Preprints.org

Article

Not peer-reviewed version

# Parallel Simulation Using Reactive Streams: A Graph-Based Approach for Dynamic Modeling and Optimization

Oleksii Sirotkin *, Arsentii Prymushko, Ivan Puchko, Hryhoriy Kravtsov, Mykola Yaroshynskyi, Volodymyr Artemchuk

Posted Date: 21 April 2025

doi: 10.20944/preprints202504.1608.v1

Keywords: parallel simulation; reactive streams; logical processors; transition functions; state space; synchronization protocol

Preprints.org is a free multidisciplinary platform providing preprint service that is dedicated to making early versions of research outputs permanently available and citable. Preprints posted at Preprints.org appear in Web of Science, Crossref, Google Scholar, Scilit, Europe PMC.

Copyright: This open access article is published under a Creative Commons CC BY 4.0 license, which permit the free download, distribution, and reuse, provided that the author and preprint are cited in any reuse.

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 21 April 2025 doi:10.20944/preprints202504.1608.v1

Disclaimer/Publisher's Note: The statements, opinions, and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions, or products referred to in the content.

*Article*

# Parallel Simulation Using Reactive Streams: A Graph-Based Approach for Dynamic Modeling and Optimization

**Oleksii Sirotkin [1,*], Arsentii Prymushko [1], Ivan Puchko [1], Hryhoriy Kravtsov [1], Mykola Yaroshynskyi [1] and Volodymyr Artemchuk [1,2,3,4]**

[1] Department of Mathematical and Computer Modeling, G.E. Pukhov Institute for Modelling in Energy Engineering of the NAS of Ukraine, 15 General Naumov Str., 03164 Kyiv, Ukraine
[2] Department of Environmental Protection Technologies and Radiation Safety, Center for Information-Analytical and Technical Support of Nuclear Power Facilities Monitoring of the NAS of Ukraine, 34a Palladin Ave., 03142 Kyiv, Ukraine
[3] Department of Information Systems in Economics, Kyiv National Economic University Named after Vadym Hetman, 54/1 Peremohy Ave., 03057 Kyiv, Ukraine
[4] Department of Intellectual Cybernetic Systems, State Non-Profit Enterprise State University "Kyiv Aviation Institute", 1 Liubomyra Huzara Ave., 03058 Kyiv, Ukraine
* * Correspondence: cabemailbox@gmail.com

**Abstract:** Modern computational models tend to become more and more complex, especially in fields like computational biology, physical modelling, social simulation and others. With the increasing complexity of simulations, modern computational architectures demand efficient parallel execution strategies. This paper proposes a novel approach leveraging the reactive streams paradigm as a general-purpose synchronization protocol for parallel simulation. We introduce a method to construct simulation graphs from predefined transition functions, ensuring modularity and reusability. Additionally, we outline strategies for graph optimization and interactive simulation through push and pull patterns. The resulting computational graph, implemented using reactive streams, offers a scalable framework for parallel computation. Through theoretical analysis and practical implementation, we demonstrate the feasibility of this approach, highlighting its advantages over traditional parallel simulation methods. Finally, we discuss future challenges, including automatic graph construction, fault tolerance, and optimization strategies, as key areas for further research.

**Keywords:** parallel simulation; reactive streams; logical processors; transition functions; state space; synchronization protocol

## 1. Introduction

As simulations become increasingly complex, more and more computational resources are required to execute them. Computing power continues to grow per Moore's law, but this growth shifts to the horizontal plane—i.e., it happens due to an increase in the number of parallel processors and their cores. Thus, there is a need to develop parallel-simulation algorithms capable of utilizing the computing resources of multiple CPUs.

Today several approaches exist for parallelizing simulations. In particular, we can consider the Time Warp algorithm,[1] described in detail in.[2] This algorithm has been studied for many years and has several implementations in the code.[3–5] However, Time Warp uses its own synchronization protocol, which is complex and low-level.[6] The RxHLA software framework (based on the reactive adaptation of IEEE 1516 standard)[7] is similar to Time Warp in terms of complexity and low-levelness. Another approach, based on the CQRS + ES architecture, is described

© 2025 by the author(s). Distributed under a Creative Commons CC BY license.

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 21 April 2025 doi:10.20944/preprints202504.1608.v1



in. [8] However, the authors of that work concentrate more on the practical aspects of implementation without much theoretical background. The HPC simulation platform [9] is also a more practical implementation of parallel simulation; it is based on actors and the AKKA library, constituting a more conservative approach than reactive streams.

The key concept of this paper is to use a general-purpose synchronization protocol to parallelize simulations: namely, the reactive-streams protocol [10–12], particularly the version that is implemented in the AKKA library. [13–16]. Thus, on the one hand, we have a classical mathematical model. On the other hand, we have a general-purpose synchronization protocol. The goal of this work is to unite them.

The rest of this manuscript is organized as follows:

- Section 2 explains the basic modeling concepts and entities that we will use in this paper.
- Section 3 extends basic modeling to be represented in the form of a transition graph and shows how a simulation can be performed on this graph.
- Section 4 shows how the transition graph can be implemented with reactive streams and how simulation can be executed.

## 2. Substates

Before we start developing a parallel-simulation algorithm with reactive streams, we define substates concepts and some objects for later use in this paper. Before reading this section, we suggest you check **Appendix A,** which describes notation and **Appendix B** which give common basic definition used in this article. Also, in the **Section 5** and **Appendix F** you can find real word examples which illustrate the described approach.

### *2.1. Substates $\mathfrak{S}^{\mathfrak{K}}{}_{q}$ as a Decomposition of the State $\mathfrak{V}$*

Each state $\mathfrak{V}$ can be represented as a set of **substates**, each of which contains only a part of the values of $\mathfrak{v}_i \in \mathfrak{V}$. There must be a way to determine which of the substates belongs to a certain $\mathfrak{V}$. One option to achieve this is to use a unique **key** to mark all substates belonging to a certain $\mathfrak{V}$.

---

***Definition**: Let us define a substate $\mathfrak{S}^{\mathfrak{K}}{}_{q}$ where $\mathfrak{S} \subseteq \mathfrak{V}$ is part or all of the set of values $\mathfrak{v}_i \in \mathfrak{V}$, $\mathfrak{K}$ is some key unique to the state $\mathfrak{V} \in \mathbb{V}^n$, and $q \in \mathbb{N}$ is the index of the substate with the same key, $\mathfrak{K}$.*

---

One or more $\mathfrak{v}_i \in \mathfrak{V}$ values can be used as key $\mathfrak{K}$. In this case, it makes no sense to include them in any of the substates, since they will be presented in the key.

For some state $\mathfrak{V}$, we have the set of substates $\{\mathfrak{S}^{\mathfrak{K}}{}_{q}\}$ with the same key $\mathfrak{K}$. We will denote this set by a bold $\mathfrak{S}^{\mathfrak{K}}$. With such representation of the state $\mathfrak{V}$, it is necessary to ensure that all $\mathfrak{S}^{\mathfrak{K}}{}_{q}$ marked by the same $\mathfrak{K}$ are not contradictory. The pair of $\mathfrak{S}^{\mathfrak{K}}{}_{q}$ with the same key $\mathfrak{K}$ can be contradictory if one or more values $\mathfrak{v}_i \in \mathfrak{S}, \mathfrak{V}$ differ under the same index.

---

***Definition**: Let us define the set of substates*

$$\mathfrak{S}^{\mathfrak{K}} := \left\{ \mathfrak{S}^{\mathfrak{K}}{}_{q}, \mathfrak{S}^{\mathfrak{K}'}{}_{q'} \mid \mathfrak{K} = \mathfrak{K}' \ \wedge q \neq q'' \right\}$$

*where for all $\mathfrak{S}^{\mathfrak{K}}{}_{q} \in \mathfrak{S}^{\mathfrak{K}}$, the consistency criterion*

$$\nexists\, \mathfrak{S}^{\mathfrak{K}}{}_{i}, \mathfrak{S}^{\mathfrak{K}}{}_{j} \in \mathfrak{S}^{\mathfrak{K}} \left( \exists\, \mathfrak{v}_l \in \mathfrak{S}^{\mathfrak{K}}{}_{i} \neq \mathfrak{v}_l \in \mathfrak{S}^{\mathfrak{K}}{}_{j} \ \forall\, l \right)$$

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 21 April 2025 doi:10.20944/preprints202504.1608.v1



*is true.*

---

In this paper, we will talk about arbitrary **sets of substates** $\mathfrak{S}^{\mathfrak{K}}{}_q$, the only requirement for which is to meet the consistency criterion.

---

***Definition**: Let us define an arbitrary set of substates*

$$\mathfrak{S}^K \coloneqq \left\{ \mathfrak{S}^{\mathfrak{K}} \cup \mathfrak{S}^{\mathfrak{K}'} \mid \mathfrak{K} \neq \mathfrak{K}' \right\}$$

*as the union of the sets* $\mathfrak{S}^{\mathfrak{K}}$ *with different* $\mathfrak{K}$.

---

Notice that these definitions do not require the presence in the set $\mathfrak{S}^{\mathfrak{K}}$ (and consequently in $\mathfrak{S}^K$) of a sufficient number of substates $\mathfrak{S}^{\mathfrak{K}}{}_q$ to cover all values $\mathfrak{v}_i \in \mathfrak{V}$.

Let us also note that, by definition, the set $\mathfrak{S}^K$ can contain more than one substate $\mathfrak{S}^{\mathfrak{K}}{}_q$ with the same key $\mathfrak{K}$. However, on an arbitrary set $\mathfrak{S}^K$ that contains duplicate keys $\mathfrak{K}$, we can construct a set $\mathfrak{S}^K$ that does not contain them. For this, we need to combine all substates with the same key into one substate:

$$\mathfrak{S}^K{}_d \xRightarrow{\forall \mathfrak{S}^{\mathfrak{K}} \subseteq \mathfrak{S}^K{}_d \left( \mathfrak{S}^{\mathfrak{K}}{}_1 \in \mathfrak{S}^K{}_u = \bigcup_{\mathfrak{K}} \mathfrak{S}^{\mathfrak{K}} \right)} \mathfrak{S}^K{}_u$$

where $\mathfrak{S}^K{}_d$ is a set with duplicate keys and $\mathfrak{S}^K{}_u$ is a set with unique $\mathfrak{K}$. Thus, we can say that an arbitrary set of substates $\mathfrak{S}^K$ can be considered as a key-value structure or as the surjective function

$$f : \{\mathfrak{K}\} \to \{\mathfrak{S}\}$$

As follows from the definition, the set $\mathfrak{S}^K$ can only be constructed from a set of states $\mathfrak{V} \subseteq \mathbb{V}^n$ in which a unique key $\mathfrak{K}$ can be associated with each state $\mathfrak{V} \in \mathfrak{V}$. Otherwise, this will lead to the appearance of substates $\mathfrak{S}^{\mathfrak{K}}{}_q$ in conflict.

The inverse transformation, i.e., the construction of $\mathfrak{V} \subseteq \mathbb{V}^n$ from an arbitrary $\mathfrak{S}^K$

$$\mathfrak{S}^K \xRightarrow{\forall \mathfrak{S}^{\mathfrak{K}} \subseteq \mathfrak{S}^K \left( \mathfrak{V} \in \mathfrak{V} = \bigcup_{\mathfrak{K}} \mathfrak{S}^{\mathfrak{K}} \right)} \mathfrak{V}$$

is possible only if $\mathfrak{S}^K$ contains enough substates $\mathfrak{S}^{\mathfrak{K}}{}_j$ to construct each state $\mathfrak{V} \in \mathfrak{V}$ completely.

The set of substates $\mathfrak{S}^K$ can be equivalent to the state space $\mathbb{V}^n$ if this state-space contains enough substates to construct each state $\mathfrak{V} \in \mathbb{V}^n$. We will denote such a set by $\mathbb{S}^K$.

### *2.2. Representation of the Dependence of Y on X as a Set of Substates: $Y = \mathfrak{S}^{X|\mathfrak{G}}$*

The dependence of the variables $Y$ upon $X$ can be represented as a set of states $\mathfrak{S}^K$. This representation is an alternative to a set of functions $F(X|\mathfrak{G})$. In this case, it is convenient to choose the values $\mathfrak{X} \in \mathbb{X}^n$ as the key $\mathfrak{K}$ and the subset of the values $\mathfrak{Y} \in \mathbb{Y}^n$ as the values of $\mathfrak{S}$ (including $\mathfrak{S} = \emptyset$).

---

***Definition**: Let us define the substate* $\mathfrak{S}^{\mathfrak{X}}{}_q$, *where the key is* $\mathfrak{K} = \mathfrak{X}$, $\mathfrak{X} \in \mathbb{X}^n$, *the value* $\mathfrak{S} \subseteq \mathfrak{Y}$, $\mathfrak{Y} \in \mathbb{Y}^n$, *and the index* $q \in \mathbb{N}$ *is such that*

$$\forall\, \mathfrak{S}^{\mathfrak{X}}{}_q,\ \mathfrak{S}^{\mathfrak{X}}{}_{q'} \left( \mathfrak{X} = \mathfrak{X}',\ q \neq q' \right)$$

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 21 April 2025 doi:10.20944/preprints202504.1608.v1



*also,* $\mathfrak{X} \subseteq \mathfrak{S}^{\mathfrak{X}}{}_q$ *and* $\left(\mathfrak{S}^{\mathfrak{X}}{}_q \not\ni \mathfrak{X}\right) \subseteq \mathfrak{Y}$.

---

Since the values of the parameters $\mathfrak{G} \in \mathbb{G}^n$ are constant for any possible values of $\mathfrak{X}$ and $\mathfrak{Y}$, they are also constant for any possible substates $\mathfrak{S}^{\mathfrak{X}}{}_q$ composed of the values of $\mathfrak{X}$ and $\mathfrak{Y}$. Thus, the definition of $\mathfrak{S}^{\mathfrak{X}}{}_q$does not include values $\mathfrak{G}$. Being joined into a set, the substates $\mathfrak{S}^{\mathfrak{X}}{}_q$ will have the same parameters $\mathfrak{G}$, but may have different values of the key $\mathfrak{X}$.

