aspects of Agent Systems. This is according to Davidsson (2002) about studying the social notions of norms, organizations and competition for instance.

Combining all the fields of Davidsson (2002) in Figure 2.5, agent-based social simulations (i.e. ABSS in Figure 2.5) are simulations in which complex cognitive agents (e.g. BDI-agents) can interact with each other (e.g. communication and coordination) and their environment.

The difference between agent-based social simulations and the subfields can best be seen from the point of an agent. In social simulations it is social (i.e. can interact with other agents) but it has no complex cognitive state, whereas it has a cognitive state in agent-based simulations. In the form of social aspects of agent systems, the agent does not have to be a computer program it can for instance be a human which can of course interact with other humans and has a complex cognitive state.

2.4.3 Parallel and distributed simulations

Parallel simulations or distributed simulations are simulations in which the order of execution is not strictly serial. This means that there may exist multiple threads (in the case of parallel simulation) or machines (in the case of distributed simulation) that run concurrently, meaning that they both perform a set of actions serial with minimal interference between these multiple threads of machines.



(a) State of processes with initial marking M^0

(b) State of processes after applying next marking M^1

Figure 2.6: Example of concurrency in marked petri net

An influential researcher in this field is Fujimoto, in his publications (e.g. Fujimoto (1990, 1995, 1997, 2001)) he describes the fundamentals of parallel simulation. By Fujimoto (2001, 1995) the degree of parallelism is simply the speedup gained by executing the simulation parallel.

The fields of distributed simulation and parallel simulation are strongly connected. Both these fields have some sort of concurrency in the way they execute simulations. Difference is that distributed simulations are a simulation where the concurrency is between different machines. Concurrency, is formally defined by Nielsen and Thiagarajan (1984) as two transitions t_1 and t_2 that fire, but there is no order between these two (Nielsen and Thiagarajan, 1984)[p.90].

The simplest possible example for this is the existence of two transitions that run concurrently. We describe this example for clarification in Figure 2.6 as petri nets. The two transitions t_1 and t_2 are said to run concurrently. By firing the markings the nets' markings change from Figure 2.6a to the marking shown in Figure 2.6b. These markings M^0 and M^1 are consecutive markings and therefore no ordering of first t_1 and then t_2 or the other way around exists.

Historically the U.S. Department of Defense has used distributed simulation a lot for training for instance, sometimes (like in Fujimoto (2000)) these simulations are called 'Distributed Interactive Simulation'. These systems simulated battlefields (i.e. virtual worlds) for training of soldiers (e.g. Calvin et al. (1993)) and are often considered distributed simulations. In parallel simulation the concurrency exists in a single machine through multi-threading for instance. This is different from distributed simulations where the concurrency exists between different machines (i.e. t_1 and t_2 from Figure 2.6 are on different machines).

Time management

Important in the field of parallel and distributed simulation is time management. Strictly speaking everything that has some concurrency in it has to deal with time management, so also simulations. Think for instance about two processes P and Q that run concurrently. They can send messages in the form of tuples $m = (info, time)$ consisting of information and the time that the message was

sent. Both P and Q perform some tasks and keep track of their notion of time t_P and t_Q respectively. Time management is for instance needed for the case that $t_P = 5$ and $t_Q = 3$ and Q sends a message $(info, 3)$ to P at time 3, because t_P receives the message in the past we call this a conflict.

Reason that time management is needed it that concurrently running processes can receive information from other processes that should be handled at a time that was already handled in the past. Problem with this is that process P has new information based on which it should have behaved differently at times $t_P = 4$ and $t_P = 5$ in the example above.

The question now raises, how such cases described above are handled. This is the field of time management. There are roughly two solutions: prevent these cases from happening or solve them when they happen. These two solutions form the basis for the two approaches for time management: conservative and optimistic.

One approach is the conservative approach, this approach states that one should assure such a message cannot arrive at a process before one continues. They make use of the *local causality constraint*, which states that a process will only advance in time if it can assure no message from any other process will arrive for the current moment in time. This constraint exists in both parallel simulations and distributed simulations.

The conservative approach will at all times satisfy the local causality constraint. To prevent deadlocks multiple solutions have been purposed. Null-messages inform other processes with a new timestamp of the current time the sending process is at, preventing many deadlocks. Also one looked at the dependencies between processes in the form of process topologies, graphs that represent the dependencies between processes, whether they can send messages or not to limit the amount of assurance of other processes. Chandy and Misra (1981) looked at whether processes can assure that their behavior does not change whenever any message arrives, if this is the case it can inform other processes about it.

Another approach of time management is solving conflicts when they occur. This is called optimistic time management and whenever a process identifies that a received message violates the local causality constraint it goes back in time to the moment it would not have caused a conflict, which is called a roll-back. This roll-back is implemented in the Time Warp mechanism by Jefferson (1985), which saves states of processes (or differences between states) and when a conflict occurs it resets itself to an earlier state.

This approach has a few advantages. Firstly, it allows a greater amount of parallelism, because it does not have to wait for a conflict that can possibly occur but eventually does not, since it only does conflict detection. This leads on average to a greater speedup.

Secondly, it does not rely on domain specific properties like the distance between two logical processes. This makes this type of time management more broadly applicable and does not require special knowledge from the designer that wants to use this type of time management.

2.4.4 Scale of simulations

Scale is another important notion in simulations. The scale of a simulation is more than just 'more entities' as Law (1998) is pointing out. With the notion of scalability from Fujimoto (1995) he means more parallelism. He argues that scaling the amount of machines (in distributed simulations) or the amount of cores (in parallel simulation) can have great business benefits such as the possibility to perform real time simulations.

The notion of scalability therefore seems relatively straight forward: it simply means 'more'. This implies the use of concurrency within the simulation model, meaning we are dealing with parallel simulation. Therefore the requirement for the notion of 'large-scale' in the context of simulations is the existence of concurrency within the simulation model.

2.5 DSOL

DSOL is a simulation environment developed by Jacobs (2005). It was developed as a distributed simulation platform, which also makes it suited for parallel simulation. This platform, written in Java, consists of implementations for allowing distributed and parallel simulation. For this purpose it implements core features for simulation like statistical tooling, implementations of different types of simulations and experimental tooling and extends this with easy to use components like an user interface and animations.

2.5.1 Experimental tooling

Simulation is all about testing a real world system, therefore it performs experiments. DSOL consists of an inbuilt simulator that can store different parameters for experimentation and fires events at certain times and can even change the model used for simulation.

An experiment in DSOL defines the amount of runs, the different scenarios and initial states used for simulation and the model. Experiments are executed by a simulator, which handles the experiments.

The simulator has two types of replications: for the experiment and the simulator. With replication of the experiment, computational replication is meant. This means that a certain task, in this case the experiment, is performed multiple times, for simulations this leads in several outputs of which for instance the average is used as the correct. This way extreme outliers are balanced out. The replication of the experiment means generating averages over the outcomes of one experimentation run.

DSOL knows two types of replication modes, namely terminating and steady state. The terminating state mode is used when simulations are always terminated, it continues to the next replication if the termination is reached. This is impossible when we have a continuous nonterminating simulation, so then the steady state mode is used. In such case when a steady state is reached (i.e. subsequent states do not differ or have a very small margin of difference) we

call the state 'stable' and assume the outcome has converged, then the next replication is started.

2.5.2 Statistical tooling

One of the core features of simulation is dealing with uncertainty. Therefore, different types of statistical tooling is included in DSOL. These include probability distributions and tooling for gathering statistics.

Simulation input

Probability distributions are the key part of statistical tooling regarding the input of simulations. When creating a model, one can find that certain input values change according to some probability distribution. Many commonly used probability distributions are included in DSOL, both discrete and continuous probability distributions.

Discrete probability distributions are probability distributions that can only produce a finite set of outcomes, say O and keep track of a probability over all outcomes of $o \in O$, where the probability of o is denoted as $P(o)$. Since it is a probability distribution, all probabilities add up to 1. DSOL consists of multiple discrete probability distributions like the uniform distribution, constant distribution (i.e. always the same value) and a Poisson distribution.

Note that therefore the outcome of a discrete probability distribution can never be any other value than an element in O. This is the main difference between discrete and continuous probability distributions.

For continuous probability distribution, a given outcome o, which is a sample from a continuous distribution the probability of that outcome, denoted $P(o) = 0$, however because there are an infinite amount of outcomes, these values together still add up to 1.

DSOL consists of many continuous probability distributions, including for instance the normal distribution and the gamma distribution. Also it consists of continuous versions of earlier mentioned discrete probability distributions like the uniform distribution.

Simulation output

For the output of the simulations, it is first important to gather statistics. To do this, DSOL has as set of commonly used practices to gather statistics. A good example is for instance the Counter, a simple mechanism that can, due to the modular structure of DSOL, easily be added to the desired element and keeps track of whatever needs to be traced for the purpose of the simulation. It by itself knows when to fire and can also automatically generate tables that store the results. Such a counter can be seen as a measurement of a simulation.

This table generation illustrates the output generation methods of DSOL, by building this once and a module, it can easily be included into other simulations.

Furthermore, DSOL consists of a GUI and can directly display these models to a non-technical user.

2.5.3 Agents

On the website of DSOL agent-based modeling (Verbraeck, 2017) is mentioned as an implemented, however in the source code and in Jacobs (2005), this discipline is not mentioned. Even the word 'agent' is not mentioned by Jacobs (2005). Due to the modularity of DSOL however, agent-based modeling can easily be implemented by using frameworks like OO2APL or JADE as the model.

2.6 Repeatability and reproducibility

The notions of repeatability and reproducibility are often used together, like in Minitab Inc (2016); Taylor and Kuyatt (1994). We will discuss the notions or repeatability used in the literature.

2.6.1 Repeatability

First repeatability. In Minitab Inc (2016), a publication about statistical tooling, repeatability is defined as:

> The ability of an operator to consistently repeat the same measurement of the same part, using the same gage, under the same conditions ((Minitab Inc, 2016) , p.1)

Here they speak about the subject of repeatability: measurements. With measurements they means the outcome of a measurement (i.e. the result). According to Minitab Inc (2016) repeatability is about the difference in results of the measurement. This publication does not describe what the difference may or may not be (i.e. deterministic, or from the same distribution).

The other part is about the same experimental conditions and gage, describes the second requirement of repeatability: similarity. The situation under which the measurement is performed may not change. Again, what similarity is, is not defined by Minitab Inc (2016).

Now that we have an intuitive idea about repeatability, let's look at it more in depth. The National Institute of Standards and Technology (NIST) (in the form of Taylor and Kuyatt) has defined repeatability as the following:

Definition 2.1 (Repeatability of measurements)**.** *Taylor and Kuyatt (1994)[D1.1.2 p.14] (NIST guidelines) Repeatability is the closeness of the agreement between the results of successive measurements of the same measurand carried out under the same conditions of measurement.*

Notes:

1. *These conditions are called* ***repeatability conditions***

2. *Repeatability conditions include:*

   - *the same measurement procedure*
   - *the same observer*
   - *the same measuring instrument, used under the same conditions*
   - *the same location*
   - *repetition over a short period of time*

3. *Repeatability may be expressed quantitatively in terms of the dispersion characteristics of the results.*

The above definition by Taylor and Kuyatt, which also includes these notes in the definition, gives guidelines about repeatability. These guidelines are for instance used in medical studies like Bartlett and Frost (2008), in which an overview of measurement techniques is presented.

Also from NIST, Villarrubia et al. (2003), use repeatability in their simulation study of CD-SEM. CD-SEM is a measurement system for 'measuring the dimensions of the fine patterns formed on a semiconductor wafer' according to High-Technologies Corporation (2017)[p.1]. They state:

> Repeatability is a measure of random error associated with a measurement, the amount of spread in the distribution of measured values when a measurement on the same sample is repeated many times (Villarrubia et al. (2003) , p.1)

Important to note is that Villarrubia et al. talk here about a distribution as an output rather than a value (which is a sample of the distribution). They compare two distributions of values and if these distributions are equal then they state that they are repeatable. This is a different interpretation of the equality of the results, because the equivalence is over the distribution rather than the values.

Also at (almost) the same time, McGregor worked on an overview of computer simulation (McGregor, 2002) in which repeatability plays a vital role. In computer simulation, models play a vital part: they represent the simulated system in the computer. About two runs of the same model he states:

> Two or more model runs will always execute in exactly the same way and produce precisely the same results if no parameters are changed between runs (McGregor (2002) , p.1684)

Here McGregor describes the equivalence of the input as also seen in the argument made by Villarrubia et al.. However, McGregor continues with:

> Any impression of randomness in a simulation model is due to the use of pseudo-random numbers to generate certain events such as breakdowns, cycle times and so on (McGregor (2002) , p.1684)

Here McGregor describes the input of a measurement as a distribution. Furthermore, this induces a correlation between the input distribution and the output distribution, since the randomness of the simulation model means the output of the simulation which is correlated to the random numbers that generate the events (i.e. the input). It shows a one-way implication: the input affects the output, not the other way around.

According to McGregor, repeatability is necessary to recreate and understand events during the model run and for debugging during construction. He also states:

> All events that influence the model execution are contained within the model and are therefore repeatable (McGregor (2002) , p.1685)

The above citation talks about what should be included in the scope of the initial state for repeatability. McGregor states about this that all events that influence the model execution should be included. He does not specify the exact events that exist in his model. But because he includes 'all', it can be deduced that he also includes the initial value (i.e. the seed) from pseudo-random number generators, since they also influence the model execution and should by McGregor's quote be included.

From the field of distributed simulations, Fujimoto describes repeatability for the purpose of describing simulation executions. He states:

> A distributed simulation is said to be repeatable if subsequent executions of the simulation using the same initial conditions and input as a reference execution produce exactly the same results (e.g., model statistics) as the reference execution (Fujimoto (1997) , p.2)

Again, here the same requirements for repeatability can be identified: the initial states between two runs should be equivalent and the results should be equivalent. He describes these equivalences as statistical equivalence, meaning that two results should samples from the same distribution. Meaning that unlike McGregor, he does not include the seed for pseudo-randomness in the model state but states that two inputs should be from the same distribution.

Also note that he states that parallelism does not matter for repeatability (by comparing it with a reference execution which is or is not a parallel execution). This is assuring since it allows us to ensure repeatability in parallel execution, which according to the description of repeatability of Fujimoto is possible.

Again, almost the same requirements, with a tiny difference in the interpretation of what is meant by 'equivalent'. So now we have seen parallel or distributed simulations define the notion of repeatability, however not yet in agent-based distributed simulations. However, in Riley and Riley (2003), the notion of repeatability is mentioned in this field. In a description of the results ran by the authors of this paper they write:

> [..] we repeatedly ran simulations with the same random seeds given to both the world model and the agents (Riley and Riley (2003) , p.824)

Based on this quote combined with the assured repeatability conditions, you can conclude that they do include the random seed in the scope of the initial state and the equivalence relation between the states. About the output, they state:

> In all cases, the results of the simulation in terms of the positions, the sensations, and the actions of all the agents are exactly identical (Riley and Riley (2003) , p.824)

Here they describe the output equivalence relation, for which they use a strict version that requires the outputs to be exactly the same (i.e. no samples from the same distribution). This is trivial due to the inclusion of the seed of the (pseudo) random number generator in the initial state's equivalence relation. They conclude by stating:

> It should be noted that the order of event realization is not identical (Riley and Riley (2003) , p.824)

The realization Riley and Riley make is critical, they do include parallelism, since they are writing about distributed agent-based simulations, but state that, just like Fujimoto (1997), parallelism does not influence the possibility of repeatability and in this case the order of event realization is not critical: we can allow some parallelism without harming repeatability.