---

***Definition***: *Let us define the dependence*

$$Y = \mathfrak{S}^{X|\mathfrak{G}} := \{\mathfrak{S}^{\mathfrak{X}}{}_q\}|\mathfrak{G}$$

*which represents the dependence of the variables* $Y$ *upon* $X$ *for given parameters* $\mathfrak{G}$ *that are the same for all substates included in the set* $\mathfrak{S}^{X|\mathfrak{G}}$. *At the same time, substates should not be contradictory:*

---

$$\nexists\, \mathfrak{S}^{\mathfrak{X}}{}_i, \mathfrak{S}^{\mathfrak{X}}{}_j \in \mathfrak{S}^{X|\mathfrak{G}} \left(\exists\, \mathfrak{v}_l \in \mathfrak{S}^{\mathfrak{X}}{}_i \neq \mathfrak{v}_l \in \mathfrak{S}^{\mathfrak{X}}{}_j \;\forall\, l\right) \quad , \tag{1}$$

The set $\mathfrak{S}^{X|\mathfrak{G}}$ with all substates having the same key $\mathfrak{X}$ will be denoted $\mathfrak{S}^{\mathfrak{X}|\mathfrak{G}}$.

Let us note that we do not impose a completeness restriction upon the set $\mathfrak{S}^{X|\mathfrak{G}}$—i.e., $\mathfrak{S}^{X|\mathfrak{G}}$ may not contain all of the keys $\mathfrak{X} \in \mathbb{X}^n$ or may even be empty: $\mathfrak{S}^{X|\mathfrak{G}} = \emptyset$. $\mathfrak{S}^{X|\mathfrak{G}}$ may also not contain all $\mathfrak{Y} \in \mathbb{Y}^n$ and/or it may not contain enough $\mathfrak{S}^{\mathfrak{X}}{}_q$ to build one or more complete $\mathfrak{Y}$.

The representation $Y = \mathfrak{S}^{X|\mathfrak{G}}$ is equivalent to the representation $Y = F(X|\mathfrak{G})$ if and only if, for each $\mathfrak{X} \in \mathbb{X}^n$ at a given $\mathfrak{G} \in \mathbb{G}^n$, the representations are equal:

$$Y = \mathfrak{S}^{X|\mathfrak{G}} \Leftrightarrow Y =$$

$$= F(X|\mathfrak{G}) \Rightarrow \forall \mathfrak{X} \in \mathbb{X}^n \left(F(\mathfrak{X}|\mathfrak{G}) = \mathfrak{S}^{\mathfrak{X}|\mathfrak{G}}\right)$$

For each key $\mathfrak{X}$, there exists a set of substates $\mathfrak{S}^{\mathfrak{X}}{}_q$ that cover all possible values $\mathfrak{Y} \in \mathbb{Y}^n$. We will denote this set by $\mathbb{S}^{\mathfrak{X}|\mathfrak{G}}$. This set may not satisfy the consistency criterion (**formula 1**) and will have cardinality

$$\left|\mathbb{S}^{\mathfrak{X}|\mathfrak{G}}\right| = \prod_{i=1}^{n} |\mathfrak{y}_i|$$

If we join the sets $\mathbb{S}^{\mathfrak{X}|\mathfrak{G}}$ for all possible keys $\mathfrak{X} \in \mathbb{X}^n$, we obtain the set of all possible substates. We will denote it by

$$\mathbb{S}^{\mathbb{X}|\mathfrak{G}} = \bigcup_{\mathfrak{X} \in \mathbb{X}^n} \mathbb{S}^{\mathfrak{X}|\mathfrak{G}} \quad , \tag{2}$$

The cardinality of this set when $\mathfrak{S} = \mathfrak{Y}$ will be

$$\left|\mathbb{S}^{\mathbb{X}|\mathfrak{G}}\right| = \sum_{\mathfrak{X} \in \mathbb{X}^n} \left|\mathbb{S}^{\mathfrak{X}|\mathfrak{G}}\right|$$

Moreover, $\left|\mathbb{S}^{\mathbb{X}|\mathfrak{G}}\right| \leq |\mathbb{V}^n|$ since, from the set $\mathbb{G}^n$ only, one set of values $\mathfrak{G}$ is used (note, the cardinalities will be equal in case $|\mathbb{G}^n| = 1$).

In practice, we will more often see sparse $\mathfrak{S}^{X|\mathfrak{G}}$, where it is impossible to completely construct $\mathfrak{Y}$ for every $\mathfrak{X} \in \mathbb{X}^n$. The use of sparse $\mathfrak{S}^{X|\mathfrak{G}}$ will reduce the modeling accuracy. In general, this is not a problem from an engineering standpoint since increasing or decreasing the cardinality $\mathfrak{S}^{X|\mathfrak{G}}$ allows us to choose an acceptable accuracy level for solving a specific simulation problem.

### *2.3. Reflection* $\breve{\boldsymbol{Y}}(\bar{\boldsymbol{X}}|\mathfrak{G})$ *as a Record of Changes in the Values of Variables*

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 21 April 2025 doi:10.20944/preprints202504.1608.v1



We can reflect the behavior of a modeled object by measuring its properties and recording the corresponding values of the variables $X$, $Y$, and $G$. By abstracting from a specific implementation, we will call such a record a **reflection** of the modeled object.

---

***Definition***: *Let us define the reflection* $\check{Y}(\bar{X}|\mathfrak{G})$ *as an arbitrary representation of the dependence of the dependent variables* $\check{Y}$ *upon the independent variables* $\bar{X}$ *and the values of the parameters* $\mathfrak{G}$. *Moreover, this dependence is constructed by studying and measuring the modeled object's properties.*

---

We can graphically represent the building of the reflection $\check{Y}(\bar{X}|\mathfrak{G})$ by adding the points

$$\mathfrak{V} = \overline{\mathfrak{X}} \cup \overline{\check{\mathfrak{Y}}} \cup \mathfrak{G}$$

into the state space $\mathbb{V}^n$ at the coordinates $\bar{X}, \check{Y}, \mathfrak{G}$, where $\overline{\mathfrak{X}} \in \overline{\mathbb{X}^n}$, $\overline{\check{\mathfrak{Y}}} \in \overline{\check{\mathbb{Y}}^n}$, and $\mathfrak{G} \in \mathbb{G}^n$. The added points will form a geometric figure that reflects the behavior of the modeled object.

A reflection can be represented as a set of functions

$$\check{Y}(\bar{X}|\mathfrak{G})^F = F(\bar{X}|\mathfrak{G}) = \check{Y}$$

or as a set of states

$$\check{Y}(\bar{X}|\mathfrak{G})^S = \mathfrak{S}^{\bar{X}|\mathfrak{G}} = \check{Y}$$

In the first case, a set of functions can be constructed by recording the obtained or measured values of the variables $X$, $Y$, and $G$. [17] In the second case, from the values of $\check{\mathfrak{Y}}$ obtained or measured with respect to $\overline{\mathfrak{X}}$ and $\mathfrak{G}$, the substate $\mathfrak{S}^{\overline{\mathfrak{X}}}{}_{q=1}$ can be directly built and added to the set of substates $\mathfrak{S}^{\bar{X}|\mathfrak{G}}$.

In this case, writing down the values of the stopwatch (which reflects the variable $t$) and the level gauge (which reflects $v_{water}$), we obtain the function $v_{water}(t)$, which reflects the dependence of $v_{water}$ on $t$. In practice, this function will be defined only on a certain interval or several intervals of the time $t_{\text{measuring}}$, during which the measurement was performed.

### *2.4. Model* $\hat{\boldsymbol{Y}}(\bar{\boldsymbol{X}}|\mathfrak{G})$ *as an Imitation of Changes in the Variables V*

In one of several ways, we can define the dependences of the variables $Y$ on $X$ and $G$ without directly measuring the properties of the modeled object [18–20]. We will call the dependence defined in this way the **model** of the modeled object.

---

***Definition***: *The model of the modeled object* $\hat{Y}(\bar{X}|\mathfrak{G})$ *is an arbitrary representation or implementation of the dependence of the dependent variables* $\hat{Y}$ *upon the independent variables* $\bar{X}$ *and the values of the parameters* $\mathfrak{G}$. *Moreover, this dependence is constructed without the direct participation of the modeled object.*

---

We can graphically represent the model $\hat{Y}(\bar{X}|\mathfrak{G})$ as a geometrical figure in the state space $\mathbb{V}^n$ consisting of the points

$$\mathfrak{V} = \overline{\mathfrak{X}} \cup \overline{\hat{\mathfrak{Y}}} \cup \mathfrak{G}$$

that define the relationship between the variables $\hat{Y}$ and $\bar{X}$ and the parameters $\mathfrak{G}$, where $\overline{\mathfrak{X}} \in \overline{\mathbb{X}^n}$, $\overline{\hat{\mathfrak{Y}}} \in \overline{\hat{\mathbb{Y}}^n}$ and $\mathfrak{G} \in \mathbb{G}^n$.

The model can be implemented as a set of possibly partial functions

$$\hat{Y}(\bar{X}|\mathfrak{G})^F = F(\bar{X}|\mathfrak{G}) = \hat{Y} \quad , \qquad (\mathbf{3})$$

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 21 April 2025 doi:10.20944/preprints202504.1608.v1



or as a set of states

$$\hat{Y}(\bar{X}|\mathfrak{G})^S = \mathfrak{S}^{\bar{X}|\mathfrak{G}} = \hat{Y} \quad , \tag{4}$$

In the first case, the set of functions can be determined analytically or in another way. In the second case, many substates must be pre-built in one way or another.

We can define the model $\hat{Y}(\bar{X}|\mathfrak{G})$ such that it completely coincides with certain reflection $\check{Y}(\bar{X}|\mathfrak{G})$; however, it is much more reasonable and useful to construct $\hat{Y}(\bar{X}|\mathfrak{G})$ to predict changes in the modeled object.

From a practical point of view, we are interested in how accurately the constructed model $\hat{Y}(\bar{X}|\mathfrak{G})$ corresponds to the modeled object. One way to determine compliance is to compare the model and reflection $\check{Y}(\bar{X}|\mathfrak{G})$ (i.e., to calculate the magnitude of their inconsistency in one way or another). Let us denote the inconsistency value by $\varepsilon$.

For example, for the case in which all variables $V$ have domain $\mathbb{R}$, we can define $\varepsilon \in \mathbb{R}$ as the integral sum of the difference of the values $\check{Y}$ and $\hat{Y}$ for each $\bar{\mathfrak{x}} \in \overline{\mathbb{X}^n}$:

$$\varepsilon = \sum_{\bar{\mathfrak{x}} \in \overline{\mathbb{X}^n}} \sum_{i=1}^{|Y|} \left( \check{Y}(\bar{\mathfrak{x}}|\mathfrak{G})_i - \hat{Y}(\bar{\mathfrak{x}}|\mathfrak{G})_i \right)$$

where $\mathfrak{G} \in \mathbb{G}^n$.

### *2.5. Simulation of the Model $\hat{\boldsymbol{Y}}(\bar{\boldsymbol{X}}|\mathfrak{G})$ as a Calculation of a Subset of $\hat{\mathfrak{Y}} \subseteq \widehat{\mathbb{Y}^n}$ from the Subset $\bar{\mathfrak{x}} \subseteq \overline{\mathbb{X}^n}$ and the Parameters $\mathfrak{G}$*

The simulation task can be reduced to obtaining or calculating the subset of the unknown values of the dependent variables $\hat{Y}$ from the subset of the known values of the independent variables $\bar{X}$ and the values of the parameters $\mathfrak{G}$ using a certain model $\hat{Y}(\bar{X}|\mathfrak{G})$.

---

***Definition***: *Let us define the simulation as the operator*

$$\bar{\mathfrak{x}} \overset{\hat{Y}(\bar{X}|\mathfrak{G})}{\Longrightarrow} \hat{\mathfrak{Y}} \quad , (5)$$

*where $\bar{\mathfrak{x}} \subseteq \overline{\mathbb{X}^n}$ is a possibly ordered set of unique known values of independent variables, $\hat{\mathfrak{Y}} \subseteq \widehat{\mathbb{Y}^n}$ is the desired set of possibly not unique values of the dependent variables, and $\hat{Y}(\bar{X}|\mathfrak{G})$ is a certain model used to obtain the desired $\hat{\mathfrak{Y}} \in \widehat{\mathbb{Y}^n}$ for a given $\bar{\mathfrak{x}} \in \overline{\mathbb{X}^n}$.*

---

For the case where the model is implemented as a set of functions (**formula 3**), the simulation

$$\bar{\mathfrak{x}} \overset{\hat{Y}(\bar{X}|\mathfrak{G})^F}{\Longrightarrow} \hat{\mathfrak{Y}}$$

is simply a calculation of the result $\hat{\mathfrak{Y}} \in \hat{\mathfrak{Y}}$ for each argument $\bar{\mathfrak{x}} \in \bar{\mathfrak{x}}$:

$$\bar{\mathfrak{x}} \overset{\forall \bar{\mathfrak{x}} \in \bar{\mathfrak{x}} \left( \hat{\mathfrak{y}} \in \hat{\mathfrak{Y}} = F(\bar{X}|\mathfrak{G})(\bar{\mathfrak{x}}) \right)}{\Longrightarrow} \hat{\mathfrak{Y}}$$

where

$$\hat{\mathfrak{Y}} = F(\bar{X}|\mathfrak{G})(\bar{\mathfrak{x}})$$

which is the operation for calculating $\hat{\mathfrak{Y}} \in \widehat{\mathbb{Y}^n}$ for a given $\bar{\mathfrak{x}} \in \overline{\mathbb{X}^n}$. For a model implemented as a set of substates (**formula 4**), the simulation is a matter of finding all substates for each key $\bar{\mathfrak{x}} \in \bar{\mathfrak{x}}$ and then building the values of $\hat{\mathfrak{Y}} \in \hat{\mathfrak{Y}}$ from the found substates

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 21 April 2025 doi:10.20944/preprints202504.1608.v1



$$\overline{\mathfrak{X}} \xrightarrow{\forall \overline{\mathfrak{x}} \in \overline{\mathfrak{X}} \left( \widehat{\mathfrak{y}} \in \widehat{\mathfrak{Y}} = \bigcup \mathfrak{S}^{\overline{X}|\mathfrak{G}}(\overline{\mathfrak{x}}) \right)} \widehat{\mathfrak{Y}} \quad , (6)$$

where

$$\mathfrak{S}^{\overline{X}|\mathfrak{G}}(\overline{\mathfrak{X}}) = \mathfrak{S}^{\overline{\mathfrak{X}}|\mathfrak{G}}$$

is the operation for selecting a subset $\mathfrak{S}^{\overline{\mathfrak{X}}|\mathfrak{G}} \subseteq \mathfrak{S}^{\overline{X}|\mathfrak{G}}$ of substates $\mathfrak{S}^{\mathfrak{X}|\mathfrak{G}}{}_j$ with the same key $\mathfrak{X}$.