Let us now summarize what we have seen so far. First we have seen a set of conditions under which repeatability is assured. We have seen the origins in the literature and seen how slightly different interpretations of these conditions are used throughout the literature. Furthermore we have described how in different subfields of the literature all the way to distributed agent-based simulations repeatability is defined and used.

2.6.2 Reproducibility

Let us now look at reproducibility, because in the literature this notion is used as well as repeatability. Sometimes the use of reproducibility is correct and sometimes it is not. Also, just like repeatability, Minitab Inc defines reproducibility:

> The ability of a gage, used by multiple operators, to consistently reproduce the same measurement of the same part, under the same conditions. (Minitab Inc (2016) , p.1)

Note already the difference with repeatability: reproducibility is about multiple operators. But also here the notion of similarity is not further defined by Minitab Inc for the sense of their statistical tooling. Luckily Taylor and Kuyatt again comes to the rescue with the following definition:

Definition 2.2. *Taylor and Kuyatt (1994)[D1.1.3 p.14-15] (NIST guidelines) Reproducibility is the closeness of the agreement between the results of measurements of the same measurand carried out under changed conditions of measurement*

Notes:

1. *A valid statement of reproducibility requires specification of the conditions changed*
2. *The changed conditions may include:*
   - *principle of measurement*
   - *method of measurement*
   - *observer*
   - *measuring instrument*
   - *reference standard*
   - *location*
   - *condition of use*
   - *time*
3. *Reproducibility may be expressed quantitatively in terms of the dispersion characteristics of the results.*
4. *Results are here usually understood to be corrected results.*

Let us analyze the above definition given by Taylor and Kuyatt, which also includes the notes in the definition. This states that conditions change and defines these conditions. Note that one of the possibly changed conditions here is the observer, as also mentioned by Minitab Inc. However, here this is only one of the conditions that might change, not necessarily should. By some interpretations it can be seen as that with a different observer also a different method or principle of measurement or some other conditions from Definition 2.2 is used. However Taylor and Kuyatt want to be very clear about this and define all of them.

In medical measurement, Bartlett and Frost describes reproducibility as:

> Reproducibility refers to the variation in measurements made on a subject under changing conditions (see definition 2.2). The changing conditions may be due to different measurement methods or instruments being used, measurements being made by different observers or raters, or measurements being made over a period of time, within which the error-free level of the variable could undergo non-negligible change (Bartlett and Frost (2008) , p.467)

In the above quote in reality Bartlett and Frost also refers to the definition of reproducibility in Taylor and Kuyatt (1994). Since it is a direct reference, it comes at no surprise to see that it also talks about changing conditions and the conditions that can change are somewhat similar (apart from the fact that Taylor and Kuyatt (1994) is more detailed).

What Bartlett and Frost states is that the variation of results is (almost entirely) due to the changed conditions. This is in line with what is stated by Lin. He states that:

> [...] the more the data are scattered [, the more it is] nonreproducible (Lin (1989) , p.255)

Defining here the relation between reproducibility and the variation in the results, Lin proposes a statistical degree of reproducibility based on this variation between results. The degree of reproducibility is subject to a set of parameters. Specifying the parameters correctly is key, one can imagine that with wrong parameters either everything or nothing is defined to be reproduced whenever the difference is even very minor.

To give another quote on what reproducibility is, let us look at what Santer et al. says about it:

> [Reproducibility is] the independent verification of prior findings (Santer et al. (2011) , p.1232)

This quote is from meteorological research, but still important. It namely shows the additional level of abstraction of reproducibility compared to repeatability. The 'independence' in Santer et al.'s quote clearly is about the conditions that might change and the following quote seems to confirm this:

> [Reproducibility is] the ability to independently verify the prior findings reported by an established model. The ability to independently replicate, reproduce and, if needed, extend computational artifacts associated with published work (Arifin and Madey (2015) , p.219)

The observation by Arifin and Madey is based on the verification philosophy of computer experiments described in Fomel and Hennenfent (2007) and Santer et al. (2011). When looking at this again, the independence can be interpreted as changing conditions.

Let us now summarize the notion of reproducibility from the literature. We have seen that reproducibility is about changing conditions under which the experiment is performed. Just like with the notion of repeatability the notion of reproducibility is about the variance of the results, although here the variance is primarily due to the phenomenon under investigation rather than the system. Furthermore, also two variations of looking at the equivalence of exist: one which defines equivalence based on values whereas the other defines the equivalence based on the distribution of which these values are samples.

Chapter 3

Repeatability and equivalences

Here we will define our own notion used throughout this thesis. So in this chapter we will bridge the gap between the literature and our work.

3.1 Repeatability

In section 2.6.1 we have given an overview of repeatability in the literature. We observe that repeatability is about the correlation between the input and the output of a system which adds a few requirements.

Different interpretations exist, which differ based on what means 'equivalent'. Mainly two views exist on this, a strict view which states that equivalent means the same value and a more abstract view which looks at values as samples from probability distributions and analyzes equivalence over these probability distributions. Without this interpretation though we can create a notion of repeatability:

Definition 3.1 (Repeatability)**.** *Let $f^{l,t}$ be a measurement function where l denotes the location and t the time, let p and q be two initial states such that two measurement results x and y are a result of measurement function f (resulting in $x = f^{l,t}(p)$ and $y = f^{l',t'}(q)$) and given an equivalence relation $\approx$, we call f repeatable if $p \approx q$ then $x \approx y$, when:*

- $l \approx l'$
- $t \approx t'$

Note from the above definition that we include all repeatability conditions from Definition 2.1: the measurement instrument, observer and its procedure f are the same (i.e. $f \equiv f$) in both measurements, the location of that measurement is equivalent (i.e. $l \approx l'$), the time is equivalent (i.e. $t \approx t'$), the input is equivalent (i.e. $p \approx q$) and the output is equivalent (i.e. $x \approx y$).

The measurement function f in the definition of repeatability can be anything. We try to ensure repeatability in the sense of large-scale agent-based social simulations and in that case f is a large-scale agent-based social simulation. The equivalence relation $\approx$ is given in this definition and the subject in section 3.3. For now it suffices to be able to say that there is an equivalence.

3.2 Reproducibility

Already mentioned is the relevant notion of reproducibility in section 2.6.2. In the literature the statement is made that conditions on the measurement have changed, such as the method of measurement or the observer. From this we define the notion of reproducibility as:

Definition 3.2 (Reproducibility)**.** *Let $f^{l,t}$ and $g^{l',t'}$ be measurement functions where l denotes the location and t the time, let p and q be two initial states such that two measurement results x and y are a result of measurement function f and g respectively (resulting in $x = f^{l,t}(p)$ and $y = g^{l',t'}(q)$) and given an equivalence relation $\approx$, we call f reproducible by g if $p \approx q$ then $x \approx y$*

We define the notion of reproducibility here to be able to show the exact difference between repeatability in this thesis. From the definition above one can note that the changing conditions requirement from Definition 2.2 is entirely included, the difference between f and g stands for the principle and method of measurement, the measurement instrument, reference standard, the condition of use, the time and the location.

Note the difference with repeatability. First of all, reproducibility is about two different systems f and g, where repeatability is about only one system. This difference between repeatability and reproducibility is formulated by the difference in the notation of the a strict equivalence $f \equiv f$ in Definition 3.1 (repeatability) whereas in Definition 3.2 $f^{l,t} \neq g^{l',t'}$ is possible.

Now that the formal description of reproducibility is clear, we can clearly see that for the purpose of ensuring equal or similar results in large-scale agent-based social simulations in which agents run concurrently repeatability is more what we need than reproducibility. Since also the observer, measurement instrument, measuring principles and condition of use are identical (i.e. we only make one simulation platform and model), we clearly have to deal with repeatability instead of reproducibility in the problem we are facing.

3.3 Equivalences

In Definition 3.1 we have given the equivalence relationship, here we will describe some equivalence relationships that are possible and discuss their behavior. By no means is this a complete list of equivalence relationships, but these are considered most relevant in the sense that they describe the different views of the literature on repeatability.

A quick recap shows us that the equivalence is over:

- The measurand
- The measurement procedure
- The observer
- The measurement instrument
- The location
- The repetition time period
- The results

In this part, the equivalences of these notions are covered and multiple different types of repeatability are defined.

3.3.1 Equivalent simulation assumption

We shall assume that we have the same scenario which we shall investigate. This means we assume that the same implementation of a simulation is used under the same conditions for the same purpose. So from the above repeatability equivalences, we assume that the measurement procedure, observer, instrument, location are identical. Also we assume, since we are talking about repeatability instead of reproducibility, that the difference in time period is so small, that we can state that the repetition time period of two independent simulation runs are 'similar'. We call this the *equivalent simulation assumption*.

Reason behind assuming identical equivalence has to do with the difference between repeatability and reproducibility, assuming that many of the above mentioned subjects change means a different implementation of the simulation and therefore brings us to reproducibility in the sense of large-scale agent-based social simulation, which does not serve the purpose of this thesis.

Time difference equivalence

First however we need a more formal statement when we call two repetition time periods close to each other. We shall denote a time period t as a range $[start, end]$ (where $start < end$). We identify the following time intervals that might arguably be possible:

- The duration of n repetitions of the experiment
- The duration of the research project of which the experiment is a subpart

For reasoning about these cases, assume for now that all other cases identified in Definition 3.1 are said to be equivalent. We then talk for instance about a research project that has just developed a large-scale agent-based social simulation in which all the equivalences are said to hold, except the one currently under investigation: the repetition time period. The question is whether or not

we call two measurements repeatable given that they for instance differ for more than one day?

The question then raises how small is 'small' in the statement that there must only be a 'small' difference in the repetition times. One could argue that more than one day is close enough. Because one may not make any changes to the implementation of the simulation (due to the assumption of the identical measurement instrument), the difference may not be that big. On the other case however, the runtime of the experiment is taken into account as well, making it better scalable.

CPU clock timestamp

For better determining what is a suitable difference in time interval, we shall identify the consequences of different time intervals. The most easily imaginable of any time difference is a different value of the internal clock of the CPU. This value represents the time the CPU is on and might be used on different locations in the system. Cases where this might influence the behavior of a computer are:

- When the timestamp is used for (pseudo) random number generation (i.e. the seed)
- When the timestamp is used for an application specific calculation (e.g. the duration of an event)
- When the timestamp is used for tie breaking (e.g. in multi-threaded situations)

In the first case for random number generation, if the timestamp is even slightly different, the seed will cause an entirely different stream of (pseudo) random numbers. Therefore, here it does not matter how big the difference in time is, it will differ anyhow.

In the second case, the timestamp also has no impact whatsoever, it is here all about the duration of the event, which can vary due to the amount of processing done at different moments in time.

In the last case, tie breaking is done through some process involving the timestamp of the CPU's clock. One can imagine that such a timestamp might be used to tie break. An example of this might be the following.

Given t is the timestamp and r is the result of the following calculation on the timestamp:

$$r = t \mod 2 \tag{3.1}$$

Then a simple mechanism might result in choosing option one if $r = 0$ and option two if $r = 1$. But as can be seen, also here the size of the time difference does not matter: every time difference can lead to entirely different behavior in tie breaking.

Processor load

Processor load is another thing that can vary. One can imagine that this may vary slowly over time, but also fast. Imagine a computer performing an experiment and in the background also some maintenance task that is totally independent of the experiment but does take processor power. The computer then has to assign processor time to both tasks. Again, this may change rapidly or slowly over time depending on that background task. Key is however that when the difference between the time intervals gets smaller, it becomes more likely that the same background task is still running and influences the experiment in a similar way.

Also processor load might influence certain metrics like runtime. Having background tasks running in one experiment and not in another experiment changes the measurement instrument and therefore one might argue that this is indeed a violation of the repeatability conditions and so do we.

Summary of time period equivalence

We have seen that identifying what generates noise due to differences in time is possible, it has been done above. The conclusion of what this has for a consequence is also possible: it might (given that no synchronization techniques are implemented) cause deviations in the results which cause unrepeatability of the results. However, we have also seen that the difference in time intervals has no big influence on the notion of repeatability. It can have a very disturbing effect on the repeatability, but often it already has a disturbing effect from even the smallest imaginable change (e.g. adding 1 to the timestamp).

However, for now we will create a definition of the time period equivalence according to the most strict version we have proposed above, based on the runtime of an experiment. This form is suitable by the DSOL simulator used, which consists of an inbuilt replication mechanism which performs a given amount of replications of the same experiment under the same circumstances. We state therefore that the experiment time is the time the simulator needs to perform two experiments of n replications (often 30 is used for statistical significance).

Definition 3.3 (Time period equivalence)**.** *Given two time periods $a = [a_{start}, a_{end}]$ and $b = [b_{start}, b_{end}]$ of two independent experiments (i.e. $a_{start} < b_{start}$), we define these two time periods time period equivalent in the sense of repeatability in large-scale agent-based social simulations if and only if:*

$$b_{start} - a_{end} \leq r$$

Where r is the average runtime, defined as:

$$r = (a_{end} - a_{start}) + (b_{end} - b_{start})$$

Note from the above definition we state that the difference in time may not be greater twice the duration of an experiment (i.e. all 30 replications of the same experiment). This strict notation is plausible by the inbuilt simulator replication

system in DSOL (Jacobs, 2005), which reschedules the same simulation again in milliseconds. So only when an simulation takes less than a millisecond this equivalence is not plausible for us, which seems unlikely given that the creation of a multi-agent system combined with the DSOL simulator already takes 133.5 milliseconds (average for the creation of our own experiments).

Furthermore, we will use in the experiments of this thesis a degree of repeatability, which shows the notion of repeatability empirically, where the time difference again is about milliseconds between finishing experiments. Making it again plausible to use the above definition.

Summarizing the equivalent simulation assumption

Now knowing all the components needed for the equivalence relation of time periods we can finish building the formal notation of the equivalent simulation assumption.

We use the assumption for the measurement instrument, location, observer and measurement procedure that they are used in both experiments and therefore are the same. This is because we assume that the same simulation (i.e. same implementation of the same multi-agent system) is used in both runs. This makes that we assume for the observer (i.e. the metrics gatherer in the simulator and the measurement procedure), the measurement instrument (i.e. the simulation model with the multi-agent system) and the location (i.e. same computer/operating system) that they are the same. We need with this equivalence the notion of repeatability instead of reproducibility, which is exactly what is desired.

Assuming the equivalent simulation assumption, this leaves us with two main subjects we reason about in the remainder of this chapter: the measurand and the results. We shall, as a convention, state that the measurand is the input of the experiment of the simulation, denoted I, and the results are the output of the experiment of the simulation, denoted O.

3.3.2 Deterministic repeatability

The strongest type of equivalence imaginable is the deterministic equivalence, meaning that O and O' are equal if and only if:

$$I \equiv I' \tag{3.2}$$

For repeatability this means that, with the equivalent simulation assumption, we need to have deterministic equivalence on the result as well:

$$O \equiv O' \tag{3.3}$$

By default, using the serial assumption in computers used in Duffy (1992), every serial program has this type of deterministic repeatability because the computer as we know it is a Turing Machine (Copeland, 1993).

The serial assumption used in Duffy (1992) and also described in McCool et al. (2012) states that programmers assume that a set of operations is performed strictly serial. Meaning that given a set of operations there exists a strict order meaning:

Definition 3.4 (Serial assumption)**.** *Given a set of actions $A = \{\alpha_1, \alpha_2, .., \alpha_n\}$, a trace t is a sequence of actions from A. We define that the serial assumption holds on trace t if and only if $\forall x, y \in t : x < y$ or $y < x$ (where $<$ is performed based on time).*

For this assumption to hold, McCool et al. (2012) states that one must perform at most one action at one time, otherwise one cannot ensure the relation $<$. Speaking in terms of computer, this means that at most one processor may be used, which is a problem often made by design in computer programs according to McCool et al. (2012). This design fault is not something specific to our problem.

We can state that the deterministic repeatability is described as:

Definition 3.5 (Deterministic repeatability)**.** *Given a large-scale agent-based social simulation, the input of that simulation denoted I and the results of that simulation denoted O, we state that the output of that simulation is deterministically repeatable under the equivalent simulation assumption if for two runs with inputs i and i' (where $i \equiv i'$) for the outputs o and o' it holds that $o \equiv o'$.*

3.3.3 Probabilistic repeatability

A different form of repeatability we define is that of probabilistic repeatability. Here the philosophy is that different values can be from the same probability distribution and the equivalence relation is on the distributions.

But what is a probability distribution? We have described in section 2.5.2 that they state something about a chance that something has a certain value. Formally this means:

Definition 3.6 (Probability distribution)**.** *Let $O = \{o_1, o_2, .., o_n\}$ be a set of outcomes and $P = \{p_1, p_2, .., p_n\}$ a set with the corresponding probabilities, we call $X = \{(p_1, o_1), (p_2, o_2), .., (p_n, o_n)\}$ a probability distribution where:*

$$\sum_{p \in P} p = 1$$

Note from the above definition that all probabilities, which correspond to the chance that of all values of O value o_i occurs, together sum up to 1. This is always the case with probability distributions. We also mentioned that there exist discrete and continuous distributions. We can model both types of distributions because both O and P can be infinite sets. Different types of distributions exist like uniform and normal distributions, which can all be represented using this definition, the differences between these types consist of different formulas

for calculating the mean for instance, which is not relevant for our purpose of reasoning about equivalences, but it can theoretically be defined.

As an example suppose the values a, b, c are all samples from a distribution $D = (O, P)$, which means that $a \sim D$, $b \sim D$ and $c \sim D$ where in this case the operator $\sim$ is from probability theory meaning 'sample of' (i.e. if $a \in O$ then $a \sim D$). Because it are samples it might well be possible that $a \not\equiv b$ for instance. With deterministic repeatability this would lead to a violation and therefore the two outcomes, even if we have not even considered them yet, not repeated.

Suppose we have two experiments that pick samples from two distributions D and D', we state that these are the inputs of the experiments, meaning the equivalence relation is over these. We state that $D \equiv D'$, since we are still talking about distributions we call this the equivalence relation for probabilistic repeatability.

The same goes for the results of the simulation. Because the inputs are considered equivalent, so should be the outputs. Because the input values are samples from a distribution, so should be the outputs. We call the distribution of this output X and state here we have to deal with the notion of probabilistic repeatability, meaning that given two equivalent input distributions and two corresponding output distributions the system that produces these distributions is considered probabilistically repeatable. Formally speaking:

Definition 3.7 (Probabilistic repeatability)**.** *Given a large-scale agent-based social simulation, the input probability distribution of that simulation denoted I and the result distribution of that simulation denoted R, we state that the output of that simulation is probabilistically repeatable under the equivalent simulation assumption if for two runs with inputs i and i' (where $i \in O_I$ and $i' \in O_I$) for the outputs o and o' it holds that $o \sim O$ and $o' \sim O$.*

Correlation input and output

We state that the input distribution D and output distribution X are correlated, since an arbitrary sample from D (say $a \sim D$) results in a result sample $b \sim X$. Since the simulation is run on a computer, which is a Turing machine, the output is (given the same input) always the same. This is with serial execution always the case, however for parallel execution, the whole existence of the problem of ensuring repeatability shows that it is not required, therefore we have to assume that correct measures have been taken (i.e. synchronization techniques implemented) that ensure deterministic repeatability.

Assuming this, we can see that a mapping occurs between values from the input distribution to the output distribution. We specify this mapping as a function:

Definition 3.8 (Deterministic input and output mapping)**.** *Given a large-scale agent-based social simulation, which takes inputs from an input distribution I and results according to distribution X, we define a mapping between samples*

from I (e.g. $a \sim I$) to samples from X (e.g. $b \sim X$) as a function $m : I \to D$ defined as $m(a) = b$

This mapping is what we reason about. Since given a sample $a \sim D$ (i.e. the input) it corresponds to the sample $b \sim X$, which is the output, (of the output by this mapping (i.e. $m(a) = b$) which are exactly the input and output conditions this chapter is about.

When it is possible to make such a mapping we state that repeatability is ensured in the most strict fashion: deterministic repeatability. Since given the same input, the same output is produced. However, when we drop the assumption we can have that $m(a) = c$ or $m(a) = b$ (where $c \neq b$), which basically states $m(a) \neq m(a)$, which is a problem because it allows internal noise in the program and that means that it is not simulating the real world system but has internal deviations by the execution. If a real world system does have noise, it means it is part of the initial state making that it is not the same initial state (i.e. $m(a)$ vs $m(b)$) meaning that we do not have $m(a) \neq m(a)$. Therefore we state that without the assumption of determinism within the mapping such a mapping as defined in Definition 3.8 does not exist.

When this is the case another mapping is possible, the one we call the probabilistic input and output mapping:

Definition 3.9 (Probabilistic input and output mapping)**.** *Given a large-scale agent-based social simulation, which takes inputs from an input distribution I and results according to distribution X, we define a mapping between samples from I (e.g. $a \sim I$) to samples from X as a function $m : I \to D$ defined as $m(a) \sim X$*

The nuance here is that the mapping $m(a)$ is a sample from X, but not defined as a specific sample. This might be a small adaption but opens up a whole new range of opportunities. But when implemented incorrectly and specific measurements in the simulation are used, it can also lead to internal noise in simulations by which it causes internal deviation by the model, not the modeled behavior of the simulation.

To show the new range of opportunities, suppose an auction with n bidders according to an English auction (i.e. highest bid wins and everybody gets informed about all bids). Suppose the bidder which is prepared to pay the lowest price is bidder 1 and this is ordered all the way up to the bidder who is prepared to pay the highest price, which is bidder n. Suppose this prepared price (denoted p_i for agent i) is the upper bound to what an agent will bid. Now consider the output of such a large-scale agent-based social simulation to be the price agent n pays for the item for sale in the auction. When ensuring repeatability within the protocol of the auction, we ensure that the agent with the highest price (i.e. agent n) will always win the auction, however the value the agent pays can, when not ensuring deterministic repeatability but probabilistic, vary between $[p_{n-1}, p_n]$. This range is the distribution of results (i.e. X from the formal definition).

By this example, we can see that we still ensure deterministic repeatability within the protocol of the auction: the same agent wins the auction. But the

price he pays for the highest item may vary: in one run the auctioneer receives the bid from agent i for instance earlier than that of agent j whereas in another run this is the other way around. Key here is when this happens near the last bids. Suppose that every agent increases his bid by 2 every round. Then when the auctioneer announces the price of $p_{n-1} - 1$ agent $n - 1$ will place a bid of p_{n-1} and agent n will place a bid of $p_{n-1} + 1$ (assuming there is a difference between $p_n - p_{n-1} \geq 1$). The order in which these two bids are processed by the auctioneer is very important, there are two cases: first process the bid of agent $n - 1$ or first process the bid of agent n.

If the auctioneer decides to process the bid of agent n first, the auction is finished. Because no agent will bid higher than n's bid of $p_{n-1} + 1$ since it is definitely higher than $n - 1$ is prepared to pay.

The other case is that first the bid of agent $n - 1$ is processed. In such a case the auctioneer announces this bid of p_{n-1} to all agents (only agent n will respond because all other agents have a p lower than the current price) and agent n responds with a bid with the price:

$$p_{bid} = \begin{cases} p_n & \text{if } p_n \leq p_{n-1} + 2 \\ p_{n-1} + 2 & \text{otherwise} \end{cases} \tag{3.4}$$

In this case we can see that there are two possible values from the distribution of X (i.e. the values of p_{bid}), this increases when instead of 1 there are more agents with a price of p_{n-1}.

3.4 Summary

In this chapter we have created our own notion of repeatability for use throughout this thesis and combined it with the notion of large-scale agent-based social simulations. Furthermore we have defined different types of repeatability: deterministic and probabilistic. About these two types we have reasoned and shown what their behavior might be, while we still consider them repeatable.

Chapter 4

Synchronization techniques

In this chapter we talk formally about the synchronization techniques, therefore we first describe repeatability in the sense of traces and measurements (for simulations) to continue with examples of problems with repeatability that can emerge when we do not include correct synchronization techniques. We will focus purely on the theoretical side and proof formally that these synchronization techniques work.

4.1 Formal Language

For describing the agent platform, we need a formal language for describing these agents. Specifically, we are in the terms of repeatability interested in processes. Since repeatability is about program runs, or traces, which are a result of processes, we are looking for formal languages describing processes. More specifically, repeatability in large-scale agent-based social simulations is about measurements and measurements are the result of a process. So when reasoning about the measurements, we reason about the processes, or process equivalence. By process equivalence we mean, equivalence of the traces or runs the process can produce.

Languages for describing processes are also known as process algebra. Multiple ones exist including:

- Petri nets (Petri, 1962)
- Partially ordered sets (Dushnik and Miller, 1941)
- Communicating Sequential Processes (Hoare, 1978)

Note above that we choose partially ordered sets for comparison here. We could have also included Basic Process Algebra (BPA) by Fokkink (2007) here. However these two are very similar and based on how often they are used in the literature (i.e. citations of Fokkink (2007) vs. Dushnik and Miller (1941) and related publications like MacNeille (1937); Trybulec (2000); Trotter (1995,

2001) compared to Groote and Huttel (1994); Fokkink and Zantema (1994); Bergstra and Klop (1984)), the choice is made to use Partially ordered sets.

To compare which process algebra is best suited for the desired purpose, we should identify our needs. There are a few requirements that the language should have, namely:

- It should be able to model relative time
- It should be able for different actions to take longer or shorter than others
- It should be able to model concurrency
- It should be able to model synchronization
- It should be able to model shared resources

These requirements are based upon what large-scale agent-based social simulations need. We will discuss them one by one.

The first requirement is needed because the model some real world system which means it should model time, because for repeatability timing (or time relative to other actions in processes) is important for action conflicts. Also the second requirement is about time, but then the timing of actions. These requirements can be added in CSP by two events: start and finish. These start and finish events can have an arbitrary time between them, as with the relative time in the form of comparing traces. For Petri nets it is more difficult because of the inbuilt marking transition mechanism for modeling the duration of actions, which automatically fires transitions that are active. Partially ordered sets can, just like CSP, model time in the same flexible way but lack the possibility to model flow (e.g. markings in Petri nets).

Another requirement is that of concurrency (i.e. causal independent). This is needed because we talk about large-scale agent-based social simulations, meaning that agents can run concurrently. Because all process algebras have an internal graph-like structure, one can model concurrency as two or more (partially) disconnected processes (see Figure 4.1).

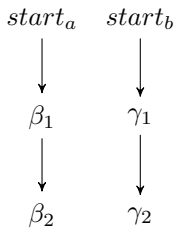
Also CSP can describe concurrency, however it uses equations for it. The example denoted in Figure 4.1, is denoted in CSP by equations 4.1, 4.2 and 4.3.

$$A = start_a \rightarrow \beta_1 \rightarrow \beta_2 \tag{4.1}$$

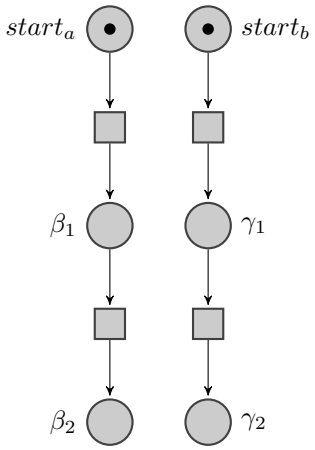
$$B = start_b \rightarrow \gamma_1 \rightarrow \gamma_2 \tag{4.2}$$

$$A||B \text{ (where } \alpha A \neq \alpha B) \tag{4.3}$$

The above equation 4.3 requires the given constraint of $\alpha A \neq \alpha B$. The sets αA and αB denote the alphabets of the processes. This alphabet is the set of events the process can handle from now up until the process is terminated (or



(a) Partially ordered set of two agent example



(b) Petri net of two agent example

Figure 4.1: Examples of two agents running concurrently in partially ordered sets (4.1a) and Petri nets (4.1b)

if it does not terminate forever). For an arbitrary process P the alphabet is denoted αP.

The requirement that $\alpha A \neq \alpha B$ is because when the opposite is true, we talk about interleaving in CSP. Therefore another operator exists, denoted as in equation 4.4.

$$A|||B \text{ (where } \alpha A = \alpha B) \tag{4.4}$$

The fourth requirement is about synchronization. This is important for solving repeatability conflicts. Synchronization means requiring cooperation between agents in order to ensure something as a group. Specifically speaking, suppose two agents A_1 and A_2 want to access a shared resource, due to synchronization one can ensure that the resource can only be accessed by one agent at one time. A violation of this, will possibly violate repeatability. Two languages have synchronization embedded: Petri nets and CSP. Petri nets use for this the transitions, which only fire when all ancestor states have a marking of at least 1. CSP uses for this the notion of the synchronization operator $|\{..\}|$, which allows processes to halt until another process is ready.

In partially ordered sets synchronization does not exist explicitly and should be added in the form of guards or some other additional formal notation to ensure synchronization.

The last requirement is about shared resources. Agents can have shared resources, think about the environment for instance, and modeling these in the language is critical. With CSP this is no problem by the use of the synchronization operator $|\{...\}|$ which can denote shared resources and synchronization of it. However Petri nets and partially ordered sets do not model resources in their languages.

Chosen language

From the survey above we decided to use is CSP. Reason for this is that it is flexible enough to model concurrent processes and specific enough to model things like synchronization. We use a few terms and notation that might look unfamiliar, so we discuss these first.

First let us start with what a process looks like. A process P that can process or handle an event x and then stops is denoted as:

$$P = x \rightarrow STOP \tag{4.5}$$

Here $STOP$ is a special type of process, meaning the process terminates. When a process can handle event x and then continues to behave differently it is denoted as:

$$P = x \rightarrow Q \tag{4.6}$$

We call in the notation above P the predecessor process of Q. Furthermore, also recursion is possible, namely when we replace in the above equation Q for P:

$$P = x \rightarrow P \tag{4.7}$$

The above means that process P can endlessly continue processing events of type x. When we introduce a second event, namely y making it is possible to perform either x or y. We denote it with the choice operator $|$:

$$P = x \rightarrow P | y \rightarrow P \tag{4.8}$$

Of the above process P we call αP the alphabet. In the example this means that $\alpha P = \{x, y\}$. The alphabet is the set of all possible events a process can perform.

4.1.1 Agents

Let us now look at an agent to start with. An agent is an autonomous entity (Shoham, 1993) and could therefore easily be modeled as an independent process in CSP, say A_i. In the sense of a large-scale agent-based social simulation, multiple agents exist. When multiple agents together exist in a system, this is called a multi-agent system (Van der Hoek and Wooldridge, 2008).

Definition 4.1 (Multi-agent system)**.** *Given a set of agents* $A = \{A_1, A_2, A_3, .., A_n\}$*, a multi agent system is denoted as:* $MAS = A_1||A_2||A_3||..||A_n$

Note here that we use the parallel operator $||$, this means different agents can have different alphabets, meaning agents can be of different types.