A simulation can be interactive—i.e., it can react with external events and produce the results to the outside right during the calculation. In the simplest case, an interactive simulation can be represented as a series of simulations

$$\left\{ \overline{\mathfrak{X}}_i \xrightarrow{\hat{Y}(\bar{X}|\mathfrak{G})_i} \widehat{\mathfrak{Y}}_i \right\} \quad , (7)$$

of the set of models

$$\left\{ \hat{Y}(\bar{X}|\mathfrak{G})_i \right\}$$

for the corresponding sets of subsets of values of independent variables $\{\overline{\mathfrak{X}}_i\}$ sequentially received during the interaction and the sets of dependent $\{\widehat{\mathfrak{Y}}_i\}$ sequentially returned as simulation results.

## 3. Graph Modeling

We show how the model $\hat{Y}(\bar{X}|\mathfrak{G})$ can be represented as a transition graph and how a simulation can be performed for this representation. We define and prove the rules for constructing a consistent transition graph. Before reading this section, we recommend to check **Appendix C**, which describes transition function concepts. In **Section 5** and **Appendix G**, we present a simple example of the construction and simulation of a transition graph.

### *3.1. The Transition Graph $\boldsymbol{\Gamma}^{|\mathfrak{G}}$ and the Simulation Graph $\gamma^{|\mathfrak{G}}$*

We can join function $\Theta^{|\mathfrak{P}}{}_j \in \boldsymbol{\Theta}^{|\mathfrak{G}}$ and a set of functions $\Theta^{|\mathfrak{O}}{}_{i,k=1}, \dots, \Theta^{|\mathfrak{O}}{}_{i,k=n} \in \boldsymbol{\Theta}^{|\mathfrak{G}}$ represented as graphs by combining the result nodes

$$\xrightarrow{\Theta^{|\mathfrak{O}}{}_{i,k=1}} \grave{\mathrm{S}}\colon \grave{\mathbb{S}}_{i,k=1}, \dots, \xrightarrow{\Theta^{|\mathfrak{O}}{}_{i,k=n}} \grave{\mathrm{S}}\colon \grave{\mathbb{S}}_{i,k=n}$$

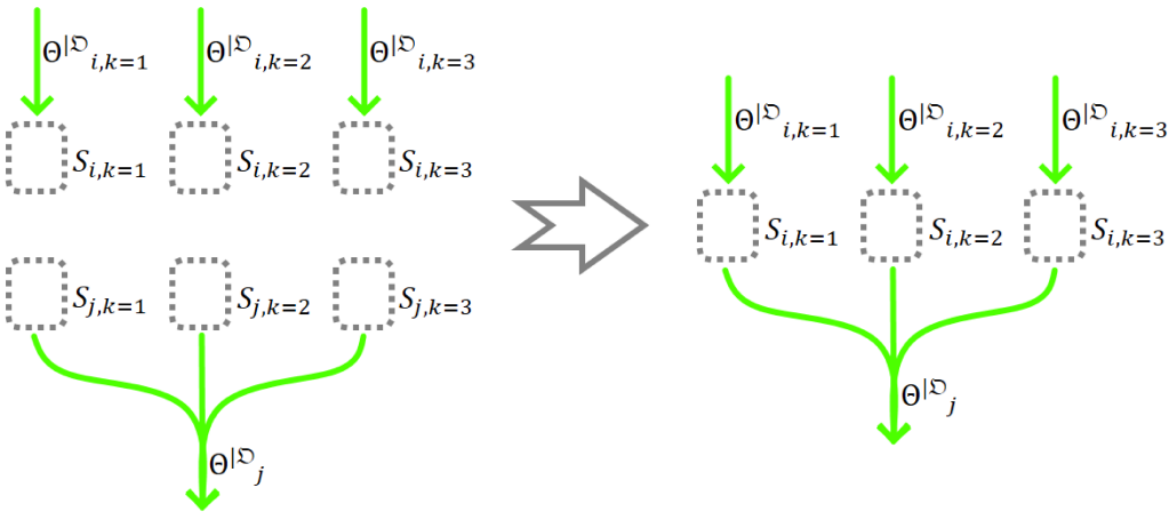
and the argument nodes

$$\mathrm{S}\colon \mathbb{S}_{k=1,j}, \dots, S\colon \mathbb{S}_{k=n,j} \xrightarrow{\Theta^{|\mathfrak{O}}{}_j}$$

with intermediate-variable nodes

$$\left( \xrightarrow{\Theta^{|\mathfrak{O}}{}_{i,k=1}} \mathrm{S}_{k=1}\colon \grave{\mathbb{S}}_{i,k=1}, \dots, \xrightarrow{\Theta^{|\mathfrak{O}}{}_{i,k=n}} \mathrm{S}_{k=n}\colon \grave{\mathbb{S}}_{i,k=n} \right) \xrightarrow{\Theta^{|\mathfrak{O}}{}_j}$$

(see **Figure 1**).



Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 21 April 2025 doi:10.20944/preprints202504.1608.v1



**Figure 1.** Joining of function $\Theta^{|\mathfrak{R}}{}_j \in \boldsymbol{\Theta}^{|\mathfrak{G}}$.

We continue in the same way to sequentially join the functions included in the same set $\boldsymbol{\Theta}^{|\mathfrak{G}}$ (possibly using the same function more than once); we obtain some DAG (see **Figure 2**). We call such DAG a **transition graph**. Also, optionally we can combine two or more root variables $S_{k,i=1}, \dots, S_{k,i=n}$ that do not have incoming edges, thereby reducing the total number of nodes.

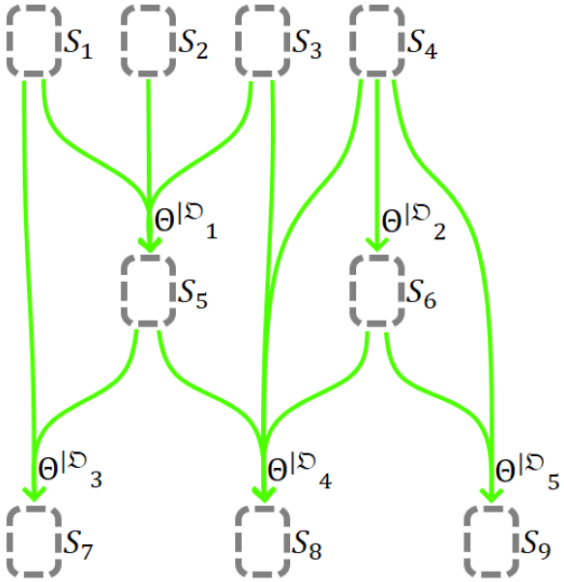



**Figure 2.** Example of the transition DAG built from functions $\Theta^{|\mathfrak{R}}{}_j \in \boldsymbol{\Theta}^{|\mathfrak{G}}$.

---

***Definition***: *We define the transition graph* $\Gamma^{|\mathfrak{G}}$ *as a DAG constructed on the set of transition functions* $\boldsymbol{\Theta}^{|\mathfrak{G}}$ *by sequentially joining arbitrary subsets of functions*

$$\Theta^{|\mathfrak{D}}{}_i, \Theta^{|\mathfrak{D}}{}_{j,k=1}, \dots, \Theta^{|\mathfrak{D}}{}_{j,k=n} \in \boldsymbol{\Theta}^{|\mathfrak{G}}$$

*and by combining the result node*

$$\grave{S}_i : \grave{\mathbb{S}}_i \xleftarrow{\Theta^{|\mathfrak{D}}{}_i}$$

*and the argument nodes*

$$S_{k,j,k=1} : \mathbb{S}_{k,j,k=1} \xrightarrow{\Theta^{|\mathfrak{D}}{}_{j,k=1}}, \dots$$

$$\dots, S_{k,j,k=n} : \mathbb{S}_{k,j,k=n} \xrightarrow{\Theta^{|\mathfrak{D}}{}_{j,k=n}}$$

*such that*

$$\grave{\mathbb{S}}_i \subseteq \mathbb{S}_{k,j,k=1}, \dots, \mathbb{S}_{k,j,k=n}$$

$$\grave{\mathbb{S}}_i \cap \mathbb{S}_{k,j,k=1}, \dots, \mathbb{S}_{k,j,k=n} \neq \emptyset$$

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 21 April 2025 doi:10.20944/preprints202504.1608.v1



*into intermediate variable nodes*

$$\xrightarrow{\Theta^{|\mathfrak{D}}{}_i} S\colon \grave{\mathbb{S}}_i \xrightarrow{\Theta^{|\mathfrak{D}}{}_{j,k=1},\ldots,\Theta^{|\mathfrak{D}}{}_{j,k=n}}$$

*Additionally, the root nodes*

$$S_{k,i=1} \cup \ldots \cup S_{k,i=n}$$

*and their domains*

$$\mathbb{S}_{k,i=1} \cap \ldots \cap \mathbb{S}_{k,i=1}$$

*may also be combined.*

---

We note that this definition imposes no restrictions on the graph structure except for its acyclicity (the result of the next joined $\Theta^{|\mathfrak{D}}$ cannot be connected with the argument of any already joined $\Theta^{|\mathfrak{D}}$) and continuity (all nodes of the graph $\Gamma^{|\mathfrak{G}}$ are connected by at least one edge).

When we join the transition functions $\Theta^{|\mathfrak{D}}$, we also join the transitions $\theta^{|\mathfrak{D}}$ from the equivalent set $\boldsymbol{\theta}^{|\mathfrak{D}} \Leftrightarrow \Theta^{|\mathfrak{D}}$, forming a set of more complex DAGs with the same structure as the graph $\Gamma^{|\mathfrak{G}}$, but which consist of the substates $\mathfrak{S}^{\mathfrak{X}}{}_q$ and transitions $\theta^{|\mathfrak{D}}$ (see **Figure 3**). We call such DAGs **simulation graphs**.

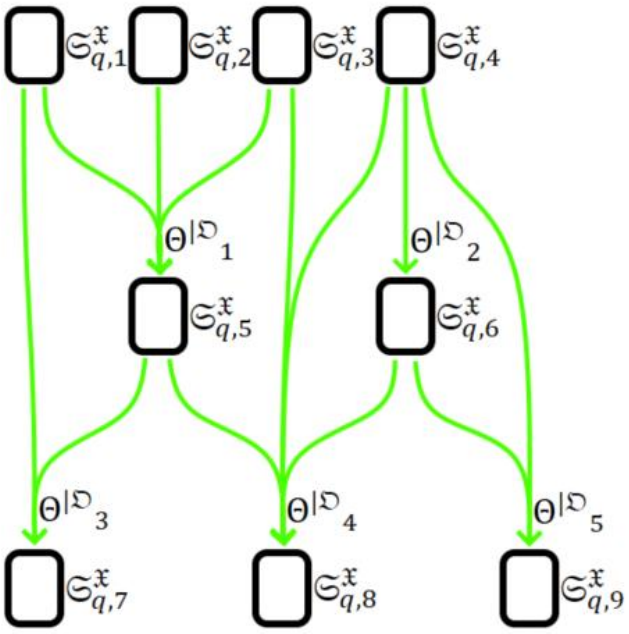



**Figure 3.** Example of the simulation graph that can be obtained from the transition graph in Figure 2.

---

***Definition***: *A simulation graph* $\gamma^{|\mathfrak{G}}$ *is defined as the DAG obtained by constructing a transition graph* $\Gamma^{|\mathfrak{G}}$*; it has the same structure as* $\Gamma^{|\mathfrak{G}}$*. The graph* $\gamma^{|\mathfrak{G}}$ *consists of constructions of the form*

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 21 April 2025 doi:10.20944/preprints202504.1608.v1



$$\xrightarrow{\theta^{|\mathfrak{O}}{}_i} \mathfrak{S}^{\mathfrak{x}}{}_{q,i} \xrightarrow{\theta^{|\mathfrak{O}}{}_{j,k=1},\dots,\theta^{|\mathfrak{O}}{}_{j,k=n}}$$

*which result from joining the transitions*

$$\theta^{|\mathfrak{O}}{}_i \in \boldsymbol{\theta}^{|\mathfrak{O}}{}_i \Leftrightarrow \Theta^{|\mathfrak{O}}{}_i$$

*and*

$$\theta^{|\mathfrak{O}}{}_{j,k=1} \in \boldsymbol{\theta}^{|\mathfrak{O}}{}_{j,k=1} \Leftrightarrow$$

$$\Leftrightarrow \Theta^{|\mathfrak{O}}{}_{j,k=1}, \dots, \theta^{|\mathfrak{O}}{}_{j,k=n} \in \boldsymbol{\theta}^{|\mathfrak{O}}{}_{j,k=n} \Leftrightarrow$$

$$\Leftrightarrow \Theta^{|\mathfrak{O}}{}_{j,k=n}$$

*belonging to the set of joined functions*

$$\Theta^{|\mathfrak{O}}{}_i, \theta^{|\mathfrak{O}}{}_{j,k=1}, \dots, \theta^{|\mathfrak{O}}{}_{j,k=n} \in \boldsymbol{\Theta}^{|\mathfrak{G}}$$

*such that*

$$\grave{\mathfrak{S}}^{\mathfrak{x}}{}_{q,i} \xleftarrow{\theta^{|\mathfrak{O}}{}_i} = \mathfrak{S}^{\mathfrak{x}}{}_{q,j,k=1} \xrightarrow{\theta^{|\mathfrak{O}}{}_{j,k=1}} =, \dots$$

$$\dots, = \mathfrak{S}^{\mathfrak{x}}{}_{q,j,k=n} \xrightarrow{\theta^{|\mathfrak{O}}{}_{j,k=n}}$$

---

We note that all $\gamma^{|\mathfrak{G}}$ will have a structure exactly matching $\Gamma^{|\mathfrak{G}}$. According to the definition of $\gamma^{|\mathfrak{G}}$, during the construction of $\Gamma^{|\mathfrak{G}}$, incomplete graphs of $\gamma^{|\mathfrak{G}}$ with structures not coinciding with that of $\Gamma^{|\mathfrak{G}}$ will be discarded. Thus, the substates $\mathfrak{S}^{\mathfrak{x}}{}_q$ included in the discarded graphs $\gamma^{|\mathfrak{G}}$ will also be removed from the domains $\mathbb{S}$ of the variables $S$ included in the constructed $\Gamma^{|\mathfrak{G}}$.

Let us denote some arbitrary set of graphs $\{\gamma^{|\mathfrak{G}}\}$ by $\boldsymbol{\gamma}^{|\mathfrak{G}}$. According to the definitions of the graphs $\Gamma^{|\mathfrak{G}}$ and $\gamma^{|\mathfrak{G}}$, each of the substates $\mathfrak{S}^{\mathfrak{x}}{}_q$ from the domains $\mathbb{S}$ of the variable nodes $\mathrm{S}$ will belong to one of the simulation graphs $\gamma^{|\mathfrak{G}}$. All $\mathfrak{S}^{\mathfrak{x}}{}_q$ terms that do not belong to any $\gamma^{|\mathfrak{G}}$ will be discarded during the construction of $\Gamma^{|\mathfrak{G}}$, along with the incomplete $\gamma^{|\mathfrak{G}}$.

Thus, we can represent the graph $\Gamma^{|\mathfrak{G}}$ as an equivalent set of graphs $\gamma^{|\mathfrak{G}}$. We will denote such a set as $\boldsymbol{\gamma}^{|\mathfrak{G}} \Leftrightarrow \Gamma^{|\mathfrak{G}}$; this set will include all $\mathfrak{S}^{\mathfrak{x}}{}_q$ from all domains $\mathbb{S}$:

$$\bigcup \mathbb{S}\left(\Gamma^{|\mathfrak{G}}\right) = \bigcup_{\gamma^{|\mathfrak{G}} \in \boldsymbol{\gamma}^{|\mathfrak{G}} \Leftrightarrow \Gamma^{|\mathfrak{G}}} \mathfrak{S}^{\mathfrak{x}}\left(\gamma^{|\mathfrak{G}}\right)$$

where $\mathbb{S}\left(\Gamma^{|\mathfrak{G}}\right)$ is the set of domains $\mathbb{S}$ of the variable nodes $\mathrm{S}$ from the graph $\Gamma^{|\mathfrak{G}}$ and $\mathfrak{S}^{\mathfrak{x}}\left(\gamma^{|\mathfrak{G}}\right)$ is the set of all $\mathfrak{S}^{\mathfrak{x}}{}_q$ belonging to $\gamma^{|\mathfrak{G}}$, which is consistent:

$$\exists\, \mathfrak{S}^{X|\mathfrak{G}} \left(\mathfrak{S}^{\mathfrak{x}}\left(\gamma^{|\mathfrak{G}}\right) = \mathfrak{S}^{X|\mathfrak{G}}\right)$$

Moreover, all simulation graphs $\gamma^{|\mathfrak{G}}$ will share the same set of parameters $\mathfrak{G}$ split into parts $\mathfrak{O}$.

Let us index each node from the set $S\left(\Gamma^{|\mathfrak{G}}\right)$ with the depth index

$$d = max(len(\{\mathbf{S}_{root} \dots S_d\})),$$

where $d \in \mathbb{N}$, $\{\mathbf{S}_{root} \dots S_d\}$ is the set of all possible paths from any root node $S_{root} \in \mathbf{S}_{root}$ (i.e., the node that has no incoming edges) to the indexed node $S_d$ and $len()$ is the length of the path (the number of edges in the path). Let us also index all transition functions $\Theta^{|\mathfrak{O}}$ with the same index $d$ same as the index of the result node

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 21 April 2025 doi:10.20944/preprints202504.1608.v1



$$\ldots \xrightarrow{\Theta^{|\mathfrak{O}}{}_{d}} S_d$$

Thus, each root node of $S_{root}$ will have $d = 0$ and each leaf node $S_{leaf}$ (i.e., such that it has no outgoing edges) will have $d = n$, where $n$ is the minimum number of edges to the nearest root node $S_{d=0}$.

*3.2. Construction of a Consistent Transition Graph* $\Gamma^{|\mathfrak{G}}$

From a practical viewpoint, we want to be able to construct transitions $\Gamma^{|\mathfrak{G}}$ from a set of predefined transition functions $\boldsymbol{\Theta}^{|\mathfrak{G}}$—in other words, to build models from a set of ready-made functional blocks, similar to the Simulink software. It is necessary to guarantee the consistency of $\Gamma^{|\mathfrak{G}}$ at the local level, i.e., at the level of individual functions $\Theta^{|\mathfrak{P}}$, to implement this approach successfully.

We can represent some simulation graph $\gamma^{|\mathfrak{G}}$ as the set of directional paths (dipaths) covering all substates $\mathfrak{S}^{\mathfrak{X}}{}_{q} \in \mathfrak{S}^{\mathfrak{X}}(\gamma^{|\mathfrak{G}})$ and transitions $\theta^{|\mathfrak{O}} \in \theta(\gamma^{|\mathfrak{G}})$.

---

***Definition**: Let us define a dipath*

$$\mathfrak{p}^{|\mathfrak{G}} := \mathfrak{S}^{\mathfrak{X}}{}_{q,l=0} \xrightarrow{\theta^{|\mathfrak{O}}{}_{l=1}} \ldots \xrightarrow{\theta^{|\mathfrak{O}}{}_{l=n}} \mathfrak{S}^{\mathfrak{X}}{}_{q,\,l=n}$$

*in the simulation graph* $\gamma^{|\mathfrak{G}} \in \boldsymbol{\gamma}^{|\mathfrak{G}} \Leftrightarrow \Gamma^{|\mathfrak{G}}$, *where* $\mathfrak{S}^{\mathfrak{X}}{}_{q,l=0} \in \mathbb{S}_{l=0}$, $\mathfrak{S}^{\mathfrak{X}}{}_{q,l=n} \in \mathbb{S}_{l=n}$, $l \in \mathbb{N}$ *is the index of the node which is in the dipath, such that* $l = 0$ *corresponds to some root node* $\mathfrak{S}^{\mathfrak{X}}{}_{q,root}$ *and* $l = n$ *corresponds to some leaf node* $\mathfrak{S}^{\mathfrak{X}}{}_{q,leaf}$ *in the graph* $\gamma^{|\mathfrak{G}}$, $\mathfrak{S}^{\mathfrak{X}}{}_{q,l=0}, \ldots, \mathfrak{S}^{\mathfrak{X}}{}_{q,l=n} \in \mathfrak{S}^{\mathfrak{X}}(\gamma^{|\mathfrak{G}})$, $\theta^{|\mathfrak{O}}{}_{l=1}, \ldots, \theta^{|\mathfrak{O}}{}_{l=n} \in \theta(\gamma^{|\mathfrak{G}})$, $\mathbb{S}_{l=0}, \ldots, \mathbb{S}_{l=n} \in \mathbb{S}(\Gamma^{|\mathfrak{G}})$, *with* $\mathfrak{S}^{\mathfrak{X}}(\mathfrak{p}^{|\mathfrak{G}}) \subseteq \mathfrak{S}^{\mathfrak{X}}(\gamma^{|\mathfrak{G}})$.

---

We denote some arbitrary set of paths by $\boldsymbol{\mathfrak{p}}^{|\mathfrak{G}}$, which is not necessarily related to the same graph $\gamma^{|\mathfrak{G}}$.

The set of paths $\boldsymbol{\mathfrak{p}}^{|\mathfrak{G}}$ can be equivalent to the graph $\gamma^{|\mathfrak{G}}$ if the paths in this set contain all substates $\mathfrak{S}^{\mathfrak{X}}{}_{q} \in \mathfrak{S}^{\mathfrak{X}}(\gamma^{|\mathfrak{G}})$ and transitions $\theta^{|\mathfrak{O}} \in \theta(\gamma^{|\mathfrak{G}})$:

$$\boldsymbol{\mathfrak{p}}^{|\mathfrak{G}} \Leftrightarrow \gamma^{|\mathfrak{G}} \Rightarrow \bigcup_{\mathfrak{p}^{|\mathfrak{G}} \ni \boldsymbol{\mathfrak{p}}^{|\mathfrak{G}}} \mathfrak{S}^{\mathfrak{X}}(\mathfrak{p}^{|\mathfrak{G}}) =$$

$$= \mathfrak{S}^{\mathfrak{X}}\left(\gamma^{|\mathfrak{G}}{}_{j}\right) \wedge \bigcup_{\mathfrak{p}^{|\mathfrak{G}} \ni \boldsymbol{\mathfrak{p}}^{|\mathfrak{G}}} \theta\left(\mathfrak{p}\left(\gamma^{|\mathfrak{G}}{}_{j}\right)\right) = \theta\left(\gamma^{|\mathfrak{G}}{}_{j}\right)$$

In order to guarantee the consistency condition

$$\forall\, \gamma^{|\mathfrak{G}} \in \boldsymbol{\gamma}^{|\mathfrak{G}} \Leftrightarrow \Gamma^{|\mathfrak{G}}\left(\mathfrak{S}^{\mathfrak{X}}(\gamma^{|\mathfrak{G}}) \subseteq \boldsymbol{\mathfrak{S}}^{X|\mathfrak{G}}\right)$$

for the graph $\Gamma^{|\mathfrak{G}}$ (i.e., to guarantee that each of the simulation graphs $\gamma^{|\mathfrak{G}}$ described by $\Gamma^{|\mathfrak{G}}$ will not contain any inconsistent substates), the graph $\Gamma^{|\mathfrak{G}}$ must meet the following two restrictions:

- *For each graph* $\gamma^{|\mathfrak{G}}{}_{j} \in \boldsymbol{\gamma}^{|\mathfrak{G}} \Leftrightarrow \Gamma^{|\mathfrak{G}}$, *each substate* $\mathfrak{S}^{\mathfrak{X}}{}_{q,j} \in \mathfrak{S}^{\mathfrak{X}}\left(\gamma^{|\mathfrak{G}}{}_{j}\right), \mathfrak{S}^{\mathfrak{X}}\left(\mathfrak{p}^{|\mathfrak{G}}{}_{j}\right)$ *(i.e., located on one of all possible paths* $\mathfrak{p}^{|\mathfrak{G}}{}_{j}$*) must have a unique key* $\mathfrak{X} \subseteq \mathfrak{S}^{\mathfrak{X}}{}_{q,j}$ *regarding the* $\mathfrak{p}^{|\mathfrak{G}}{}_{j}$.
- *For each graph* $\gamma^{|\mathfrak{G}} \in \boldsymbol{\gamma}^{|\mathfrak{G}} \Leftrightarrow \Gamma^{|\mathfrak{G}}$, *the values* $\mathfrak{y} \in \mathbb{y}$ *of some variable* $y{:}\,\mathbb{y} \in Y$ *should only belong to the set of substates* $\mathfrak{S}^{\mathfrak{X}}{}_{q,j} \in \mathfrak{S}^{\mathfrak{X}}\left(\gamma^{|\mathfrak{G}}{}_{j}\right)$ *such that there exists in* $\gamma^{|\mathfrak{G}}$ *at least one path* $\mathfrak{p}^{|\mathfrak{G}} \in \boldsymbol{\mathfrak{p}}^{|\mathfrak{G}} \Leftrightarrow \gamma^{|\mathfrak{G}}$, *including all of these substates.*

At the local level (i.e., without studying the entire graph $\Gamma^{|\mathfrak{G}}$), the above restrictions can be met by applying the following construction principles (**Appendix D**):

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 21 April 2025 doi:10.20944/preprints202504.1608.v1



*I.* *The set of keys* $\mathbb{X}^n$ *must be linearly ordered;*

*II.* *Each transition function*

$$(S{:}\,\mathbb{S}_{k=1},\ \dots, S{:}\,\mathbb{S}_{k=n}) \overset{\theta^{|\mathfrak{O}}}{\longrightarrow} \hat{S}{:}\,\hat{\mathbb{S}}$$

(where $\theta^{|\mathfrak{O}} \in \Theta(\Gamma^{|\mathfrak{G}})$) for each transition

$$(\mathfrak{S}^{\mathfrak{X}}{}_{q,j,k=1}, \dots, \mathfrak{S}^{\mathfrak{X}}{}_{q,j,k=n}) \overset{\theta^{|\mathfrak{O}}{}_j}{\longrightarrow} \hat{\mathfrak{S}}^{\mathfrak{X}}{}_{q,j}$$

(where $\theta^{|\mathfrak{O}}{}_j \in \theta\left(\mathfrak{p}^{|\mathfrak{G}}{}_j\right), \boldsymbol{\theta}^{|\mathfrak{O}} \Leftrightarrow \Theta^{|\mathfrak{O}}$) in some graph $\gamma^{|\mathfrak{G}}{}_j \in \boldsymbol{\gamma}^{|\mathfrak{G}} \Leftrightarrow \Gamma^{|\mathfrak{G}}$ must generate the resulting substate $\hat{\mathfrak{S}}^{\mathfrak{X}}{}_{q,j} \in \hat{\mathbb{S}}, \mathfrak{S}^{\mathfrak{X}}\left(\gamma^{|\mathfrak{G}}{}_j\right)$ such that its key $\mathfrak{X}_{q,j} \subseteq \hat{\mathfrak{S}}^{\mathfrak{X}}{}_{q,j}$ will always satisfy the conditions

$$\mathfrak{X}_{q,j} > max(\mathfrak{X}_{q,j,k=1}, \dots, \mathfrak{X}_{q,j,k=n})$$

or

$$\mathfrak{X}_{q,j} < min(\mathfrak{X}_{q,j,k=1}, \dots, \mathfrak{X}_{q,j,k=n})$$

where

$$\mathfrak{X}_{q,j,k} \subseteq \mathfrak{S}^{\mathfrak{X}}{}_{q,j,k} \in \mathbb{S}_k, \mathfrak{S}^{\mathfrak{X}}\left(\gamma^{|\mathfrak{G}}{}_j\right)$$

*III.* *For each variable* $y{:}\,\mathbb{y} \in Y$*, its values* $\mathfrak{y} \in \mathbb{y}$ *must belong to no more than one root node* $S{:}\,\mathbb{S}_{l=0}$*:*

$$\forall\, y{:}\,\mathbb{y} \in Y\ \left(\left|\left\{S{:}\,\mathbb{S}_{l=0} \in S\left(\Gamma^{|\mathfrak{G}}\right) \mid \exists\, \mathfrak{y} \in \mathbb{y}, \mathfrak{y}(\mathbb{S}_{l=0})\right\}\right| \leq 1\right)$$

where

$$\mathfrak{y}(\mathbb{S}) \coloneqq \bigcup \mathfrak{S}(\mathbb{S})$$
$$\mathfrak{S}(\mathbb{S}) \coloneqq \{\mathfrak{S}_q \mid \mathfrak{S}_q \subset \mathfrak{S}^{\mathfrak{X}}{}_q \in \mathbb{S}\}$$

(i.e., the set of all $\mathfrak{y}$ values in all substates $\mathfrak{S}^{\mathfrak{X}}{}_q$ form the domain of the variable $\mathbb{S}$).