4.1.2 Triggers

Now that these basic principles are discussed, let us look at the agents more in detail. Specifically about the alphabet. Based on Dastani (2008); Dastani and Testerink (2014), we know that an agent can receive multiple types of triggers. Namely:

- Internal triggers
- External triggers
- Messages
- Goals

These different types of triggers are used for the behavior of the agent, all describing different types of behavior. Internal triggers for instance are triggers sent by an agent to itself and represent a single event. An external trigger is a single event that is received by an agent. Messages are events sent by other agents to the agent. Different than triggers, which are consumed after processed, goals are persistent: they continue until they achieved or stopped when the agent finds out it is impossible to achieve the goal. These triggers are used by the agent to create a perception of the world.

4.1.3 Deliberation cycle

2APL and OO2APL use a deliberation cycle that handles all these types of triggers described above. However, this handling of a trigger can best be seen as a process of sense, reason and act (in this order) like in 3APL (Dastani et al., 2005). Here sensing means receiving the triggers, reason means determining what to do with it (i.e. which plans to select) and act means executing these plans. Therefore, since we are talking about behavior, we define the deliberation cycle as a cycle of sense-reason-act. Formally defined as:

Definition 4.2 (Deliberation cycle)**.** *Given the events sense, reason and act, which denote the different stages of the sense-reason-act cycle, we define the deliberation cycle D as the process:*

$$D = sense \rightarrow reason \rightarrow act \rightarrow D | sense \rightarrow reason \rightarrow act \rightarrow STOP$$

Note here that a cycle has to finish completely before an agent can terminate, this is a default implementation for instance used in OO2APL by Dastani and Testerink (2014) for when there are no active goals to pursue. The $STOP$ process is a default process in CSP denoting that the process has finished.

We state that the deliberation cycle of the agent in an unmanaged unsynchronized environment (i.e. the default setting of a concurrent multi-agent system) is the whole behavior of the agent, therefore the agent process is the deliberation cycle:

$$A_i = D \tag{4.9}$$

Note that a sense-reason-act cycle is a deliberation cycle for a reactive agent, since no planning is involved. However, with a small difference in interpretation proactive behavior is possible. To do this, we introduce something we call persistent sensing. It means we suppose that a goal trigger is rescheduled in the act-phase again when the goal remains active, this way proactive behavior in the form of goals is handled and we can use this definition for proactive agents as well. We do not need to adapt the definition of the deliberation cycle for this. We just assume that whenever an agent pursues a goal that has not finished, in act it will continue to hold this trigger and therefore initiates another *sense*-event.

Sense-reason-act specification

Let us further specify what exactly is part of each phase of the sense-reason-act cycle. Starting with sense. During the sense-phase, the following happens:

- Messages are received
- The current state of the environment is 'sensed'
- External triggers are received

Receiving a message in this phase means that the agent becomes aware of the fact that it has received a message. All agents have an 'inbox' in which agents that send messages put the message. When the agent senses the new messages it simply means that it checks this 'inbox' for new messages. The same goes for the external triggers, which have a separate 'inbox'.

In the reason-phase, the following happens:

- The belief base is updated based on the information it has 'sensed'
- Internal goal logic is executed, meaning:
  - Check whether the goal should still be pursued
  - Check whether the goal needs to be suspended
  - Check whether the goal is achieved
  - Perform goal specific reasoning

Most important here is that in the reason-phase the beliefs are updated based on internal logic. Think of this as a set of rules according to which the agent handles, this handling is the act-phase, so only the checking of the rules happens here. The result of the rules are executed in the act-phase. In the act-phase the following happens:

- The environment is modified
- Messages are sent

4.2 Measurements and Traces

We start of with a program, since we are talking about a computer simulation. This is a process according to Hoare (2015) in CSP, in our case a special type of program: a large-scale agent-based social simulation. A process according to Hoare is defined by its behavior. A simulation run is therefore a trace of events.

A trace is a sequence of events also known as a program run, or in the sense of simulations, a simulation run. This means we reason about events within this sequence, therefore we will need to define the operator for this sequence:

Definition 4.3 (Element in sequence operator)**.** *Given a sequence $s = e_1e_3e_2...e_n$ which consists of the set of events $E = \{e_1, e_2, ..., e_n\}$, we define the operator $\in$ as that if $e \in E$, then $e \in s$.*

Now we come to defining traces, formally speaking:

Definition 4.4 (Trace)**.** *Given a process P, we define t a trace if t is a (possibly empty) sequence of events in the order they occur, so $\forall e \in t : e \in \alpha P$.*

Note here that a trace is a sequence, meaning that $e_i \in t$ and $e_j \in t$ (where i and j denote the indexes in the sequence) may be the same event (i.e. $e_i = e_j$ where $i \neq j$), but only a different instance.

Numerous traces may exist for a process. From CSP (Hoare, 2015)[sec 1.8] comes the function *traces*, formally defined:

Definition 4.5 (Traces function by Hoare (2015))**.** *Given an arbitrary process P, we call the set of all possible traces T (see definition 4.4) the outcome of function traces $: P \to 2^T$ such that traces(P) is a set of traces.*

Note from the above definition that one of the possible traces is indeed the empty set ($\emptyset \in traces(P)$). A process can for instance have a case in which none of the events in αP fires and therefore this is also a possibility to take into account.

Given that a process consists of one agent, namely A_i, we state that a trace $t \in traces(A_i)$ is the sequence of events in the order in which it is executed. We assume that agents are internally serial, meaning that there exists only one ordering of events. We do this for a number of reasons.

Firstly, since the reality (OO2APL and 2APL) also include this in the execution of agents. 2APL does not include explicit parallelism, but OO2APL does (by the number of processors assigned to the Platform-class). However, this parallelism only exists between agents, internally they are implemented serial.

Secondly, for simplicity since it drastically decreases the amount of theoretical possible traces and we do not have to deal with the ordering of execution of plans of the same agent.

Thirdly, the repeatability primarily occur in real life when agents interact. Think for instance about non-repeatability as a result of a different ordering of tasks between humans. It is unlikely that such problems occur in the same human, since he or she is in control of that ordering. It therefore is in line with

the philosophy of seeing agents as highly intelligent, autonomous and rational entities to assume that agents are internally serial.

The magic word was already mentioned a few times: ordering. A trace is an ordering of events. Such a trace ordering is defined as the sequence ordering:

Definition 4.6 (Sequence ordering)**.** *Given a sequence $t = e_1 e_2 ... e_n$, we define the ordering of sequence t, denoted $\leq_t$ as the ordering based on index, meaning that for two events e_i, e_j if $i \leq j$, then $e_i \leq_t e_j$.*

Since a trace is a sequence, this implies that the ordering where this definition is about, talks about trace ordering as well. When we talk about the order of traces we will use the notion of trace ordering to imply that we talk about traces, otherwise we talk about the ordering of a sequence.

Note from the above definition that with e_1 and e_2 we are talking about instances, not the event types. For instance it can be the case that an action exists for multiple deliberation cycles, making an appearance in the trace more than once.

Coming back to the problem we are trying to solve: repeatability in large-scale agent-based social simulations. Within simulations we talk about measurements, for which we are trying to ensure repeatability. For this, let us define a measurement criterion, which is defined on the large-scale agent-based social simulation:

Definition 4.7 (Measurement criterion)**.** *Given an arbitrary large-scale agent-based social simulation P, we define the measurement criterion as the result of a function mc that maps a process to a finite nonempty sequence of events (where all events exist in αP).*

Note from the above definition that the measurement criterion measures the relevant events that together define an outcome. Think of the outcome for instance as a number whereas the events can be $+1$ and -1, the sequence of these events together form this number from a given initial value. This outcome is what we call a measurement, defined as:

Definition 4.8 (Measurement)**.** *Given an arbitrary large-scale agent-based social simulation P, an arbitrary measurement criterion E_P (i.e. a sequence which is the result of an arbitrary $mc(P)$) and the set of traces $T = traces(P)$, we call the measurement a function $m_{E_P} : T \to \{\top, \bot\}$, defined as:*

$$m_{E_P}(t) = \begin{cases} \top, & \forall e, e' \sqsubseteq E_P : e \leq_{E_P} e' \Leftrightarrow e \leq_t e' \\ \bot, & otherwise \end{cases}$$

Note from the above that we use the sequence ordering for $\leq_{E_P}$ and $\leq_t$ and since t is a trace, we can also call $\leq_t$ the trace ordering operator.

Also note that this measurement checks whether the value is correct. If it is correct and for all possible traces it is correct, we call the measurement repeatable. The problem of ensuring repeatability therefore is:

Definition 4.9 (Problem of ensuring repeatability)**.** *Given an arbitrary large-scale agent-based social simulation P, an arbitrary measurement criterion E_P, the measurement m_{E_P}, we call the problem of ensuring repeatability the problem that $\exists t, t' \in traces(P) : m_{E_P}(t) \neq m_{E_P}(t')$.*

Different types of measurements criteria exist, for which this problem of ensuring repeatability exists. One of them is the act-sense measurement criterion, defined as:

Definition 4.10 (Act-sense measurement criterion)**.** *Given an arbitrary large-scale agent-based social simulation P and an arbitrary measurement criterion on that system called $E_P = mc(P)$, we call the E_P an act-sense measurement criterion if all events $e \in E_P$ are about tasks that are preformed in either the sense or act-phase of a deliberation cycle and where the effects of the act-events are sensed by the sense-events.*

Note from the above definition that we here talk about actions of which the results are sensed by the sense-events in E_P, meaning that in E_P there is at least one sense-event and at least one act-event.

An example of such an act-sense measurement is communication, where the act-event is an event that sends a message and the sense-event is the receiving of the message. Problems with repeatability can occur when it is a possibility that in one trace the act-event is performed after the sense-event and the other time it is not.

Another application is that of modifying the environment and where this modification is sensed by another agent. Think for instance about the environment as a door which can be opened or closed. One agent opens the door (i.e. an act-event) and the other checks the state of the door. The problem is that one cannot assure that another agent has already sensed the current position of the door (i.e. open or closed), so in one trace it is possible that the other agent reasons about the open door whereas in another trace it reasons about the closed door.

Another type of measurement criterion is the Act-only measurement criterion, defined as:

Definition 4.11 (Act-only measurement criterion)**.** *Given an arbitrary large-scale agent-based social simulation P and an arbitrary measurement criterion on that system called $E_P = mc(P)$, we call the E_P an act-only measurement criterion if all events $e \in E_P$ are actions tasks that are preformed in the act-phase of a deliberation cycle.*

The above definition is about events in the measurement criterion that only regard modifications. For example think again about the environment as a door, this door can be opened and closed by two agents. One agent wants to open the door and the other wants to close it. The measurement is the outcome position of the door, which depends on which agent performed its action (i.e. closing or opening) last.

External influences on the multi-agent system

It is possible to influence a multi-agent system from outside the multi-agent system itself. For instance in Dastani and Testerink (2014) they use external triggers for this. When these external triggers are part of a measurement criterion we argue that these are part of the class of act-sense measurement criteria, since the receiving of these external triggers is a sense-event. The act-event, the one that sends these external triggers is however an external event. Formally we therefore add the external actor as another agent in the multi-agent system making it possible to include these external influences in the same class of measurement criteria.

Practically we then only need to include such an external actor within the platform that runs the multi-agent system.

Other measurement criterion types

We have introduced two measurement criterion types, the question therefore raises whether these are the only two relevant types for repeatability. We argue that this is the case. Let us therefore look at the other possibilities within the sense-reason-act cycle.

First look at measurement criteria like the act-sense measurement criterion. This is an criterion that occurs between different phases of the sense-reason-act cycle. We already looked at the relation between the act-phase and the sense-phase, other possibilities are the relations between sense-reason and reason-act. Sense and reason are both phases that do not have a publicly available result, because the result of sense is that the agent's perception of the world is updated and the result of reason is that one has made conclusions about the agent's current perception. Because these results are not public other agents do not depend on them and because agents are assumed to be internally serial, they do not form a problem.

The same goes for measurement criteria like the act-only measurement criterion. This criterion is about the ordering of events within one phase of the sense-reason-act cycle. For sensing this does not form a problem, think for instance about sensing a certain value of the environment. Here it does not matter whether one agent senses it first or another, as long as the value does not change, which means there should be an event from the act-phase between the two, which would make it an act-sense measurement criterion.

Also for reasoning the ordering of events in this phase does not matter because of the same argument that the results are only visible internally and by the assumption that agents are internally serial this causes no problems for repeatability.

So therefore we conclude that the two measurement criterion types are indeed all the possible types that are relevant for repeatability.

4.3 Synchronization

Now knowing the measurements and problems for repeatability that occur for different types of measurement criteria we can start looking for a solution. This solution means that we shall synchronize at certain points in the multi-agent system to ensure repeatability.

4.3.1 Synchronization operator

For describing synchronization techniques we first have to describe the most important operator in CSP for us: the synchronization operator. We define it, based on Hall (2002) (who calls it the parallel interface).

Definition 4.12 (Synchronization operator)**.** *Given two arbitrary processes $P = a \rightarrow P'$ and $Q = b \rightarrow Q'$ and its predecessor processes $X = c \rightarrow X | c \rightarrow P$ and $Y = d \rightarrow Y | d \rightarrow P$ that together form system $S = (X \rightarrow P)||(Y \rightarrow Q)$, we define the synchronization operator $|\{sync\}|$, which is an extension of the parallel operator $||$, for processes P and Q in the synchronized $S' = (X \rightarrow sync \rightarrow P)|\{sync\}|(Y \rightarrow sync \rightarrow Q)$ where the trace ordering $\leq_S$ has the properties:*

- $\forall t \in traces(X), \forall e \in t : e \leq_{S'} sync$
- $\forall t \in traces(Y), \forall e \in t : e \leq_{S'} sync$
- $\forall t \in traces(P), \forall e \in t : sync \leq_{S'} e$
- $\forall t \in traces(Q), \forall e \in t : sync \leq_{S'} e$

Intuitively the above definition states that given two processes that together form a process $P|\{sync\}|Q$, P and Q can run independently, however when one encounters the processing of event *sync*, both P and Q must process it at the same time, so either P has to wait for Q or vise versa.

Note that due to the fact that this operator is an extension to the parallel operator that it will behave as such for all events that are not mentioned in the synchronization operator.

Also we use the operator $||$ because we talk about arbitrary processes and $\alpha P \neq \alpha Q$ is often the case in here, we consider $\alpha P = \alpha Q$ as a special case which we solve by stating that different agents have different instances of for instance sense-events (because they are of another agent), this way it is for large-scale agent-based social simulations always the case that $\alpha P \neq \alpha Q$ for any P and Q that are different instances of agents. This has as an advantage that we can use it for multi-agent systems where different types of agents behave together.

4.3.2 Act-sense synchronization

Based on the problems that can occur regarding repeatability for the act-sense measurement criterion, let us try to solve these. To do this, let us describe the problem. From this we form a formal problem which we will then solve.

Example

This example is already mentioned briefly in section 3.3.3 and is based on an auction. Suppose two types of agents exist in an auction: the auctioneer and the bidder. The auctioneer has the following capabilities:

- Initiate the auction
- Announce the current highest bid and its bidder
- Receive bids
- Sell the item for sale

Here the initiation of the auction sends the initial announcement message to start the auction. When handling the receiving of bids, the auctioneer checks whether the received bid is higher than the currently highest bid and if this is the case it will announce this bid as the highest received bid. If it has not received a higher bid for n deliberation cycles (specified by the auctioneer and often chosen as 3) the auctioneer will sell the item to the highest bidder.

Opposed to the auctioneer, the bidder has the capability to:

- Receive bidding announcements and respond to it (or not if the price offered is too high for the bidder)

Here the bidder is a so called reactive agent: it does not plan by itself (i.e. it has no goals), it will only respond to the announcements it receives. Every bidder has determined a value it is maximally prepared to pay for the item for sale and will not bid higher than this value. If the value is lower it will always bid, although here the policy is that the first bid that the auctioneer receives will be accepted.

In this example the measurement criterion is who is winning the auction, which depends on the communication. Communication depends on the messages agents send and receive and is therefore an act-sense measurement criterion, more specifically the criterion consists of the sending of a bid (i.e. the action) and the receiving of it (i.e. the sense) which results in the auctioneer updating the highest price and sending an announcement. The sequence of these actions together form the eventual outcome of the simulation: the agent that is prepared to pay the highest price wins the auction. This also is the case for sending and receiving announcement messages, again an act-sense measurement criterion.

The problem with this is that when running this as a large-scale agent-based social simulation, we cannot assure that every bidder had time to process the announcement by the auctioneer and therefore it might be the case that the highest bidder had no time to process the receiving of an announcement and therefore does not bid, which leads to non-repeatability in the outcome.

Formal problem

Formally speaking, let us state that there are two types of relevant events: receiving a message and sending it. Two types of messages exist: announcement and bid. Furthermore a final message called *sold* is sent to finish the auction.

We call the sending of an announcement message $sa(i, p)$, where i is the agent and p is the price for the product agent i wants to pay. The receiving is called $ra(i, j, p)$, where agent i receives the announcement that agent j wants to pay price p for the item for sale.

We call the sending of a bid message $sb(i, p)$, where i is the agent that bids price p. The receiving is called $rb(i, p)$, where the auctioneer receives the bid of agent i who wants to pay p for the item for sale.

Note that the receiving event of a message is slightly different then sending it: such an event tries to receive a message. It checks whether there are unread messages in a message box. If there is none, it still has checked whether a message exists.

In the most simple case, there are two bidders (agents 1 and 2) and an auctioneer. The auctioneer has set the starting price at 10, bidder 1 has a maximum price of 25 and bidder 2 a maximum price of 33. Both agents bid the price $p + 1$ as a response to the received announcement from the auctioneer.

We can then have the following (partial) trace of a correct execution of this example, namely:

$$t_1 = ...sa(1, 25)ra(2, 1, 25)sb(2, 26)rb(2, 26)sa(2, 26)ra(1, 2, 26)sa(2, 26)sa(2, 26)sold(2)...$$

Another trace might be:

$$t_2 = ...ra(2, 1, 20)sa(1, 20)sa(1, 20)sa(1, 20)sa(1, 20)sold(1)ra(2, 1, 20)...$$

The measurements are different for t_1 compared to t_2 because in t_1 agent 2 wins and in t_2 agent 1 wins and in t_2 agent 2 tries to receive the announcement before it is send. So clearly we have a problem of repeatability here.

Solution

Intuitively, the problem defines the conflict that exists between multiple concurrently running agents that start shifting their times of execution (i.e. the amount of deliberation cycles they have ran) due to the amount of load focused which causes different ordering. For preventing this, we must ensure that agents wait for all *act*-events to have finished before continuing to the *sense*-events. This means synchronization between the *act*-phase and the *sense*-phase. To do so formally we define an event called *sync*, which is used in the redefinition of the deliberation cycle, which now becomes:

Definition 4.13 (Act-sense synchronized deliberation cycle)**.** *Given the events sense, reason and act, which denote the different stages of the sense-reason-act*

cycle, and the synchronization event sync, we call an act-sense synchronized deliberation cycle D^s, defined as:

$$D^s = sense \rightarrow reason \rightarrow sync \rightarrow act \rightarrow sync \rightarrow D^s|$$

$$sense \rightarrow reason \rightarrow sync \rightarrow act \rightarrow sync \rightarrow STOP$$

Furthermore, for ensuring synchronization, we will redefine the multi-agent system as:

Definition 4.14 (Synchronized multi-agent system)**.** *Given a set of agents $A = \{D_1^s, D_2^s, ..., D_n^s\}$, a synchronized multi-agent system is denoted as: $MAS^s = D_1^s|\{sync\}|D_2^s|\{sync\}|...|\{sync\}|D_n^s$.*

The intuition behind this synchronized multi-agent system is that the operator $|\{sync\}|$ (see Definition 4.12) waits until both the process on the left and on the right can process the event *sync* and then processes a new instance of the event *sync*, after this the processes continue with the sense or act-phase of the deliberation cycle. We call the adaptations we made to the default system MAS (see Definition 4.1) to make it MAS^s as a function $sync_1$, meaning $MAS^s = sync_1(MAS)$.

So implementation based, we only have to check if all agents have finished their deliberation cycle before we continue to the next deliberation cycle.

Proof

We have described intuitively that this works. Now let us proof it formally using the introduced definitions. We therefore state:

Theorem 4.1. *Given a large-scale agent-based social simulation P, an act-sense measurement criterion E_P (which is the result of a function mc over P), the measurement m_{E_P} is repeatable for $sync_1(P)$, meaning $\forall t, t' \in traces(sync_1(P)) : m_{E_P}(t) \Leftrightarrow m_{E_P}(t')$.*

Proof. Suppose an arbitrary P and an arbitrary measurement criterion E_P.

Problems with repeatability occur in the ordering between events from Act (i.e. the set of all possible events in the act-phases of all agents) and the sense-events, which form the set $Sense$ (i.e. all the possible events in the sense-phase of all agents).

We call S the set of all sense-events $s \in Sense$ for which also holds that $s \in E_P$. And we call A the set of all act-events in $a \in A$ for which also holds that $a \in E_P$.

All $s \in S$ are by Definition 4.10 sense-events that notice a result that is caused by one or multiple actions which are in A. Meaning we can combine these events as causal relations, formally $C \subseteq 2^A \times 2^S$. This C consists of tuples $(x, y) \in C$ (read all elements of x are caused by all elements of y) where $x \subseteq S$ and $y \subseteq A$ and x and y are both nonempty.

All we have to prove is that there exists only one possible order for these causal relations C which we can do by the synchronization operator.

By Definition 4.13 we have that synchronization happens between the act and sense-phase in the form of a *sync*-event. By Definition 4.12 we see that for that it is required that all processes have arrived up until *sync* to continue, by Definition 4.13 this means that all agents must have finished their act-phase before continuing to the sense-phase and must have finished their sense-phase before continuing to the act-phase. Meaning for the causal relations that $\forall t \in trace(P), \forall (x, y) \in C, \forall a \in x, \forall b \in y : a \leq_t b$.

By the notion of ordering within all causal relations in C for all possible traces we can conclude that $\forall t, t' \in traces(sync_1(P)) : m_{E_P}(t) \Leftrightarrow m_{E_P}(t')$. Since we assumed this for an arbitrary P and E_P this holds for all possible large-scale agent-based social simulations and all measurement criteria on these large-scale agent-based social simulations. That is what had to be shown. □

4.3.3 Act-only synchronization

The other type of measurement criterion we identified is the act-only measurement criterion. Also for this we shall propose a synchronization technique, which we call act-only synchronization.

We therefore first introduce an example for which the problem occurs. Then describe it formally and finally solve it by synchronizing the multi-agent system.

Example

This example is about cleaning a world of bombs, two types of agents exist in this: harry and sally. Both agents have in common that they are persons and according to the characteristics of this example, a person can move from point to point in a world. This world is nothing more than a k by l grid of discrete locations that can be occupied by an agent. A move is a horizontal or vertical movement in this grid. A maximum of one move in this grid per deliberation cycle is allowed per agent.

In this environment, three types of static entities exist: bombs, traps and walls. Bombs have to be picked up and delivered at traps to remove them. An agent can only carry one bomb at a time. Traps cannot be picked up. Agents cannot pass through walls, which therefore add some extra complexity to the path planning.

The first type of agent is harry, his task is to identify bombs. For this he has the following capabilities:

- Search for bombs (i.e. randomly walking around the environment)
- Inform sally about the bombs it discovered

The second type of agent is sally, she has the following capabilities:

- Search for traps (i.e. randomly walking around the environment)
- Receive bomb information from harry

- Go to bombs, pickup the bomb and then deliver the bomb at a trap

When agents move, they 'sense' environment. Meaning they only get information about the environment which is close to them. The sensing range in this example is chosen as 4 in each size, so a sensing radius of 8. This is almost half the size of the world, which is a 20 by 20 grid in the example here. Also all bombs and traps are assumed to exist when starting the multi-agent system, they are predefined by the designer, only the agents do not know them yet (i.e. they do not exist in their belief bases at the start). When bombs are discovered by harry, he sends a message to the closest sally to inform him about this bomb she has found.

For the problem to occur, at least one harry and two sallies should exist. This is because the conflict is between the two sallies and one harry in which the two sallies both try to remove the same bomb which is identified by harry. The problem is that a case exists in which one harry removes a bomb at location (x, y) and at the same time harry scans this bomb and sends it to the other sally. Then sally-2 thinks there is a bomb at that location, however this only is the case if harry first scans the area and then sally removes the bomb at that location, the other way around the behavior is different which might lead to non-repeatability.

The measurement in this is the amount of bombs picked up per agent. This may differ if two or more types of behavior are possible that lead to different outcomes in the measurement.

Formal problem

Let us define the problem formally, for this we again use the sense-reason-act deliberation cycle (see Definition 4.2). Formally speaking here, the problem is a different ordering of actions of removing bombs (i.e. act-events). We state that attempt to remove a bomb at location (x, y) by agent sally-1 is denoted as $r(x, y, 1)$. This attempt succeeds if there is indeed a bomb at the given location and fails otherwise, but therefore it still exists within the trace. The measurement criterion in this case is an act-only measurement criterion existing of remove bomb attempts.

A possible (partial) trace is:

$$t = ...r(x, y, 1)...r(x', y', 2)r(x', y', 1)...$$

The above trace leads to the outcome of the simulation that both sally-1 and sally-2 have cleaned 1 bomb. Another possible trace is:

$$t' = ...r(x, y, 1)...r(x', y', 1)r(x', y', 2)...$$

This here leads to the outcome that sally-1 has cleaned 2 bombs and sally-2 has cleaned none. Both traces are allowed by the default implementation and the synchronization for an act-sense measurement criterion does not handle this, because it exists between different act-events.

The outcomes are clearly different and that is a violation of the ordering defined by the measurement criterion. This measurement criterion either is $r(x, y, 1)r(x', y', 2)r(x', y', 1)$ or $r(x, y, 1)r(x', y', 1)r(x', y', 2)$, but because both are allowed we have a problem here.

Solution

The solution for this is fixing the ordering of execution of the agents. One can do a very strict ordering by fixing every agent in a specific spot in the execution, but this would lead to serial execution. Therefore, we need something more flexible.

We use for this a directed acyclic graph, which means a graph $G = (N, E)$ where there do not exist a cycle $c \subseteq E$ where $c = \{(a, b), (b, d), ..., (x, a)\}$.

Intuition behind the use of a directed acyclic graph is that only if all agents that come earlier in the ordering have finished their deliberation cycles, an agent can be scheduled for execution its deliberation cycle. For this we introduce the notions of predecessor sets and child sets, which can be seen as the pre-sets and post-sets used in Petri nets but then for graphs.

Definition 4.15 (Predecessor set)**.** *Let $G = (N, E)$ be a directed acyclic graph where N is the set of nodes and E the set of edges (i.e. E consists of tuples $E \subseteq N \times N$ where an tuple $(a, b) \in E$ denotes a directed edge from node a to b), we define the predecessor set of $n \in N$ as all $a \in N$ for which it holds that if $(a, n) \in E$ then a is in the predecessor set P_n (i.e. $a \in P_n$) and for every node a that is in P also all nodes of its predecessor set are in P.*

Now that we have defined a set with all predecessor nodes, we also need to have a set with all children. Note however that this is a little bit different because for determining which agents can be scheduled only the direct children are of interest. Indirect children (i.e. children from direct children) cannot be scheduled when an agent (when it represents a node) is finished and that is what we are using this graph for. Formally:

Definition 4.16 (Children set)**.** *Let $G = (N, E)$ be a directed acyclic graph where N is the set of nodes and E the set of edges (i.e. E consists of tuples $E \subseteq N \times N$ where an tuple $(a, b) \in E$ denotes a directed edge from node a to b), we define the children set C_n of $n \in N$ as all nodes $a \in N$ for which $(n, a) \in E$ holds.*

We use this directed acyclic graph as a graph of agents (i.e. the nodes represent the agents). Such a graph represents the execution order and for this it is required that all agents have an edge that directs to the node or departs from it. We call such a graph of agents an agent ordering graph, formally defined as:

Definition 4.17 (Agent ordering graph)**.** *Let A be a set of agents and $E \subseteq A \times A$ be a set of directed edges $(a, b) \in E$ that should be read as the existence of a directed edge from agent a to agent b, we define the agent ordering graph $O = (A, E)$ where:*

- *All agents are connected by an edge (i.e.* $\forall a \in A : \exists x \in A$ *such that* $(a, x) \in E$ *or* $(x, a) \in E$*)*
- *The graph is acyclic, meaning there does not exist a cycle* $c \subseteq E$ *where* $c = \{(a, b), (b, d), ..., (y, a)\}$

The whole idea of using an acyclic-graph structure for execution comes from the work-span model by McCool et al. (2012). There the work-span model is used for defining the ordering of task execution.

It is now possible to generate an execution ordering. We use for proving the correctness of this approach a graph where every agent's execution is followed by one other agent's execution (i.e. serial execution). We call this agent ordering graph the *serial agent ordering graph* where for every agent there is at most one edge starting from that agent node and at most one edge ending at that agent node.

This way we can proof that this approach always works, because if two agents can perform their actions at the same time and these two agents are for instance sally-1 and sally-2 from the example (see Figure 4.2), we still have problems with ensuring repeatability. Here we do not (see Figure 4.3), however Figure 4.2's graph can result in a greater degree of parallelism so for efficiently it is worth mentioning.

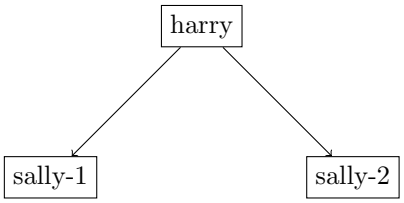


Figure 4.2: Incorrect agent ordering graph for harry-sally example

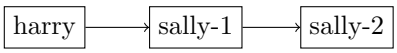


Figure 4.3: Serial agent ordering graph for harry-sally example

The scheduling of agents is done using the agent ordering graph. At the start of a new round (i.e. every agent is allowed to perform another deliberation cycle) for every agent the predecessor set is checked, whenever this one is empty the agent can be scheduled. Whenever an agent is finished, for all its children the predecessor set is checked to see if these agents have all finished, if that it the case the child is scheduled. The same agent ordering is used during the whole execution of the multi-agent system to prevent problems between different agent orderings that can cause non-repeatability.