*IV.* *If, for some node* $S{:}\,\mathbb{S}_i \in S\left(\Gamma^{|\mathfrak{G}}\right)$ *and some variable* $y{:}\,\mathbb{y} \in Y$ *the condition* $\exists\, \mathfrak{y} \in \mathbb{y}, \mathfrak{y}(\mathbb{S}_i)$ *is true, then, either the node* $S_i$ *must be a root, or there must be a transition function*

$$(\dots, S{:}\,\mathbb{S}_{i,k=0}, \dots, S{:}\,\mathbb{S}_{i,k=n}, \dots) \overset{\theta^{|\mathfrak{O}}{}_i}{\longrightarrow} \hat{S}_i$$

*with one or more arguments* $S{:}\,\mathbb{S}_{i,k}$ *for which the condition* $\exists\, \mathfrak{y} \in \mathbb{y}, \mathfrak{y}(\mathbb{S}_i)$ *is true and in the graph* $\Gamma^{|\mathfrak{G}}$ *there exists a chain*

$$S_{i,k=0} \overset{\theta^{|\mathfrak{O}}{}_{i,k=1}}{\longrightarrow}, \dots, \overset{\theta^{|\mathfrak{O}}{}_{i,k=n}}{\longrightarrow} S_{i,k=n}$$

*that includes all* $S_{i,k}$*. Moreover, for the last argument* $S_{i,k=n}$ *in the chain, there should not be another function*

$$(\dots, S_{i,k=n}, \dots) \overset{\theta^{|\mathfrak{O}}{}_i{}'}{\longrightarrow} \hat{S}_i{}'$$

*for which the condition* $\exists\, \mathfrak{y} \in \mathbb{y}, \mathfrak{y}\left(\hat{S}_i{}'\right)$ *is true.*

In practice, principle (I) can be easily implemented since linearly ordered sets are common. For example, time, speed, etc., can be represented using variables with $\mathbb{R}$. Next, if the domains of all independent variables are in linear order, then the set of keys $\mathbb{X}^n$ will also be in linear order.

Principle (II) says that the key-value constantly increases or decreases as the simulation graph $\gamma^{|\mathfrak{G}}$ is calculated. This approach can be applied, for example, to physical models, where independent variables are usually rational numbers that increase or decrease over the simulation.

Principle (III) holds if the graph $\Gamma^{|\mathfrak{G}}$ has a single root node $S{:}\,\mathbb{S}_{l=0} \in S\left(\Gamma^{|\mathfrak{G}}\right)$ such that in each graph $\gamma^{|\mathfrak{G}}{}_j$ there will be only one substate $\mathfrak{S}^{\mathfrak{X}}{}_{q,j,l=0} \in \mathbb{S}_{l=0}$, thereby excluding the possibility that the values $\mathfrak{y} \in \mathbb{y}$ of the same variable $y{:}\,\mathbb{y} \in Y$ are in different substates $\mathfrak{S}^{\mathfrak{X}}{}_{q,j,l=0}$.

Another approach to implementing (III) is for each root node $S{:}\,\mathbb{S}_{l=0}$ to include $\mathfrak{y} \in \mathbb{y}$ values only from its own unique set of variables $\boldsymbol{y}_1, \dots, \boldsymbol{y}_n \subset Y$, such that

$$\forall\, \boldsymbol{y}_i, \boldsymbol{y}_j (i \neq j,\ \boldsymbol{y}_i \cap \boldsymbol{y}_j = \emptyset)$$

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 21 April 2025 doi:10.20944/preprints202504.1608.v1



This approach, for example, is convenient in the graphs $\Gamma^{|\mathfrak{G}}$ used for interactive simulation, where each next node $S_{l=0}$ reflects the next input of data from outside the simulation.

In practice, a simple way to implement principle (IV) is to check whether adding the next function $\Theta^{|\mathfrak{D}}$ to form $\hat{S}$ does not include variables that are already in the results of the functions that have joint arguments $S$ with $\Theta^{|\mathfrak{D}}$. For example, if there are nodes $S_{k=1}$ and $S_{k=2}$ for which $y(S_{k=1}) = [a, b]$ and $y(S_{k=2}) = [x, y]$, where

$$y(S{:}\,\mathbb{S}) \coloneqq \{y{:}\,\mathbb{y} \in Y \mid \exists\, \mathfrak{y} \in \mathbb{y}, \mathfrak{y}(\mathbb{S})\}$$

and these nodes are the arguments of some function

$$(S_{k=1}, S_{k=2}) \xrightarrow{\Theta^{|\mathfrak{D}}{}_{j=1}} \hat{S}_{j=1}$$

for which the result is $y(\hat{S}_{j=1}) = [a, x]$, then we can add only a function

$$(S_{k=1}, S_{k=2}) \xrightarrow{\Theta^{|\mathfrak{D}}{}_{j=1}} \hat{S}_{j=1}$$

for which $y(\hat{S}_{j=2}) = [b, y]$ and either $y(\hat{S}_{j=2}) = [y]$ or $y(\hat{S}_{j=2}) = [b]$, but not $y(\hat{S}_{j=2}) = [a, b, y]$.

### *3.3. Computability of the Simulation Graph $\boldsymbol{\gamma}^{|\mathfrak{G}}$ and the Initial Set of Substates $\mathfrak{S}'$*

In practice, we will need to find some specific simulation graph $\gamma^{|\mathfrak{G}} \in \boldsymbol{\gamma}^{|\mathfrak{G}} \Leftrightarrow \Gamma^{|\mathfrak{G}}$ from some known set of consistent substates $\mathfrak{S}^{X|\mathfrak{G}} \subseteq \mathfrak{S}^{\mathfrak{x}}(\gamma^{|\mathfrak{G}})$ associated with the nodes $S$ of the graph $\Gamma^{|\mathfrak{G}}$. We will call $\mathfrak{S}^{X|\mathfrak{G}}$ the **initial set of substates**.

---

***Definition***: *Let us define the initial set of substates*

$$\mathfrak{S}' \coloneqq \{S = \mathfrak{S}^{\mathfrak{x}}{}_q\}$$

*associated with the specific nodes $S$ of the graph $\Gamma^{|\mathfrak{G}}$ such that*

$$\exists!\, \gamma^{|\mathfrak{G}} \left(\mathfrak{S}' \subseteq \mathfrak{S}^{\mathfrak{x}}(\gamma^{|\mathfrak{G}})\right) \quad , (\mathbf{8})$$

*where $S{:}\,\mathbb{S} \in S(\Gamma^{|\mathfrak{G}})$, $\mathfrak{S}^{\mathfrak{x}}{}_q \in \mathbb{S}, \mathfrak{S}^{\mathfrak{x}}(\gamma^{|\mathfrak{G}})$, $\gamma^{|\mathfrak{G}} \in \boldsymbol{\gamma}^{|\mathfrak{G}} \Leftrightarrow \Gamma^{|\mathfrak{G}}$.*

---

The search for a specific graph $\gamma^{|\mathfrak{G}}$ with some set $\mathfrak{S}'$ can be imperatively represented as a calculation of all functions $\Theta^{|\mathfrak{D}} \in \Theta(\Gamma^{|\mathfrak{G}})$, using $\mathfrak{S}'$ as the initial arguments for these functions.

Note that the definition requires that $\mathfrak{S}'$ be a subset of the one and only one set $\mathfrak{S}^{\mathfrak{x}}(\gamma^{|\mathfrak{G}})$. However, in the general case, some $\mathfrak{S}^{X|\mathfrak{G}}$ can be a subset of more than one $\mathfrak{S}^{\mathfrak{x}}(\gamma^{|\mathfrak{G}})$. In this case, in the imperative representation of the search, a single graph $\gamma^{|\mathfrak{G}}$ cannot be calculated from such $\mathfrak{S}^{X|\mathfrak{G}}$, since for some or all functions $\Theta^{|\mathfrak{D}} \in \Theta(\Gamma^{|\mathfrak{G}})$, not all arguments be defined.

Representing the search for a specific $\gamma^{|\mathfrak{G}}$ in the form of a calculation of the functions $\Theta^{|\mathfrak{D}} \in \Theta(\Gamma^{|\mathfrak{G}})$, we notice that all $\Theta^{|\mathfrak{D}}$ will be calculated only if the values of all root nodes $S_{root} \in S(\Gamma^{|\mathfrak{G}})$ are known or can be obtained in some way. Thus, $\mathfrak{S}'$ is a subset of the unique set $\mathfrak{S}^{\mathfrak{x}}(\gamma^{|\mathfrak{G}})$ (**formula 8**) if and only if, for each initial node $S_{root} \in S(\Gamma^{|\mathfrak{G}})$, there exists a path

$$S_{root} \xrightarrow{\Theta^{|\mathfrak{D}}{}_1} \dots \xrightarrow{\Theta^{|\mathfrak{D}}{}_n} S_{def}$$

where $S_{def} \in S(\Gamma^{|\mathfrak{G}}), S(\mathfrak{S}')$ is a node whose value is defined in $\mathfrak{S}'$. And all function $\Theta^{|\mathfrak{D}}{}_i$ on this reversible (**Appendix E**).

Another important property of this approach is the glitching freedom described in [21,22]. Since only one graph $\gamma^{|\mathfrak{G}}$ is to be found, there never exist inconsistent substates $\mathfrak{S}^{\mathfrak{x}}{}_q$.

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 21 April 2025 doi:10.20944/preprints202504.1608.v1



*3.4. Representation of the Dependence of Y on X in the Form of a Simulation Graph* $\boldsymbol{Y = \langle \Gamma^{|\mathfrak{G}}, \mathfrak{S}' \rangle^{X}}$ *and a Graph Model* $\hat{Y}(\bar{X}|\mathfrak{G})^{\Gamma}$

The dependence of the dependent variables $Y$ upon the independent variables $X$ can be represented as a tuple of the transition graph $\Gamma^{|\mathfrak{G}}$, and the set of initial substates $\mathfrak{S}'$ with given values of the parameters $\mathfrak{G}$.

---

***Definition***: *Let us define a pair*

$$Y = \langle \Gamma^{|\mathfrak{G}}, \mathfrak{S}' \rangle^{X} :=$$

$$:= \bigcup\nolimits_{\mathfrak{x} \in \mathbb{X}^n} \mathfrak{S}^{\mathfrak{x}} \left( \gamma \left( \Gamma^{|\mathfrak{G}} | \mathfrak{S}' \right) \right) (X)$$

*representing the dependence of the variables* $Y$ *on the variables* $X$*, as parametrized by the values of* $\mathfrak{G}$*, where*

$$\gamma^{|\mathfrak{G}} = \gamma \left( \Gamma^{|\mathfrak{G}} | \mathfrak{S}' \right)$$

*is a simulation graph* $\gamma^{|\mathfrak{G}} \in \boldsymbol{\gamma}^{|\mathfrak{G}} \Leftrightarrow \Gamma^{|\mathfrak{G}}$ *found for a given* $\Gamma^{|\mathfrak{G}}$, $\mathfrak{S}'$ *and* $\mathfrak{G}$*, and*

$$\mathfrak{Y} = \bigcup\nolimits_{\mathfrak{x} \in X} \mathfrak{S}^{X|\mathfrak{G}} (\mathfrak{x})$$

*is the merging operation of the substates* $\mathfrak{S}^{\mathfrak{x}}{}_{q} \in \mathfrak{S}^{X|\mathfrak{G}}$ *with the same key* $\mathfrak{x} \in \mathbb{X}^n$ *into the set of values* $\mathfrak{Y} \in \mathbb{Y}^n$.

---

The representation $Y = \langle \Gamma^{|\mathfrak{G}}, \mathfrak{S}' \rangle^{X}$ can be used to implement the model $\hat{Y}(\bar{X}|\mathfrak{G})$; we call this implementation a **graph model** and denote it as

$$\hat{Y}(\bar{X}|\mathfrak{G})^{\Gamma} = \langle \Gamma^{|\mathfrak{G}}, \mathfrak{S}' \rangle^{\bar{X}} = \hat{Y} \qquad , (\mathbf{9})$$

This implementation is similar to a representation in the form of a set of substates (**formula 4**), except that the set $\mathfrak{S}^{\bar{X}|\mathfrak{G}}$ must first be found as

$$\mathfrak{S}^{\bar{X}|\mathfrak{G}} = \mathfrak{S}^{\mathfrak{x}} \left( \gamma \left( \Gamma^{|\mathfrak{G}} | \mathfrak{S}' \right) \right)$$

*3.5. Simulation of the Graph Model* $\boldsymbol{\hat{Y}(\bar{X}|\mathfrak{G})^{\Gamma}}$ *as a Calculation of a Subset of the Values* $\boldsymbol{\hat{\mathfrak{Y}} \subseteq \widehat{\mathbb{Y}^n}}$ *on the Subset* $\boldsymbol{\bar{\mathfrak{x}} \subseteq \overline{\mathbb{X}^n}}$ *and the Parameters G*

For the graph model $\hat{Y}(\bar{X}|\mathfrak{G})^{\Gamma}$, we can define the simulation as the operator

$$\bar{\mathfrak{x}} \xrightarrow{\hat{Y}(\bar{X}|\mathfrak{G})^{\Gamma}} \hat{\mathfrak{Y}} \qquad , (\mathbf{10})$$

where $\bar{\mathfrak{x}} \subseteq \overline{\mathbb{X}^n}$ is the subset of known values of the set of independent variables $\bar{X} \subset V$ and $\hat{\mathfrak{Y}} \subseteq \widehat{\mathbb{Y}^n}$ is the subset of unknown values of dependent variables $\hat{Y} \subset V$.