Let us now embed this agent ordering into the multi-agent system. For this we redefine the deliberation cycle as:

Definition 4.18 (Act-only synchronized deliberation cycle)**.** *Given the events sense, reason and act, which denote the different stages of the sense-reason-act cycle, and the synchronization events start and finish for the start and finish of the deliberation cycle, we call an act-only synchronized deliberation cycle for agent i* D_i^{as}*, defined as:*

$$D_i^{as} = sense \rightarrow reason \rightarrow start_i \rightarrow act \rightarrow finish_i \rightarrow D_i^{as}|$$

$$sense \rightarrow reason \rightarrow start_i \rightarrow act \rightarrow finish_i \rightarrow STOP$$

Note from the above definition that it is very similar to the deliberation cycle for the act-sense synchronization (see Definition 4.13), however here the $start_i$ and $finish_i$ events are specific to one agent (agent i in this case). Furthermore, given an ordering that agent i's execution is followed by agent j's execution it is the case that $finish_i = start_j$. Meaning generally that $\forall(a, b) \in E : end_a = start_b$. Using this, the multi-agent system is synchronized as:

Definition 4.19 (Act-only synchronized multi-agent system)**.** *Given a set of agents* $A = \{D_1^{as}, D_2^{as}, ..., D_n^{as}\}$ *and an arbitrary agent ordering graph* $O = (N, E)$ *(where N is the set of agent identifiers 1 to n representing the agents), an act-only synchronized multi-agent system is denoted as:* $MAS^{as} = D_1^{as}||D_2^{as}||...||D_n^{as}$ *where* $\forall x, y \in N$ *: if* $(x, y) \in E$ *then* $D_x^{as}|\{start_y\}|D_y^{as}$*.*

Note from the above synchronized multi-agent system that it synchronizes on the $start_i$ synchronization events, which correspond to the *end*-events based on the agent ordering, making this definition flexible for every possible agent ordering. We call the adaption from the default multi-agent system P to an act-only synchronized multi-agent system for an agent ordering graph O as $sync_2(P, O)$.

So let us now formally proof the correctness of this synchronization technique for a serial agent ordering graph. Therefore we state:

Theorem 4.2. *Given a large-scale agent-based social simulation P, an act-only measurement criterion* E_P *(which is the result of a function mc over P), an arbitrary serial agent ordering graph* $O = (N, E)$*, the measurement* m_{E_P} *is repeatable for* $sync_2(P, O)$*, meaning* $\forall t, t' \in traces(sync_2(P, O)) : m_{E_P}(t) \Leftrightarrow m_{E_P}(t')$*.*

Proof. Suppose an arbitrary P, an arbitrary serial agent ordering graph O and an arbitrary measurement criterion E_P.

Problems with repeatability occur in the ordering between events from *Act* (i.e. the set of all possible events in the act-phases of all agents).

All $a \in Act$ are by Definition 4.11 act-events that perform some modification. Given that there are n agents this means that there are at most $(n+1)!$ possible orderings of actions for one turn (i.e. factorial of n is $\prod_{k=1}^{n} k$) given that the ordering of actions within the agents is fixed, which it is by the internal serial assumption. This is $n + 1$ instead of n because there is also a possibility that action $a \in Act$ does not occur in this deliberation cycle.

When we can proof that we can decrease the number of possible orderings to just 1 we can proof that the ordering of the act-only measurement criterion is satisfied and therefore we ensure repeatability.

To proof this, let us look at the agent ordering. We assumed an arbitrary serial agent ordering graph O. By the characteristic of the serial agent ordering graph we know that any agent node has at most one predecessor and one child (because of the serial ordering characteristic of at most one edge into and out of the node). Given that the same agent ordering is used throughout execution (which is what is stated) we can deduce the amount of possible orderings to just 1, namely the ordering of the agents in O.

This is what had to be shown to ensure the ordering of the measurement criterion and therefore ensure repeatability.

□

4.4 Consequences of synchronization

We have introduced synchronization techniques that limit the amount of possible traces. Therefore, we can also state that by doing so, we limit the behavior. This is done for ensuring repeatability, but doing so we must discuss the consequences.

It is important for the agent designer to think about this, because too much synchronization is not needed, but to little means the possible emergence of non-repeatable behavior. To help the agent designer with this task, we have a checklist with questions the designer should consider when determining the synchronization method:

- What is the influence of one agent performing more deliberation cycles then another?
- What is the influence of one agent performing a sense of a result of which the action is not performed yet by another agent?
- What is the influence of external triggers?
- What is the influence of one agent performing its actions before another agent (and vise versa)?

Not surprisingly these questions correspond to the synchronization techniques purposed. Doing so helps determining which behavior should be synchronized and prevents limiting too much possible traces.

Chapter 5

Results

In this chapter we practically show the correctness of the synchronization techniques proposed in chapter 4 so that they ensure repeatability. We have implemented as a proof of concept the examples for the different problems identified the developed synchronization techniques to show that we can ensure repeatability in large-scale agent-based social simulations.

The structure of this chapter is as follows: first we describe the examples as implemented in OO2APL (Dastani and Testerink, 2014) with DSOL (Jacobs, 2005). Then we continue with the method used in the experiments that ensures repeatability. Then for each of the classes of repeatability problems described in chapter 4 we describe the default behavior (i.e. the non-repeatability) and then we describe the implementation of the corresponding synchronization techniques and show that practically it ensures repeatability. We finish with describing some of the observed behavior in the experiments.

5.1 Examples

In this section we describe the examples implemented for showing repeatability or the lack of it. We have already described the examples from a functional point of view in chapter 4, where we described their capabilities. Now however we look at them from an implementation standpoint.

5.1.1 Auction

The first example is that of an auction. A small recap of what is already described in section 4.3.2: there are two types of agents, an auctioneer and a bidder. The auctioneer manages the auction by sending announcements about bids it has received and accepted to all participating bidders. Bidders can respond to this with a new (higher) bid or decide not to respond. These bidders do this based on the maximum price they have in their belief base and a step size.

This step size is, just like the maximum price, a predefined value in the belief base of the bidder. The values for the maximum price are samples from an uniform distribution with a minimum of 0 and a maximum of 100. The step sizes are samples from an uniform distribution with a minimum of 1 and a maximum of 5. These values are predefined for this example as part of the input. The auctioneer starts with a starting price of 5 for the item for sale (which is simple an empty instance of the class 'Item').

Note that because we choose a minimum of 0 for the maximum price distribution it can happen that there are agents within the range of $[0, 5]$ and therefore will never place a bid. This is of course dependent on the random values generated and causes the auction to vary in the amount of participating agents, since some agents will have a maximum price lower than the starting price of 5 causing them to remain idle and never bid. This variance causes the amount of active agents to vary, which causes a bigger probability of non-repeatability because the amount of scheduling differs.

Furthermore, for experimental purposes, some non-functional behavior has been implemented to store the results of the auction. This is done through an additional message called 'Sold'. This is sent by the auctioneer to the bidders when the item is sold. Receiving this message means that all agents are done. The receiving initiates an agent-death case, removing the agent from the multi-agent system and by this the simulation is finished.

5.1.2 Harry and sally

The example of harry and sally is also discussed before, to recall: two agents exist, one identifies bombs (i.e. harry) and one removes them (i.e. sally). They walk around in a grid that consists of walls, bombs and traps, where they have to clean the world of the bombs. For the functional behavior we refer to section 4.3.3.

Technically speaking there we go more in detail of the world manager. This manager keeps track of when the world is cleaned of bombs and finishes the simulation. It finishes the simulation by triggering an 'EndOfSimulation' and storing the results. Furthermore, this agent can receive external triggers which adds another bomb to the environment per external trigger received.

Another big element is the path planning and path traversal. Harry and sally share the code for this because they are both considered agents of the type person, which at the creation of the type harry gets extended with harry specific functionality like identifying bombs and for sally with her specific behavior. This way the path planning and path traversal is only implemented once, which improves maintainability. Also they both share a part of the belief base, namely a so called 'PersonContext'. This context consists of data about the agent's relation with the environment (i.e. the location), data for the path planning (i.e. the destination and path) and data for rendering the agents.

To allow interaction with the functionality of the person implementation and the harry and sally implementations, internal triggers are used. Internal triggers occur when:

- The location of the agent changes
- A new path must be planned
- The destination of the planned path is reached

The above three triggers all have their own purpose. The trigger for when the location of the agent is changed is used for sensing the environment for bombs and traps (depending on the type of agent). The trigger for planning a new path is implemented by the person and its task is to find the shortest path from the current location to the new destination. The internal trigger for the destination is used for planning a new path, harry chooses a random destination as its next destination whereas sally chooses a bomb or trap as its destination, depending on whether it carries a bomb or not.

For getting to the desired destination in a person agent has a goal called 'TraversePath'. This goal simply traverses the path calculated by the handling of the internal trigger for planning a new path, which is stored in the agent's belief base. Every deliberation cycle the goal performs the next step in the generated path. For the path planning the A^*-algorithm (Hart et al., 1968) is used and prevents agents from walking through walls, the agents cannot collide with other agents or bombs.

5.2 Method of experimentation

For the notion of repeatability, having the same initial state is an important characteristic. Within the implementation of the experiments this initial state is generated in a serial phase in which pseudo-random numbers are generated for determining the values of the belief base of the agents. The initial state generated by this pseudo-random number sequence is equal given that the seed (i.e. the initial value of the random number generator) is the same. This is the case because the state is generated in a serial phase and by this serial ordering the same values are assigned to the same attributes. Therefore we can conclude that the random state is equal given the same seed.

Important in this thesis is ensuring repeatability. This means we always want the same result. What this is depends on the measurement and is therefore specific to the different examples. We use the same measurements as described in chapter 4. In the sense of the auction we measure who is the winner. We therefore want to ensure that the winner is always the same and more importantly, it is the bidder who is prepared to pay the highest price. In the harry-sally example this is about the amount of bombs each agent has cleaned.

Every initial state is for us one case, for statistical significance about the synchronization techniques we therefore talk about at least 30 cases and take the averages. However, for some analysis we will dive deeper into one example to show specific behavior. For significance within these cases every case exists of 30 runs with the identical seed, which should result in the same or similar results.

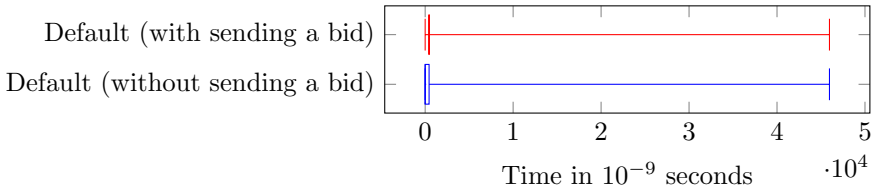


Figure 5.1: Durations of one bid by one bidder agent in the default configuration

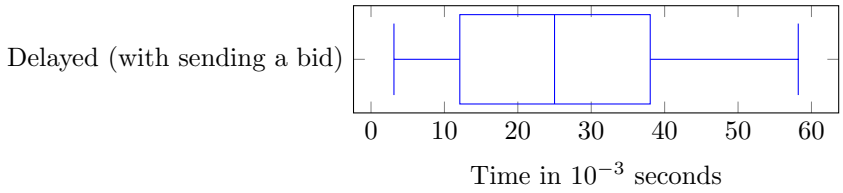


Figure 5.2: Durations of one bid by one bidder agent in the default configuration

For allowing parallelism we use OO2APL's inbuilt mechanism to use the Java Thread Pool with 4 threads, because our test machine contains a processor (Intel I5-2410M at 2.3 Ghz) with four threads. Therefore it is most logical to use all these threads.

5.2.1 Enforcing non-repeatability in the auction example

The scheduler by default can sometimes cause repeatability by coincidence. We assume that this is caused by some greedy scheduling algorithm within the Java Virtual Machine. We can however force non-repeatability by ensuring that some agents have an increased runtime per deliberation cycle, without synchronization we see that some agents in the same time perform 10 deliberation cycles where others only perform 1. To do this, we add to the bidders a random delay (where the seed used for the delay generation is the current processor time) before they respond with a bid to the auctioneer's announcement. We therefore first measure the time it takes for the agent respond.

First we analyze the current durations of a bid by one agent. For this we set up an experiment where we pick 30 random seeds by which we run 30 auctions. In total this resulted in 4403 bids which we timed. In Figure 5.1 the results of this are shown. We see that measuring the plan schemes that actually send bids takes significantly longer (t-test value of $1,01716 * 10^{-6}$) with a median of 0 compared to a median of 446 nanoseconds.

We therefore only delay the bids that actually send a message to the auctioneer. We do this by adding a delay from an uniform distribution with a minimum of 0 milliseconds and a maximum of 50 milliseconds. The results of the durations are shown in Figure 5.2. From this we can see that they are picked

from an uniform distribution because the mean is $mean \approx \frac{max-min}{2}$, the first quartile around $\frac{mean}{2}$ and the last quartile around $\frac{3*mean}{2}$.

We also choose a maximum of 50 milliseconds for another reason: from the first quartile onwards, most the samples of this distribution (i.e. 75 percent by the first quartile and upwards which is by Figure 5.2 around 12) have a degree of repeatability of 50 percent or lower (only delays 15, 20 and 40 milliseconds have a value of just 51, which we say is around 50). This degree of repeatability is the probability that the best bidder in the auction (i.e. the one with the highest best price) also wins the auction. The lower this degree, the more work for our synchronization techniques to enforce repeatability. This is shown in Figure 5.3, which shows the the probability that the best agent wins the auction compared to the delay. Note here that the line drastically decreases whenever even a small delay of 1 millisecond exists, by the implementation of the Java runtime it is not possible to stably wait for less than 1 millisecond, which is why we have not tested this. There are tricks available for waiting for less than one millisecond but (on Windows computers at least) this is not accurate: we did one test (30 trials and averaged) trying to have a delay of 1000 nanoseconds where we ended with an average delay of 43344 nanoseconds instead and another test with a delay of 100000 nanoseconds where we ended up with an average delay of 139044 nanoseconds.

Note that it is possible that by the generation of a sequence of random numbers, multiple numbers are equal. This means that it is possible for two bidders to have the same (highest) price and therefore both are considered the highest bidder. We state that this in this case both agents can win, paying exactly the maximum price, because otherwise one of the two agents would have placed a higher bid (assuming repeatability).

There is by the distribution we choose a probability of $\frac{1}{50}$ that the delay is 0, this by itself is a small amount which putting it against the rest of the samples also causes non-repeatability because it is faster than the rest. This way we increase the probability of non-repeatability even further.

5.2.2 Alternative delay forms

Up until now we have delayed the agents by waiting before the bidder sends a message. This all works for auctions, but is implementation specific. We do not think agent designers deliberately make their agents slower by these delays and implement them in real world scenarios. The only reason these delays are needed is for forcing the synchronization mechanisms to step in quite often, which leads to better testing of our synchronization mechanisms.

A generic implementation is therefore not needed, although it can be achieved. We shall propose a few ways of doing this. For this it is important to know that OO2APL creates instances of plan schemes for every agent of the same type, making them unique over every agent. This allows one to have multiple agents of the same type with different implementations when one chooses an implementation of a certain plan scheme when the agent is created.

Furthermore, one is able to add a random number of empty plans at every

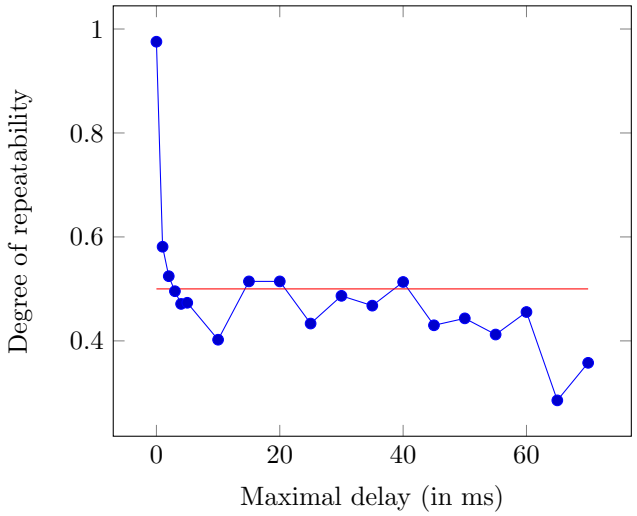


Figure 5.3: Degree of repeatability for maximal delay

deliberation cycle, which are then executed. But when leaving these plans empty (or just simply doing some arbitrary code) one is able to influence the execution time of an agent's deliberation cycle indirectly. One must be careful with this because the inbuilt empty plans are recognized by the platform and not executed by default.

5.3 Turn-based synchronization results

For ensuring synchronization for act-sense measurement criteria (except external triggers) we have developed a different variant of the agent platform. We call this turn-based synchronization.

The platform consists of a mechanism that contains a state of deliberation cycle per agent to ensure that an agent can only perform one deliberation cycle and then has to wait until the other agents have finished their deliberation cycle. This corresponds to the second synchronization point (i.e. after the act-phase) in the formal description of act-sense synchronization. The first synchronization point is already embedded into the practical implementation of OO2APL because before executing the deliberation runnable (the implementation that executes the deliberation cycle) it copies all triggers, messages and goals to separate collections so that the perception of the agent does not change during the deliberation cycle, which is the purpose of the first synchronization step.

This however does not take into account the agent sensing the environment during the sense-phase, which is not a trigger. However also the so called 'contexts' of the agent are synchronized such that during the deliberation cycle they cannot change. Contexts represent the belief base of the agent, but also

it contains the instance that represents the environment. By synchronizing this using Java's synchronized-operator the synchronization before the act-phase is ensured.

For this, agents have a state that tells in what phase of scheduling their deliberation cycle they are. The possible states are:

- Scheduled, the agent's deliberation cycle is scheduled
- Started, the agent's deliberation cycle is scheduled and has started (but not finished yet)
- Done, the agent's deliberation cycle has finished and because there are no triggers (i.e. goals or internal triggers) that exist for the next deliberation cycle, no new deliberation cycle needs to be scheduled for the time $t + 1$
- Rescheduled, the agent's deliberation cycle has finished and there are triggers that exist for the next deliberation cycle so at time $t + 1$ this agent will perform a deliberation cycle

These states can be seen as a very linear process: it goes from 'scheduled', to 'started', to 'done' or 'rescheduled'. When all agents have finished their deliberation cycle (i.e. all have a state of 'done' or 'rescheduled') all agents that have a state 'rescheduled' are scheduled for execution and the whole process starts all over again. We need the state 'rescheduled' together with 'done' because we make a distinction on which agents are finished and do not need to perform another deliberation cycle and agents that are finished and need another deliberation cycle. Furthermore we need to distinct which agents have already finished (i.e. 'done' or 'rescheduled') and still need to perform their deliberation cycle (i.e. 'scheduled'). The distinction between 'scheduled' and 'started' is necessary because we need to distinct agents that are already performing their deliberation cycle and agents that have just been scheduled when the simulation is terminated during the execution of the agent platform. Then we know which agents are currently performing a deliberation cycle and based on this we terminate their threads by waiting before they have finished the current deliberation cycle.

Let us now analyze a specific example that shows non-repeatability. For this we use a seed of 3, which is chosen because the values it generates for the belief bases of the bidders show some interesting things that result in interesting behavior. This interestingness is in the fact that there are in this case 2 bidders that have a maximum price that is very close to each other (i.e. 95 vs. 92) and a another group of bidders with a price just below it (i.e. 3 agents with a price of 85 and 86) with no highest prices in between these groups. Furthermore there is one auctioneer and 40 bidders.

From this configuration, the implementation generates 40 bidders and 1 auctioneer that together perform an auction. The belief bases of the bidders are denoted in Table 5.1. From here we see that agent 35 should win (note that the agent IDs are ascending numbers where agent 1 is the auctioneer).

ID	Maximum price	Step size	ID	Maximum price	Step size
2	34	1	**22**	57	5
3	10	2	**23**	32	5
4	28	3	**24**	70	3
5	49	5	**25**	28	1
6	59	2	**26**	24	5
7	85	3	**27**	39	2
8	77	5	**28**	57	5
9	81	2	**29**	29	4
10	76	3	**30**	7	2
11	86	5	**31**	30	1
12	11	3	**32**	26	1
13	85	1	**33**	75	2
14	78	4	**34**	71	3
15	57	5	**35**	95	3
16	46	4	**36**	50	5
17	38	1	**37**	65	1
18	22	4	**38**	46	1
19	92	3	**39**	30	2
20	61	5	**40**	3	4
21	53	3	**41**	20	1

Table 5.1: Maximum price and step size for every bidder with the $seed = 3$

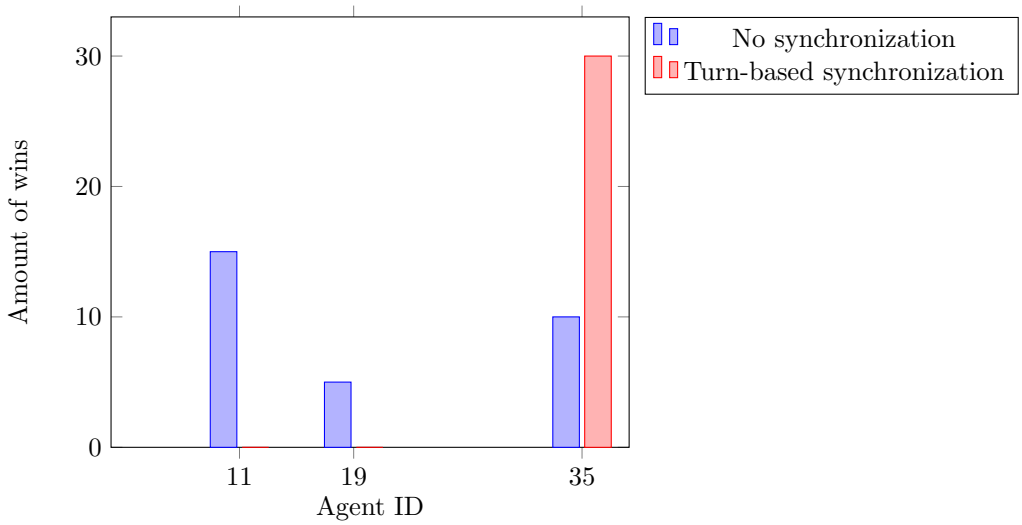


Figure 5.4: Winners of auctions based on synchronization type

	Default implementation	Turn-based synchronization
Degree of repeatability	0.5088888888888889	1

Table 5.2: Results of random seeds for turn-based synchronization in auction-example

We first show that non-repeatability in this case is a problem for the measurement of the winner by recording how many times of the 30 runs an agent has won the auction in Figure 5.4. In this figure we have left out all the bidder agents that have not won an auction.

From Figure 5.4 we can see that without proper synchronization non-repeatability is an issue in this example. By the nature of the measurement, since it is an act-sense measurement criterion, the platform only has to synchronize on a deliberation cycle basis, so turn-based synchronization suffices. We also show in this particular case (i.e. $seed = 3$) that repeatability is ensured by turn-based synchronization.

It is however more important to generally show that turn-based synchronization ensures repeatability in the auction example, not only for $seed = 3$. Therefore we generalize the experiment by running it for 30 randomly generated seeds. For these seeds we run two experiments: one with the default implementation of OO2APL and we compare this to the implementation with turn-based synchronization. The results are shown in Table 5.2.

From Table 5.2 we can see that here we show empirically that turn-based synchronization works. A degree of repeatability of 1 means that the value of the measurement is always the same, the measurement taken is the winner of

the auction. Note here that we talk about deterministic repeatability, since it must be exactly the same value.

Also we see that the degree of repeatability of the default implementation is around 0.5, exactly what we were aiming for when we tuned the delay up to 50 milliseconds. This way we show that the repeatability is caused by our implementation and it is very unlikely that it is this occurs by chance, since around 50 percent of the cases, no repeatability is assured, the probability that this occurs by chance is around 0.5089^{900} (900 because of 30×30 auctions), which is extremely small.

5.3.1 Price distributions

Another measurement might be the price that the highest bidder pays for the item for sale. When ensuring turn-based synchronization on the auction we ensure repeatability for the highest bidder, therefore we ensure that the highest bidder is always the same bidder (assuming there is only one bidder that is prepared to pay the specified highest price). We also talk here about repeatability, but this time about probabilistic repeatability. For this we analyze the output distribution, which is considered always the same by the assumption that we talk about the same winner and the same maximum price it is prepared to pay.

To properly analyze such an output distribution it makes no sense to talk about averages over 30 different seeds (i.e. input states) because we would then analyze an average output distribution of 30 different agents, therefore we will pick one seed. In our experiment for the price distribution, we choose $seed = 3$ (see Table 5.1) as the input and a delay of 50 milliseconds. We run 30 times 30 auctions of which the average resulting price distributions are shown in Figure 5.5.

When looking at the resulting distribution in Figure 5.5 we can see that although we ensure repeatability it is not deterministic repeatability for this measurement, because the value can change. There are four values agent 35 can bid ranging from 92 to 95. We note that the two most likely values are 92 and 95, this is because 92 is the maximum value of the second highest bidder and therefore it will not bid any higher and furthermore, the step size of agent 35 (i.e. the highest bidder) is 3, therefore 92 + 3 is the most likely outcome.

It is much less likely that circumstances occur in which the prices 93 and 94 are reached, because they include multiple agents that behave in similar ways (i.e. same delays) and only this coordination leads to this behavior. We can see from the initial state in Table 5.1 that the chance of reaching 94 is higher because both highest bidders have a step size of 3 and that can be caused by starting from a bid of 85 and then adding up 3 all over again, where at a bid of 91 the second highest agent should be slower to bid than the highest agent, resulting in a final bid of 94. There are two agents with a maximum price of 85 which can result in a final bid of 94 by adding 3 to the maximum bid three times. To have a resulting price of 93, both the highest bidder and second highest bidder should be slower to bid than agent 13 bidding 84 for instance and then continue to add 3 to the bid and then at a price of 90 the second

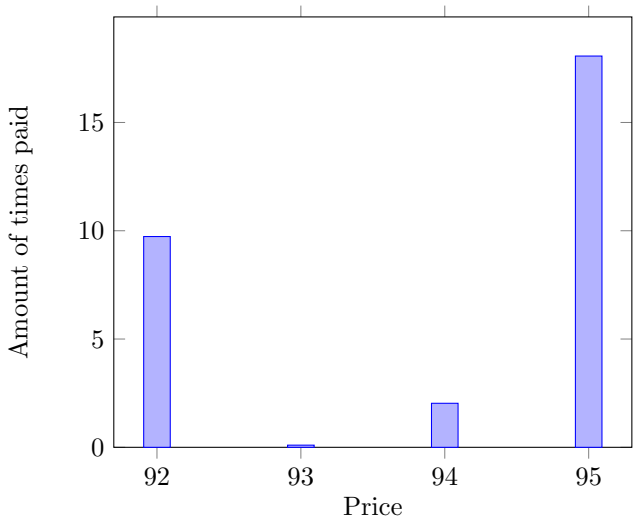


Figure 5.5: Repeatable price distribution for agent 35 in auction example (with $seed = 3$)

highest bidder should bid later than the highest bidder, resulting in a price of 93. We see here that the amount of requirements is higher and therefore it is much less likely to occur.

5.4 Agent ordering synchronization

Agent ordering synchronization implements the agent partial ordering, theoretically described in section 4.3.3. This type of synchronization is an extension to turn-based synchronization (see section 5.3).

The agent ordering is defined during the creation of the multi-agent system. In the implementation the so called preconditions of the agent (i.e. the agents that had to finish before this agent can perform its deliberation cycle) are stored. Whenever an agent reaches the finished-state from the turn-based synchronization, the platform checks which agent's preconditions are met, if all preconditions are met it will schedule the agent to execute its deliberation cycle.

From an implementation standpoint we use every agent's execution state and create a graph of it. This corresponds to the predecessor set and child set defined in section 4.3.3. Every element in the graph is considered a node we call 'OrderingNode'. An ordering node consists of:

- The agent's identifier
- The preconditions (i.e. nodes that come directly before in the agent ordering graph and have a parent-child relation)

- The postconditions (the counterpart of the preconditions, for every precondition in the child node a postcondition exists in the parent-node)
- Whether the agent has finished its deliberation cycle at that step
- Whether the agent is sleeping
- Whether the agent is scheduled for performing its deliberation cycle

On this, a few operations can be performed, namely:

- Checking whether the agent can be scheduled (based on the preconditions)
- Which agents can be scheduled after this has finished

This whole ordering is stored in a class called 'AgentOrdering'. It stores all agents based on their identifier in a hash table such that finding them based on the agent identifier is fast. It has four tasks:

- Update the ordering
- Return the agents that can start a new round
- Return the agents that can be scheduled after an agent is finished
- Reset the properties of the nodes in the ordering such that a new round can start

This agent ordering reacts to two types of events within the agent platform. The first case is when a new round is started, then the properties are reset (i.e. whether an agent has finished a deliberation cycle) and the agents that can start a new round are returned by the agent ordering. The second case is when an agent has finished a deliberation cycle and it checks whether the round is finished and executes the first case or otherwise it checks which agent can be started now that could not start earlier.

The above two cases form three of the four tasks of the agent ordering class. The other case is about updating the order and is normally done during the creation of the multi-agent system. By default all agents can be executed at the same time but the designer can for each agent identifier give a list of preconditions (i.e. agents that have to be finished) for the specified agent. When the agent ordering is manually overridden, the designer specifies the preconditions and the postconditions are automatically updated.

In all experiments the ordering is updated before the start and not changed during the experimentation. We use for this 30 randomly picked seeds for which we run 30 experiments. We state that a result is repeated in the sense of deterministic repeatability: every agent delivers the same amount of bombs to the traps. The degree of repeatability is again the probability that the result is repeated, meaning the chance that the result is equal to that of the most frequent result (i.e. result with the highest probability of occurring). When repeatability is ensured, this probability must of course be equal to 1.

	Default implementation	Agent ordering synchronization
Degree of repeatability	0.5953703703703703	1

Table 5.3: Results of random seeds for agent ordering synchronization in harry and sally example

Also we have to set the agent ordering. The ordering we use is simply the following: first all harry agents execute in order, then all sally agents and then the world manager. The results of this experiment is shown in Table 5.3. Here we clearly see that agent ordering synchronization ensures repeatability for the given measurement, whereas the default implementation does not.

5.5 Planned external triggers

External triggers are a special type of act-sense measurement criterion because the sending part is outside the multi-agent system. We built a mechanism that plans the arrival of external triggers repeatable.

Different than formally we also have to deal with sending external triggers. For this we need a scheduler, something that schedules the external trigger in such a way that it sends the external trigger at the same time during different runs and therefore ensures repeatable delivery of these external triggers. We call this the planning of external triggers.

For time based planning it is important to define a notion of time in the OO2APL platform. However, when we define it independent of the platform, like in the case of seconds or minutes for instance, it can happen that inconsistency occurs due to the fact that in one run the agent performs n tasks whereas in the other run it performs m tasks (assuming $n \neq m$). Deliveries of external triggers using the real system time do not solve the problem of non-repeatability. Therefore, we base time on the amount of deliberation cycles run by one agent and whenever all agents are at the same time (i.e. the round has finished), we deliver an external trigger.

We implement a different method for sending external triggers to the agents to include a time at which the trigger is fired. These are all stored in a queue where the platform checks at every turn (i.e. every agent has finished one deliberation cycle) if the agent needs to receive external triggers. Because it works on a turn basis on the platform, both turn-based synchronization and agent ordering synchronization can be equipped with handling planned external triggers.

From the example of harry and sally it means that we schedule n external triggers which are sent to the world manager. This world manager then adds a bomb at a randomly picked location in the world. This amount of external triggers is picked from an uniform distribution with a minimum of 1 and a maximum of 20. In this experiment we also run 30 times 30 experiments of the

	Default implementation + external triggers	Agent ordering synchronization + external triggers synchronization
Degree of repeatability	0.5005555555555555	1

Table 5.4: Results of random seeds for agent ordering synchronization with external trigger scheduling in harry and sally example

example of harry and sally but this time with the implementation of random bombs by the world manager through external triggers.