The simulation can be implemented as a search for the simulation graph $\gamma^{|\mathfrak{G}} \in \boldsymbol{\gamma}^{|\mathfrak{G}} \Leftrightarrow \Gamma^{|\mathfrak{G}}$ for a given initial set $\mathfrak{S}'$ and a set $\mathfrak{G}$. Then, from the set $\mathfrak{S}^{\bar{X}|\mathfrak{G}}$, $\hat{\mathfrak{Y}} \in \hat{\mathfrak{Y}}$ is constructed for each $\bar{\mathfrak{x}} \in \bar{\mathfrak{x}}$ as:

$$\bar{\mathfrak{x}} \xrightarrow{\forall \bar{\mathfrak{x}} \in \bar{\mathfrak{x}} \left( \hat{\mathfrak{y}} \in \hat{\mathfrak{Y}} = \bigcup \mathfrak{S}^{\mathfrak{x}} \left( \gamma \left( \Gamma^{|\mathfrak{G}} | \mathfrak{S}' \right) \right) (\bar{\mathfrak{x}}) \right)} \hat{\mathfrak{Y}}$$

In the simulation problem, we can significantly optimize the search for the graph $\gamma^{|\mathfrak{G}}$. Since the set $\bar{\mathfrak{x}}$ is usually much smaller than $\overline{\mathbb{X}^n}$, we can search or calculate only a part of the substates from $\mathfrak{S}^{\mathfrak{x}}(\gamma^{|\mathfrak{G}})$, which contain all the required keys $\bar{\mathfrak{x}} \in \bar{\mathfrak{x}}$:

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 21 April 2025 doi:10.20944/preprints202504.1608.v1



$$\mathfrak{S}^{\bar{\mathfrak{X}}} = \{\mathfrak{S}^{\mathfrak{X}}{}_q \in \mathfrak{S}^{\mathfrak{X}}(\gamma^{|\mathfrak{G}}) \mid \mathfrak{X}(\mathfrak{S}^{\mathfrak{X}}{}_q) \in \bar{\mathfrak{X}}\} \quad ,(11)$$

We can also optimize $\mathfrak{S}'$ by including substates that are as close as possible (from the point of view of the distance in graph $\Gamma^{|\mathfrak{G}}$) to substates from the desired $\mathfrak{S}^{\bar{\mathfrak{X}}}$ or even equivalent to these substates. This will reduce the number of calculations not related to the search for $\mathfrak{S}^{\bar{\mathfrak{X}}}$ (see **Figure 4**).

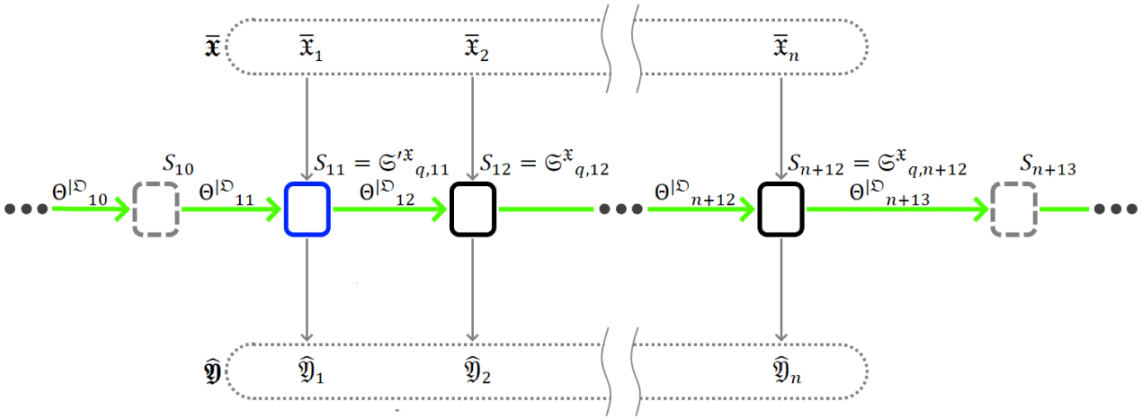



**Figure 4.** Reduction of the number of calculations by including substates that are as close as possible to those from the desired $\mathfrak{S}^{\bar{\mathfrak{X}}}$.

For the graphical model $\hat{Y}(\bar{X}|\mathfrak{G})^{\Gamma}$, an interactive simulation can also be performed. In the simplest case, this requires many models $\{\hat{Y}(\bar{X}|\mathfrak{G})^{\Gamma}{}_i\}$; however, a more interesting and optimal approach is to undertake interactive manipulation of the values of the nodes $S \in S(\Gamma^{|\mathfrak{G}})$ when imperative representations (sequential calculation of the functions $\Theta^{|\mathfrak{P}}$) of the operation $\gamma(\Gamma^{|\mathfrak{G}}|\mathfrak{S}')$ are used. This approach was explored briefly in. [23]

Two patterns are possible here:

- *Push pattern:*
  *This pattern can help synchronize the simulation with some external processes (for example, to synchronize with real-time). The essence of the pattern is that some function $\Theta^{|\mathfrak{D}}$ cannot be calculated until all its arguments $S$ are defined; thus, we can locally pause the simulation, leaving some of the root nodes $S_{root} \in S(\Gamma^{|\mathfrak{G}})$ uninitialized. We can then continue it by defining these nodes.*
- *Pool pattern:*
  *This pattern can be used to implement an asynchronous simulation reaction to some external events — for example, to respond to user input. As in the previous case, some $S_{root} \in S(\Gamma^{|\mathfrak{G}})$ remain uninitialized. However, the simulation does not stop there. Their values are constructed as needed to calculate the next $\Theta^{|\mathfrak{D}}$. Using this approach, it is figuratively possible to imagine that undefined $S_{root}$ is computed by some set of unknown transition functions, possibly also combined into a transition graph. In other words, there is some "shadow" or "unknown" part of the graph $\Gamma^{|\mathfrak{G}}$ and, as a result of its calculation, the $S_{root}$ is initialized (see **Figure 5**).*

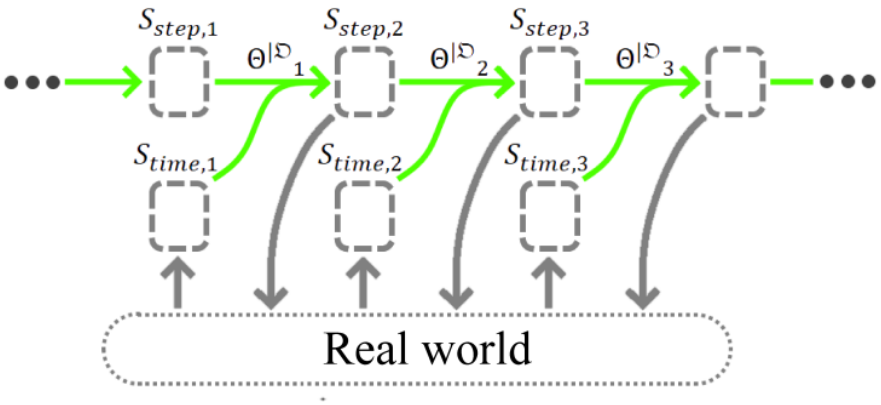



**Figure 5.** Representing the input/output as a set of unknown transition functions.

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 21 April 2025 doi:10.20944/preprints202504.1608.v1



## 4. Logical Processors

We show how the graph model $\hat{Y}(\bar{X}|\mathfrak{G})^{\Gamma}$ can be implemented using the reactive-streams paradigm in the form of a computational graph. We also show how the simulation can be evaluated on this graph and offer ways to optimize it. Additionally, in **Appendix H**, we implement a simple computational graph and perform a simulation with it.

### *4.1. Reactive Streams and Graph Model* $\hat{Y}(\bar{X}|\mathfrak{G})^{\Gamma}$

The concept of reactive streams was formulated in 2014 in manifest [10–12] and extended by AKKA library developers with tools for composing reactive streams into **computational graphs,** [13,24] which are already widely used in practice. [9,25–27] The graph nodes are logical processors, and the edges are the channels representing the stream of **messages** that transmit data; each of the processors transforms the messages in some way. Generally, reactive streams are an implementation of the well-known dataflow-programming paradigm.[25]

This chapter will follow the approach described in (i.e., we will compose a computational system from small blocks that process data streams) [22,28,29]. However, we will use reactive streams to do all the hard work for us to distribute computation and load balancing.

We will denote messages (values) by $M$, logical processors (reactors) by $LP$, channels connecting the processors by $D$, and the numeral graph by $C$.

We can transform an arbitrary graphical model $\hat{Y}(\bar{X}|\mathfrak{G})^{\Gamma}$ (**formula 9**) into a computational graph $C$:

- *To represent each substate* $\mathfrak{S}^{\mathfrak{X}}{}_{q} \in \mathfrak{S}^{\mathfrak{X}}\left(\Gamma^{|\mathfrak{G}}\right)$ *with the message* $M = \left[\mathfrak{S}^{\mathfrak{X}}{}_{q}\right]$.
- *To replace all* $\Theta^{|\mathfrak{P}} \in \Theta\left(\Gamma^{|\mathfrak{G}}\right)$ *for which* $S'_{k=1}, \ldots, S'_{k=n} \in S(\mathfrak{S}')$ *with equivalent processors* $LP^{eval}$*:*

$$S'_{k=1} \Rightarrow D_{k=1}, \ \ldots, S'_{k=n} \Rightarrow$$
$$\Rightarrow D_{k=n} \xrightarrow{\Theta^{|\mathfrak{P}} \Rightarrow LP^{eval}} \grave{S} \Rightarrow D$$

*and all* $\Theta^{|\mathfrak{P}}$ *(for which* $\grave{S} \in S(\mathfrak{S}')$*) with processors* $LP^{eval}$ *equivalent to the inverse functions* $\Theta^{-1|\mathfrak{P}}$*:*

$$\grave{S}' \Rightarrow D \xrightarrow{\Theta^{|\mathfrak{P}} \Rightarrow \Theta^{-1|\mathfrak{P}} \Rightarrow LP^{eval}} S_{k=1} \Rightarrow$$
$$\Rightarrow D_{k=1}, \ \ldots, S_{k=n} \Rightarrow D_{k=n}$$

- *To successively replace all functions* $\Theta^{|\mathfrak{P}} \in \Theta\left(\Gamma^{|\mathfrak{G}}\right)$ *and all arguments* $S_{k=1}, \ldots, S_{k=n}$ *that are already replaced by the channels* $D_{k=1}, \ldots, D_{k=n}$*, which are equivalent to* $LP^{eval}$*:*

$$D_{k=1}, \ \ldots, D_{k=n} \xrightarrow{\Theta^{|\mathfrak{P}} \Rightarrow LP^{eval}} \grave{S} \Rightarrow D$$

- *And to successively replace all* $\Theta^{|\mathfrak{P}}$*, the result* $\grave{S}$ *of which has already been replaced by channel* $D$*, by* $LP^{eval}$ *that equivalent to the inverse functions* $\Theta^{-1|\mathfrak{P}}$*:*

$$D \xrightarrow{\Theta^{|\mathfrak{P}} \Rightarrow \Theta^{-1|\mathfrak{P}} \Rightarrow LP^{eval}} S_{k=1} \Rightarrow$$
$$\Rightarrow D_{k=1}, \ \ldots, S_{k=n} \Rightarrow D_{k=n}$$

As a result, we obtain a graph $C$ containing the equivalent $LP^{eval}$ for each $\Theta^{|\mathfrak{P}} \in \Theta\left(\Gamma^{|\mathfrak{G}}\right)$ but possibly differing structures compared to $\Gamma^{|\mathfrak{G}}$, since its construction was carried out starting from $S' \in S(\mathfrak{S}')$ rather than from the root nodes $S_{root} \in S\left(\Gamma^{|\mathfrak{G}}\right)$ (see **Figure 6**).

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 21 April 2025 doi:10.20944/preprints202504.1608.v1



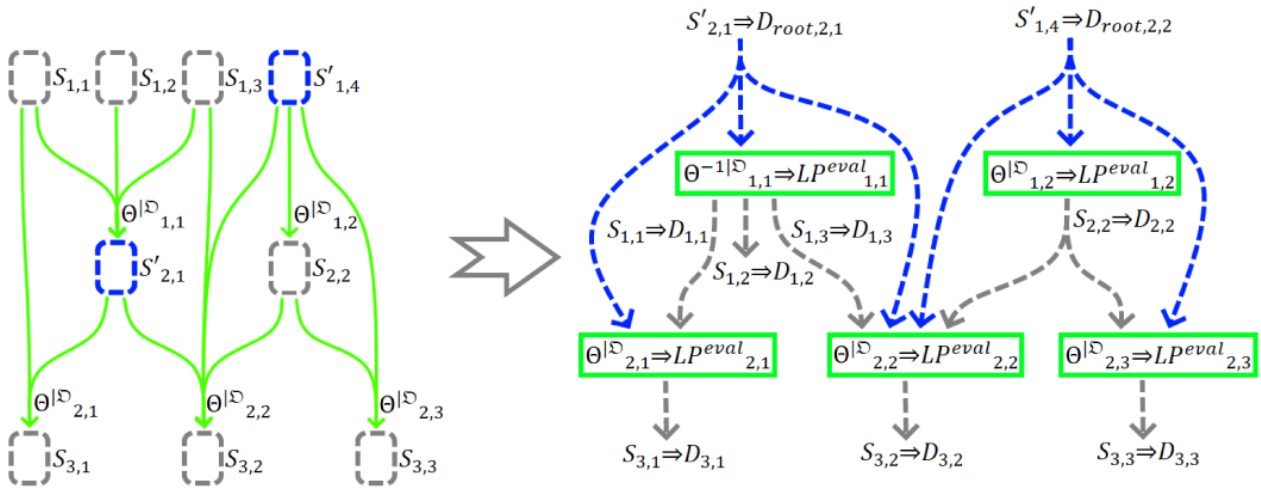



**Figure 6.** Example of constructing of the computational graph $C$ from the model $\hat{Y}(\bar{X}|\mathfrak{G})^{\Gamma}$.

Next, each root channel $D_{root} \in D(C)$ must be connected with a logical processor $LP^{init}$, whose task is to send the corresponding $M' = [\mathfrak{S}'^{\mathfrak{X}}{}_q]$ (where $\mathfrak{S}'^{\mathfrak{X}}{}_q \in \mathfrak{S}^{\mathfrak{X}}(\mathfrak{S}')$), which starts the computational process (see **Figure 7**).