We have however a problem because by default adding external triggers with some delay is not possible. Because when we schedule all external triggers directly (i.e. the default behavior) it would always be the same and therefore is repeatable. We therefore create an additional thread which sends these external triggers bases it on an interval to the agent platform.

In the experiment we compare the default implementation of the platform with the external trigger synchronization and agent ordering synchronization for the harry-sally example. The default implementation sends external triggers on an unmanaged time interval basis whereas the external trigger synchronization based on the time which is sent together with the external trigger when it was scheduled. The results of these experiments are shown in Table 5.4.

From the results of the external trigger scheduling we see that external trigger scheduling ensures deterministic repeatability on the measurement of the amount of bombs picked up by every agent. What we also see is that the default implementation performs worse when adding external triggers (by comparing it to the default implementation in Table 5.3, which is not surprising since we increase the amount of ordering errors that can occur by adding bombs in a way that is for the multi-agent system itself unpredictable and most of the time different.

5.6 Summary

In this chapter we have shown that our implementations for ensuring repeatability works on two examples: the auction and the harry-sally example. For this we have described the way these synchronization techniques are implemented and we have discussed probabilistic and deterministic repeatability.

From all these experiments we have concluded that we can ensure repeatability in large-scale agent-based social simulations practically.

Chapter 6

Conclusion & Future work

In this thesis we asked the question 'how to ensure repeatability in large-scale agent-based social simulations?'. In this chapter we answer this research question and discuss possible future work.

6.1 Research question

We divided the research into three sub-questions. Which we will answer one by one to gradually answer the main question.

How can we ensure repeatability in complex reasoning agents such as BDI-agents? We identified that problems of repeatability occurred when the ordering of actions differed. This had influence on interactions between different agents. Within these agents we assumed that agents are internally serial, which ensured the same ordering. However, complex reasoning agents can receive external triggers and in this sense it is important for an agent on its own to handle this in a repeatable way. We proposed and implemented a synchronization technique called external trigger synchronization that sends triggers with a timestamp, which is the amount of deliberation cycles the agent has executed since the creation of the agent. This way an agent always processes external triggers in the same order with other actions within the agent and therefore repeatability is ensured within complex reasoning agents.

How to ensure repeatability in large-scale multi-agent systems where interactive agents run concurrently? For this we had a look at the workings of agents, which use a deliberation cycle, and identified two types of problems: problems in the ordering of sense and act events of the deliberation cycle of agents and in the ordering of act events only. For the first problem we synchronize before and after the act-phase of the deliberation cycle and for the second type of problem we define an order of execution for the agent's act-phase of the deliberation cycle. We show that these synchronization techniques work.

How to build large-scale agent-based social simulations that ensure repeatability? For this we implemented the two synchronization techniques

described earlier and built a system for sending external triggers repeatably. We did some testing over these techniques and showed that they ensure repeatability for two examples identified to have problems with repeatability for the three ordering mistakes that can be made. These examples, an auction and an example of two agents that try to clean a world of bombs, show that repeatability is not by default ensured in OO2APL, but with the implemented synchronization techniques they do ensure repeatability.

Overall we can now answer the main question: how to ensure repeatability in large-scale agent-based social simulations. For this we purpose three synchronization techniques that ensure this. When implemented we have shown that they do ensure repeatability whereas they otherwise would not.

6.2 Future work

During the research we have discovered many different research alternatives from which we want to discuss a few. These range from ensuring a different notion to different solutions we will discuss further.

First of all there is the related notion of reproducibility in large-scale agent-based social simulations. This notion is for differences between simulation implementations interesting and can be used to compare different multi-agent systems in a formal way. Think for instance about different implementations for delivering messages to agents and how this influences repeatability in large-scale agent-based social simulations. This does apply to message deliveries but nearly every decision made by the platform that executes the agents and thereby possibly affects repeatability.

Another possibility is that of investigating the possibility of some sort of Time Warp mechanism (Jefferson, 1985) for repeatability. Therefore one has to identify a problem with repeatability and then try to solve it through a rollback. Doing so makes the use of conservative synchronization techniques like the once we implemented unnecessary.

Bibliography

Arifin, S. M. N. and Madey, G. R. (2015). *Verification, Validation, and Replication Methods for Agent-Based Modeling and Simulation: Lessons Learned the Hard Way!*, pages 217–242. Springer International Publishing, Cham.

Balmer, M., Meister, K., and Nagel, K. (2008). *Agent-based simulation of travel demand: Structure and computational performance of MATSim-T*. ETH, Eidgenössische Technische Hochschule Zürich, IVT Institut für Verkehrsplanung und Transportsysteme.

Bartlett, J. and Frost, C. (2008). Reliability, repeatability and reproducibility: analysis of measurement errors in continuous variables. *Ultrasound in Obstetrics and Gynecology*, 31(4):466 – 475.

Bellifemine, F., Caire, G., Poggi, A., and Rimassa, G. (2007). Jade: A software framework for developing multi-agent applications. lessons learned. *Information and Software Technology*, 50(12):10 – 21.

Bergstra, J. A. and Klop, J. W. (1984). Process algebra for synchronous communication. *Information and control*, 60(1-3):109–137.

Best, E. and Fernandez, C. (1988). *Nonsequential Processes - A Petri Net View*. Springer-Verlag Berlin Heidelberg.

Calvin, J., Dickens, A., Gaines, B., Metzger, P., Miller, D., and Owen, D. (1993). The simnet virtual world architecture. In *Virtual Reality Annual International Symposium, 1993., 1993 IEEE*, pages 450–455. IEEE.

Cetin, N., Burri, A., and Nagel, K. (2003). A large-scale agent-based traffic microsimulation based on queue model. In *IN PROCEEDINGS OF SWISS TRANSPORT RESEARCH CONFERENCE (STRC), MONTE VERITA, CH*, pages 3–4272.

Chandy, K. M. and Misra, J. (1981). Asynchronous distributed simulation via a sequence of parallel computations. *Communications of the ACM*, 24(4):198–206.

Copeland, J. (1993). *Artificial Intelligence: A Philosophical Introduction*. Wiley-Blackwell.

Dastani, M. (2008). 2apl: a practical agent programming language. *Autonomous Agents and Multi-Agent Systems*, 16(3):214–248.

Dastani, M. and Testerink, B. (2014). *From Multi-Agent Programming to Object Oriented Design Patterns*, pages 204–226. Springer International Publishing, Cham.

Dastani, M., van Birna Riemsdijk, M., and Meyer, J.-J. C. (2005). Programming multi-agent systems in 3apl. In *Multi-agent programming*, pages 39–67. Springer.

Davidsson, P. (2002). Agent based social simulation: A computer science view. *Journal of Artificial Societies and Social Simulation*, 5(1).

Duffy, G. (1992). Concurrent interstate conflict simulations: testing the effects of the serial assumption. *Mathematical and Computer Modelling*, 16(8-9):241–270.

Dushnik, B. and Miller, E. W. (1941). Partially ordered sets. *American Journal of Mathematics*, 63(3):600–610.

Filippi, J.-B., Morandini, F., Balbi, J. H., and Hill, D. R. (2010). Discrete event front-tracking simulation of a physical fire-spread model. *Simulation*, 86(10):629–646.

Finin, T., Fritzson, R., McKay, D., and McEntire, R. (1994). Kqml as an agent communication language. In *Proceedings of the third international conference on Information and knowledge management*, pages 456–463. ACM.

FIPA (1999). Specification part 2 - agent communication language.

Fokkink, W. (2007). *Introduction to Process Algebra*. Springer-Verlag, 2 edition.

Fokkink, W. and Zantema, H. (1994). Basic process algebra with iteration: Completeness of its equational axioms. *The Computer Journal*, 37(4):259–267.

Fomel, S. and Hennenfent, G. (2007). Reproducible computational experiments using scons. In *Acoustics, Speech and Signal Processing, 2007. ICASSP 2007. IEEE International Conference on*, volume 4, pages IV–1257. IEEE.

Fujimoto, R. M. (1990). Parallel discrete event simulation. *Commun. ACM*, 33(10):30–53.

Fujimoto, R. M. (1995). Parallel and distributed simulation. In C. Alexopoulos, K. Kang, W. L. and D.Goldsman, editors, *Proceedings of the 1995 Winter Simulation Conference*, pages 118–125, Piscataway, New Jersey. Institute of Electrical and Electronics Engineers.

Fujimoto, R. M. (1997). Zero lookahead and repeatability in the high level architecture. In *Proceedings of the 1997 Spring Simulation Interoperability Workshop*, pages 3–7.

Fujimoto, R. M. (2000). *Parallel and distributed simulation systems*, volume 300. Wiley New York.

Fujimoto, R. M. (2001). Parallel simulation: parallel and distributed simulation systems. In *Proceedings of the 33nd conference on Winter simulation*, pages 147–157. IEEE Computer Society.

Glowacka, K. J., Henry, R. M., and May, J. H. (2009). A hybrid data mining/simulation approach for modelling outpatient no-shows in clinic scheduling. *Journal of the Operational Research Society*, 60(8):1056–1068.

Groote, J. F. and Huttel, H. (1994). Undecidable equivalences for basic process algebra. *Information and Computation*, 115(2):354–371.

Haddock, J. (1987). An expert system framework based on a simulation generator. *Simulation*, 48(2):45–53.

Hall, R. S. (2002). Lecture 13: Introduction to csp (communicating sequential processes). `http://www.inf.fu-berlin.de/lehre/WS01/19530-V/lectures/Lecture13.pdf`.

Hart, P. E., Nilsson, N. J., and Raphael, B. (1968). A formal basis for the heuristic determination of minimum cost paths. *IEEE transactions on Systems Science and Cybernetics*, 4(2):100–107.

High-Technologies Corporation (2017). CD-SEM - What is a Critical Dimension SEM? `http://www.hitachi-hightech.com/global/products/device/semiconductor/cd-sem.html`. Accessed: 2017-03-10.

Hoare, C. (2015). *Communicating Sequential Processes*.

Hoare, C. A. R. (1978). Communicating sequential processes. In *The origin of concurrent programming*, pages 413–443. Springer.

Jacobs, P. H. (2005). *The DSOL simulation suite*. PhD thesis, TU Delft, Delft University of Technology.

Jefferson, D. R. (1985). Virtual time. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 7(3):404–425.

Kabanza, F., Barbeau, M., and St-Denis, R. (1997). Planning control rules for reactive agents. *Artificial Intelligence*, 95(1):67 – 113.

Kirkpatrick, S., Gelatt, C. D., and Vecchi, M. P. (1983). Optimization by simulated annealing. *science*, 220(4598):671–680.

Lane, M. S., Mansour, A. H., and Harpell, J. L. (1993). Operations research techniques: A longitudinal update 1973–1988. *Interfaces*, 23(2):63–68.

Law, A. M. and Kelton, D. M. (2014). *Simulation modeling and analysis*. McGraw-Hill, 5th edition.

Law, D. R. (1998). Scalable means more than more: a unifying definition of simulation scalability. In *Proceedings of the 30th conference on Winter simulation*, pages 781–788. IEEE Computer Society Press.

Lin, L. I.-K. (1989). A concordance correlation coefficient to evaluate reproducibility. *Biometrics*, 45(1):255–268.

Lu, M. (2003). Simplified discrete-event simulation approach for construction simulation. *Journal of Construction Engineering and Management*, 129(5):537–546.

MacNeille, H. M. (1937). Partially ordered sets. *Transactions of the American Mathematical Society*, 42(3):416–460.

McCool, M. D., Robison, A. D., and Reinders, J. (2012). *Structured parallel programming: patterns for efficient computation*. Elsevier.

McGregor, I. (2002). The relationship between simulation and emulation. In *Simulation Conference, 2002. Proceedings of the Winter*, volume 2, pages 1683–1688. IEEE.

Minitab Inc (2016). Repeatability and reproducibility in measurement systems. `http://support.minitab.com/en-us/minitab/17/topic-library/quality-tools/measurement-system-analysis/gage-r-r-analyses/repeatability-and-reproducibility/`. Accessed: 2017-03-08.

Mitchell, E. E. and Gauthier, J. S. (1976). Advanced continuous simulation language (acsl). *Simulation*, 26(3):72–78.

Nielsen, M. and Thiagarajan, P. S. (1984). *Degrees of non-determinism and concurrency: A Petri net view*, pages 89–117. Springer Berlin Heidelberg, Berlin, Heidelberg.

Özgün, O. and Barlas, Y. (2009). Discrete vs. continuous simulation: When does it matter. In *Proceedings of the 27th international conference of the system dynamics society*, volume 6, pages 1–22.

Petri, C. A. (1962). Communication with automata.

Pollock, J. L. (1999). *Planning Agents*, volume 14, pages 53–79. Kluwer Academic Publishers.

Rao, A. and Wooldridge, M. (1999). *Foundations of Rational Agency*, volume 14, pages 1–10. Kluwer Academic Publishers.

Rao, A. S. and Georgeff, M. P. (1991). Modeling rational agents within a bdi-architecture. pages 473–484.

Riley, P. F. and Riley, G. F. (2003). Next generation modeling iii - agents: Spades — a distributed agent simulation environment with software-in-the-loop execution. In *Proceedings of the 35th Conference on Winter Simulation: Driving Innovation*, WSC '03, pages 817–825. Winter Simulation Conference.

Santer, B., Wigley, T., and Taylor, K. (2011). The reproducibility of observational estimates of surface and atmospheric temperature change. *Science*, 334(6060):1232–1233.

Shoham, Y. (1993). Agent-oriented programming. *Artificial Intelligence*, 60(1):51 – 92.

Stimpson, J. L. and Goodrich, M. A. (2003). Learning to cooperate in a social dilemma a satisficing approach to bargaining. In *Proceedings of the 20th International Conference on Machine Learning*, pages 728 – 735.

Taylor, B. N. and Kuyatt, C. E. (1994). Guidelines for evaluating and expressing the uncertainty of nist measurement results. Accessed: 2017-03-08.

Trotter, W. T. (1995). Partially ordered sets. *Handbook of combinatorics*, 1:433–480.

Trotter, W. T. (2001). *Combinatorics and partially ordered sets: Dimension theory*, volume 6. JHU Press.

Trybulec, W. A. (2000). Partially ordered sets. *Def*, 2:2D.

Tumer, K. and Agogino, A. (2007). Distributed agent-based air traffic flow management. In *Proceedings of the 6th international joint conference on Autonomous agents and multiagent systems*, page 255. ACM.

Van der Hoek, W., van Linder, B., and Meyer, J.-J. C. (1999). *An Integrated Modal Approach to Rational Agents*, pages 133–167. Springer Netherlands, Dordrecht.

Van der Hoek, W. and Wooldridge, M. (2008). Multi-agent systems. *Foundations of Artificial Intelligence*, 3:887–928.

van Leeuwen, J. (1990). *Handbook of theoretical computer science*, volume 1. Elsevier.

Varga, A. (2001). Discrete event simulation system. In *Proc. of the European Simulation Multiconference (ESM'2001)*.

Verbraeck, A. (2017). DSOL core project. `http://www.simulation.tudelft.nl/dsol/3.0/dsol-core/index.html`. Accessed: 2017-05-03.

Villarrubia, J. S., Vladar, A. E., and Postek, M. T. (2003). Simulation study of repeatability and bias in the cd-sem. In *Microlithography 2003*, pages 138–149. International Society for Optics and Photonics.

Weiss, W. (2011). Development and use of the dynamic security model for airports. *Journal of Airport Management*, 5(3):245–254.

Wooldridge, M. (2009). *An introduction to multiagent systems*. John Wiley & Sons.