Moreover, all or part of the channels $D \in D(C)$ must be connected with one or more $LP^{collect}$, which will collect part or all of the calculated substates $\mathfrak{S}^{\mathfrak{X}}{}_q \in \mathfrak{S}^{\mathfrak{X}}(\gamma^{|\mathfrak{G}})$ belonging to the graph $\gamma^{|\mathfrak{G}} \in \boldsymbol{\gamma}^{|\mathfrak{G}} \Leftrightarrow \Gamma^{|\mathfrak{G}}$, given by the set of initial states $\mathfrak{S}'$ (see **Figure 7**).

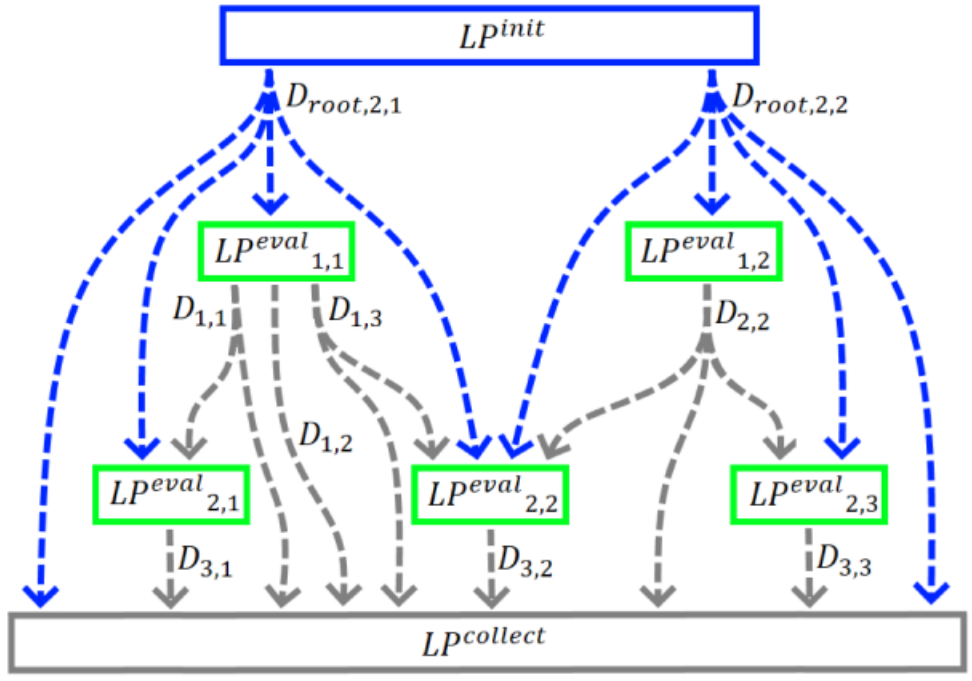



**Figure 7.** Addition of $LP^{init}$ and $LP^{collect}$ into the computational graph $C$.

*4.2. Graph $\boldsymbol{C}$ Optimization*

Simply replacing $\Theta^{|\mathfrak{P}}$ functions with processors $LP^{eval}$ yields an extremely suboptimal and potentially infinite processing graph $C$, which is not good from the viewpoint of minimizing computing resources. To solve this problem, we can optimize graph $C$. For example, consider two optimization methods:

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 21 April 2025 doi:10.20944/preprints202504.1608.v1



- *Folding of cyclic sequences in the graph* $\Gamma^{|\mathfrak{G}}$*:*
  *Consider a chain with an arbitrary length of the same functions* $\theta^{|\mathfrak{P}}$*, as in* ***Figure 8.a****. This can be transformed into a chain of logical processors* $LP^{eval}$ *of equal length, as in* ***Figure 8.b****. We can fold this chain into a single* $LP^{eval}$ *by adding a message-return loop as in* ***Figure 8.c****. Thus, more than one message* $M$ *will go through one* $LP^{eval}$*, so that if* $\theta^{|\mathfrak{P}}$ *has more than one argument, it can lead to collisions. To resolve collisions and also to implement breakage of the loop, we need to determine the loop-iteration number of messages* $M$*. The simple way to do this is to add an iteration counter for each loop in* $C$*. Another approach is to use history-sensitive values.[21] As a more complex example, we consider the graph* $\Gamma^{|\mathfrak{G}}$ *in* ***Figure 9.a****, which can be converted and collapsed into a compact graph* $C$ *as in* ***Figure 9.b****.*
- *Folding of graph* $C$*:*
  *Inside each* $LP^{eval}$*, we can implement more than one function* $\Theta^{|\mathfrak{P}} \in \Theta(\Gamma^{|\mathfrak{G}})$*, thus reducing the number of nodes in the graph* $C$*. This folding can be performed over a wide range, up to the realization of all* $\Gamma^{|\mathfrak{G}}$ *in one* $LP^{eval}$*. For example, graph* $C$ *from* ***Figure 9.b*** *can be folded into single* $LP^{eval}$ *and will look like* ***Figure 9.c****.*

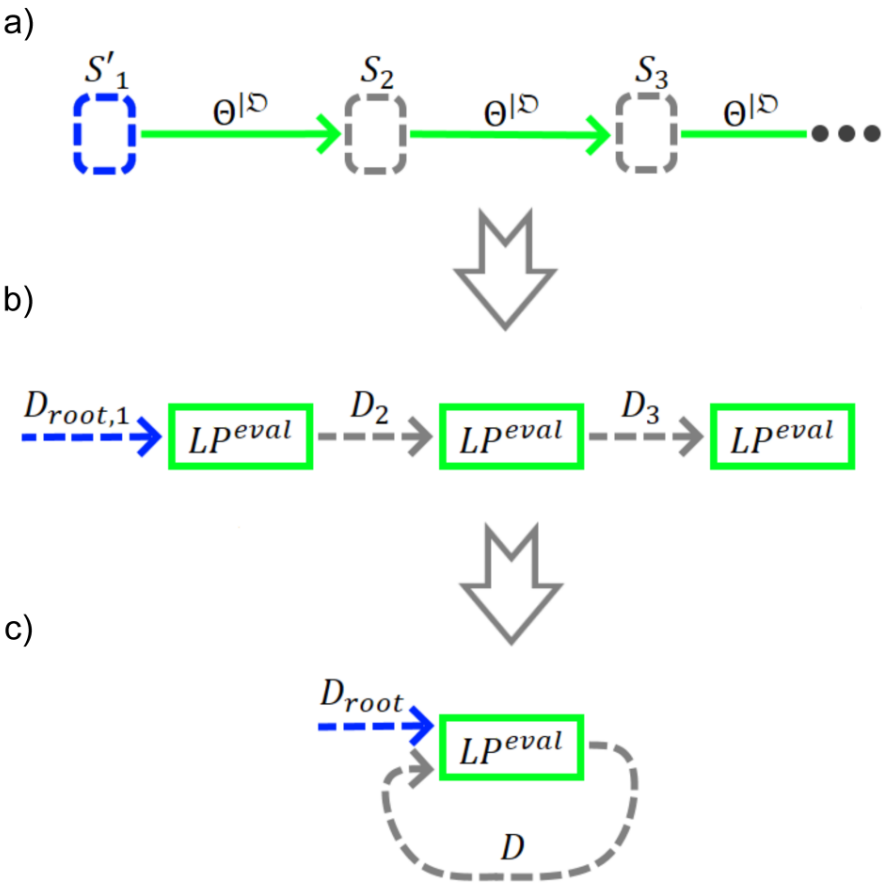



**Figure 8.** Example of the folding of the simple computational graph $C$.

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 21 April 2025 doi:10.20944/preprints202504.1608.v1



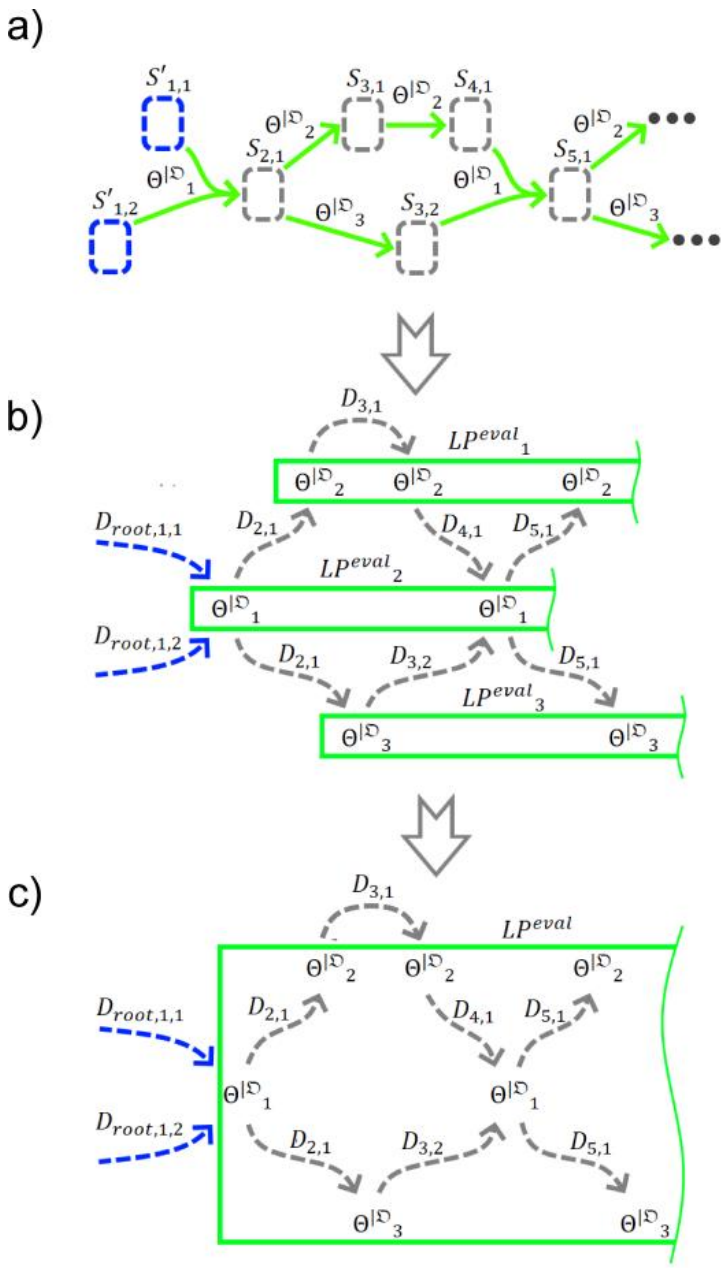



**Figure 9.** Example of the folding of a more complex computational graph $C$.

In general, the optimization problem for graph $C$ is rather complex and goes beyond the scope of this article.

*4.3. Simulation of the Graph Model $\hat{\boldsymbol{Y}}(\bar{\boldsymbol{X}}|\mathfrak{G})^{\Gamma}$ Using the Computational Graph $C$*

In the simplest case, we can simulate the model $\hat{Y}(\bar{X}|\mathfrak{G})^{\Gamma}$ (**formula 10**) using the graph $C$ constructed on it in two stages:

- *Calculate the set of substates*

$$\mathfrak{S}^{\bar{X}|\mathfrak{G}} = \mathfrak{S}^{\mathfrak{X}}\left(\gamma\left(\Gamma^{|\mathfrak{G}}|\mathfrak{S}'\right)\right)$$

*For this, we initialize the calculation by sending $M'$ messages using the processors $LP^{init}$. Using the processors $LP^{collect}$, we collect all the calculated messages, $M = [\mathfrak{S}^{\mathfrak{X}}{}_q]$.*

- *Find all substates for each key $\bar{\mathfrak{X}} \in \bar{\mathfrak{X}}$ and then collect the values $\hat{\mathfrak{Y}} \in \hat{\mathfrak{Y}}$ from the found substates (**formula 6**).*

In most cases, this approach will be computationally expensive, since in practice, usually $|\mathfrak{X}(\gamma^{|\mathfrak{G}})| > |\bar{\mathfrak{X}}|$.

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 21 April 2025 doi:10.20944/preprints202504.1608.v1



Generally, simulation optimization is the minimization of the number of calculated substates $\mathfrak{S}^{\mathfrak{X}}_q$ such that $\mathfrak{X}(\mathfrak{S}^{\mathfrak{X}}_q) \notin \bar{\mathfrak{X}}$. Several approaches are possible here—for example, constructing a minimalistic $\Gamma^{|\mathfrak{G}}$ with a certain well-known collection $\bar{\mathfrak{X}}$. Alternatively, lazy algorithms that cut off the calculations $\Theta^{|\mathfrak{P}}$ whose result is not required to cover $\bar{\mathfrak{X}}$ could be used. However, this topic is beyond the scope of the present article.

## 5. Practice

In this chapter we show our approach in practice. First, we describe modeled object, then define it mathematical model and the analytical solution. In the next step, explain the procedure of construction of graph and parallelization scheme and present the results. This chapter contains shortened description, please check Appendix F, G and H for the full one.

### *5.1. Description of the Modeled Object and the Construction of Model $\hat{\boldsymbol{Y}}(\bar{\boldsymbol{X}}|\mathfrak{G})$:*

As an example, consider the classic model of saline mixing. Here, the simulated object is a system of two connected tanks of volumes $v_1 = 4L$ and $v_2 = 8L$. Over time $t$, a saline solution circulates from the first tank to the second with a speed $q_3 = 5L/m$ and in the opposite direction with a speed of $q_2 = 2L/m$. In addition, the saline solution is poured into the first tank at a speed of $q_1 = 3L/m$ and drains from the second tank at the same speed $q_4 = 3L/m$, i.e., the volume of the saline solution in the tanks does not change. Initially, the first and second tanks are entirely filled with solutions with initial salt concentrations of $\omega_1 = 0g/L$ and $\omega_2 = 20g/L$, respectively. A saline solution with a concentration of $\omega_3 = 10g/L$ is supplied to the first tank constantly. Thus, the set of variables reflecting the properties of interest will look like:

$$V = \begin{bmatrix} t \\ \omega_1 \\ \omega_2 \\ \omega_3 \\ v_1 \\ v_2 \\ q_1 \\ q_2 \\ q_3 \\ q_4 \end{bmatrix}.$$

The modeling task is to predict the change in the salt concentrations $\omega_1 = 0g/L$ and $\omega_2 = 20g/L$ over time $t$.

As part of the modeling problem to be solved, we represent the simulated object in the form of the model $\hat{Y}(\bar{X}|\mathfrak{G})^F$ (**formula 3**), breaking the variables $V$ as

$$\bar{X} = [t],$$
$$\hat{Y} = \begin{bmatrix} \omega_1 \\ \omega_2 \end{bmatrix},$$
$$\mathfrak{G} = \begin{bmatrix} v_1 = 4 \\ v_2 = 8 \\ q_1 = 3 \\ q_2 = 2 \\ q_3 = 5 \\ q_4 = 3 \\ \omega_3 = 10 \end{bmatrix}$$

and specifying their dependence as a set of functions:

$$\hat{Y} = F(\bar{X}|\mathfrak{G}) = \begin{bmatrix} \hat{\omega}_1(\bar{t}) = \frac{13e^{\frac{(\sqrt{105}-15)\bar{t}}{16}}\sqrt{105}}{21} - \frac{13e^{-\frac{(15+\sqrt{105})\bar{t}}{16}}\sqrt{105}}{21} - 5e^{\frac{(\sqrt{105}-15)\bar{t}}{16}} - 5e^{-\frac{(15+\sqrt{105})\bar{t}}{16}} + 10 \\ \hat{\omega}_2(\bar{t}) = \frac{5e^{-\frac{(15+\sqrt{105})\bar{t}}{16}}\sqrt{105}}{21} - \frac{5e^{\frac{(\sqrt{105}-15)\bar{t}}{16}}\sqrt{105}}{21} + 5e^{\frac{(\sqrt{105}-15)\bar{t}}{16}} + 5e^{-\frac{(15+\sqrt{105})\bar{t}}{16}} + 10 \end{bmatrix}, \textbf{(12)}$$

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 21 April 2025 doi:10.20944/preprints202504.1608.v1



Which are obtained by solving of the Cauchy problem

$$\begin{cases} \dfrac{d\hat{\omega}_1}{d\bar{t}} = \dfrac{3 * 10 + 2 * \hat{\omega}_2 - 5 * \hat{\omega}_1}{4} \\ \hat{\omega}_1(0) = 0 \\ \dfrac{d\hat{\omega}_2}{d\bar{t}} = \dfrac{5 * \hat{\omega}_1 - 2 * \hat{\omega}_2 - 3 * \hat{\omega}_2}{8} \\ \hat{\omega}_2(0) = 20 \end{cases}.$$

We can also represent the simulated object in the form of model $\hat{Y}(\bar{X}|\mathfrak{G})^S$ (**formula 4**). In this case, the values of the variable $t$ will be used as keys and those of the variables $\omega_1$ and $\omega_2$ can be separated by different substates, such that we obtain two types of substates $\mathfrak{S}^{\mathfrak{x}}{}_{q=1} = [\omega_1]^{[t]}{}_{q=1}$ and $\mathfrak{S}^{\mathfrak{x}}{}_{q=2} = [\omega_2]^{[t]}{}_{q=2}$. In the code, we can represent the values $\bar{\mathfrak{x}}$, $\hat{\mathfrak{y}}$ and the substate $\mathfrak{S}^{\mathfrak{x}}{}_q$ as OOP classes (source code B.1.L27).

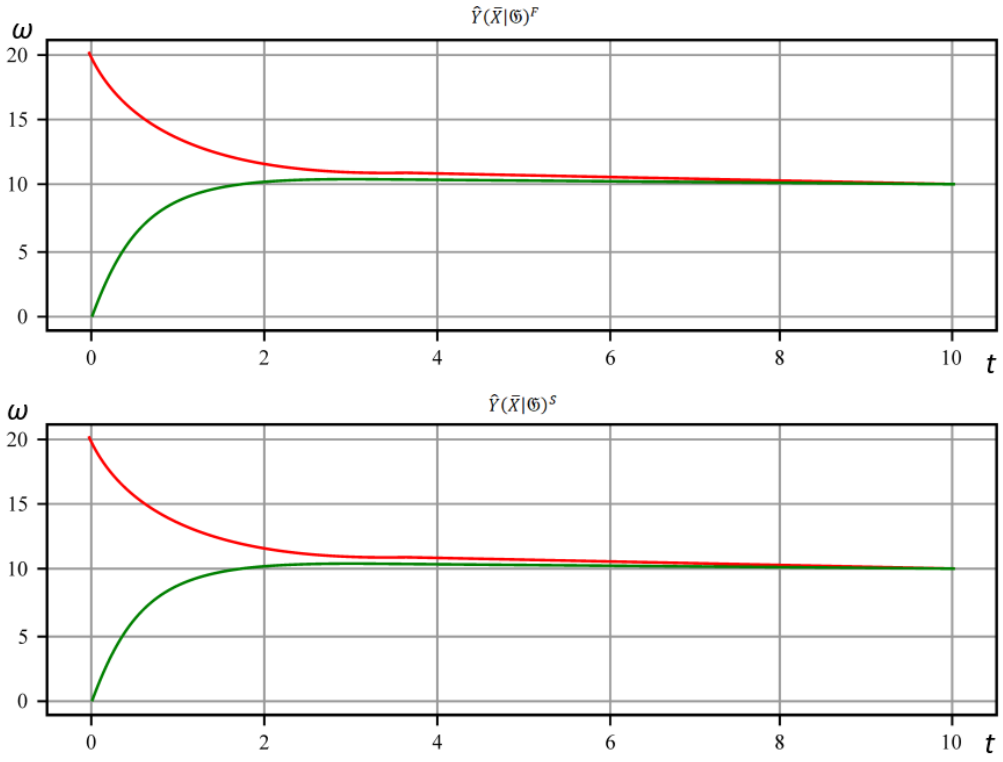
One simple, but impractical, way to construct a set of substates $\mathfrak{S}^{\bar{X}|\mathfrak{G}}$ is to generate $\mathfrak{S}^{\mathfrak{x}}{}_{q=1}, \mathfrak{S}^{\mathfrak{x}}{}_{q=2} \in \mathfrak{S}^{\bar{X}|\mathfrak{G}}$ using a set of functions (**formula 12**) with some step of key $\Delta t$ (source code B.2.L60).

Using the model $\hat{Y}(\bar{X}|\mathfrak{G})$, we can perform the simulation (**formula 5**) for some segment $\bar{\mathfrak{x}} = [t_{begin}, t_{end}]$ and obtain the corresponding set of values $\hat{\mathfrak{y}}$ (source code B.1.L71 is an implementation of $\hat{Y} = F(\bar{X}|\mathfrak{G})$ and the source code B.2.L70 is an implementation of $\mathfrak{S}^{\bar{X}|\mathfrak{G}} = \hat{Y}$). Looking at the output plots, we can see that they are similar (see **Figure 10**).

We can compare the results of executing of models $\hat{Y}(\bar{X}|\mathfrak{G})^F$ and $\hat{Y}(\bar{X}|\mathfrak{G})^S$ just by accumulating different overall output values:

$$\varepsilon = \sum_{\bar{\mathfrak{x}} \in \bar{\mathbb{X}}^n} \sum_{i=1}^{|Y|} \left( \hat{Y}(\bar{X}|\mathfrak{G})^F{}_i - \hat{Y}(\bar{X}|\mathfrak{G})^S{}_i \right).$$

Evaluating this algorithm (source B.3.L24), we obtain $\varepsilon = 1.1546319456101628^{e-14}$.




*X axis is time and Y axis is salt concentration, green line $\omega_1$ is salt concentration in tank 1 and red line $\omega_2$ is salt concentration in tank 2.*

**Figure 10.** Results of a simulation of the $\hat{Y}(\bar{X}|\mathfrak{G})^F$ (first plot) and $\hat{Y}(\bar{X}|\mathfrak{G})^S$ (second plot) models. Where X axis is time and Y axis is salt concentration, green line $\omega_1$ is salt concentration in tank 1 and red line $\omega_2$ is salt concentration in tank 2.

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 21 April 2025 doi:10.20944/preprints202504.1608.v1



*5.2. Building and Simulating a Graphical Model* $\widehat{\boldsymbol{Y}}(\bar{\boldsymbol{X}}|\mathfrak{G})^{\Gamma}$*:*

The graph $\Gamma^{|\mathfrak{G}}$ for this example will represent an infinite chain of pairs of nodes S connected by edges $\Theta^{|\mathfrak{D}}$. For convenience, in addition to the index of depth $d$, we to index the nodes S with indices of width $w \in N$, such that the nodes $S_d$, with the same index $d$, will have different values of $w$. Moreover, we set w = k = q, where k is the index of the argument (edges) ${\Theta^{|\mathfrak{D}}}_d$ and q is the index of the substate assigned to $S_{d,w}$. Each pair $S_{d,w=1}$ and $S_{d,w=2}$ corresponds to a certain moment of discrete-time $\bar{t}$. For simplicity, we will use a fixed time-step $\Delta t = d * \gamma$, where $d$ is the depth index and $\gamma$ is the time-step coefficient. Also, we restrict model time to a small interval $[t_{begin}, t_{end}] \supset \bar{t}$. In this case, the graph $\Gamma^{|\mathfrak{G}}$ will contain

$$\mathrm{n} = \frac{t_{end} - t_{begin}}{\Delta t} + 1$$

pairs of nodes $S_{d,w}$.

The simplest way to implement the transition functions ${\Theta^{|\mathfrak{D}}}_{d,w=1}$ and ${\Theta^{|\mathfrak{D}}}_{d,w=2}$ is to use the functions $\widehat{\omega}_1(\bar{t})$ and $\widehat{\omega}_2(\bar{t})$ from the set $F(\bar{X}|\mathfrak{G})$ (**formula 13**). In this case,

$$\begin{gathered}{\Theta^{|\mathfrak{D}}}_{d,w=1}\left([\widehat{\omega}_1]^{[\bar{t}]}{}_{k=1}, [\widehat{\omega}_2]^{[\bar{t}]}{}_{k=2}\right) = \\ = [\widehat{\omega}_1(\bar{t} + \Delta t)]^{[\bar{t}+\Delta t]}{}_{q=1}; \\ {\Theta^{|\mathfrak{D}}}_{d,w=2}\left([\widehat{\omega}_1]^{[\bar{t}]}{}_{k=1}, [\widehat{\omega}_2]^{[\bar{t}]}{}_{k=2}\right) = \\ = [\widehat{\omega}_2(\bar{t} + \Delta t)]^{[\bar{t}+\Delta t]}{}_{q=2}.\end{gathered}$$

A slightly more complicated implementation is to rewrite the system of differential equations (**formula 14**) to be solved by the Euler method

$$\begin{cases}\widehat{\omega}_{1,i} = \widehat{\omega}_{1,i-1} + \Delta t * \dfrac{q_1 * \omega_3 + q_2 * \widehat{\omega}_{2,i-1} - q_3 * \widehat{\omega}_{1,i-1}}{v_1} \\ \widehat{\omega}_{2,i} = \widehat{\omega}_{2,i-1} + \Delta t * \dfrac{q_3 * \widehat{\omega}_{1,i-1} - q_2 * \widehat{\omega}_{2,i-1} - q_4 * \widehat{\omega}_{2,i-1}}{v_2}\end{cases}$$

as iterated by $\Delta t$:

$$\widehat{\omega}_{1,0} = 0;\ \widehat{\omega}_{2,0} = 20; i = 1,2,3, \ldots$$

In this case

$${\Theta^{|\mathfrak{D}}}_{d,w=1}\left([\widehat{\omega}_1]^{[\bar{t}]}{}_{k=1}, [\widehat{\omega}_2]^{[\bar{t}]}{}_{k=2}\right) = \left[\widehat{\omega}_1 + \Delta t * \frac{\mathfrak{D}.q_1 * \mathfrak{D}.\omega_3 + \mathfrak{D}.q_2 * \widehat{\omega}_2 - \mathfrak{D}.q_3 * \widehat{\omega}_1}{\mathfrak{D}.v_1}\right]^{[\bar{t}+\Delta t]}{}_{q=1};$$

$${\Theta^{|\mathfrak{D}}}_{d,w=2}\left([\widehat{\omega}_1]^{[\bar{t}]}{}_{k=1}, [\widehat{\omega}_2]^{[\bar{t}]}{}_{k=2}\right) = \left[\widehat{\omega}_2 + \Delta t * \frac{\mathfrak{D}.q_3 * \widehat{\omega}_1 - \mathfrak{D}.q_2 * \widehat{\omega}_2 - \mathfrak{D}.q_4 * \widehat{\omega}_2}{\mathfrak{D}.v_2}\right]^{[\bar{t}+\Delta t]}{}_{q=2}.$$

We implement the nodes S and the sets of edges $\Theta^{|\mathfrak{P}}$ as OOP classes (source code C.1.L89). S nodes are essentially variables that are not initially defined. The transition graph $\Gamma^{|\mathfrak{G}}$ and the simulation graph $\gamma^{|\mathfrak{G}}$ can be represented as classes containing collections of nodes S of sets of edges $\Theta^{|\mathfrak{P}}$ (source code C.1.L149). Moreover, the graph $\gamma^{|\mathfrak{G}}$ is the same as graph $\Gamma^{|\mathfrak{G}}$, but with all variables S defined.

Due to the simplicity of the transition graph $\Gamma^{|\mathfrak{G}}$, we can implement the function build_Γ($n$, $\Delta t$), which automatically constructs $\Gamma^{|\mathfrak{G}}$ based on the given number of steps and the time-step (source code C.1.L190).

The search for the simulation graph $\gamma\left(\Gamma^{|\mathfrak{G}}|\mathfrak{S}'\right)$ is a calculation of the values of all nodes S from the initial set of substates

$$\mathfrak{S}' = \left\{\mathrm{S}_{d=0,w=1} = \mathfrak{S}^{\mathfrak{X}}{}_{j=1},\ \mathrm{S}_{d=0,w=1} = \mathfrak{S}^{\mathfrak{X}}{}_{j=2}\right\}.$$

We implement the search as method Γ.γ($\mathfrak{S}'$), using the indices $d$ and $w$ as the key in the set $\mathfrak{S}'$ (source code C.1.L156). The method first initializes the nodes $\mathrm{S}_{d=0,w=1}$ and $\mathrm{S}_{d=0,w=2}$ with the initial substates ${\mathfrak{S}'^{\mathfrak{X}}}_{j=1}$ and ${\mathfrak{S}'^{\mathfrak{X}}}_{j=2}$ and then calculates the values of the rest nodes $\mathrm{S}_{d,w}$ by calling each method ${\Theta^{|\mathfrak{D}}}_{d,w}$.eval() until all $\mathrm{S}_{d,w}$ are defined. The method ${\Theta^{|\mathfrak{D}}}_{d,w}$.eval() checks whether the arguments

$$\mathrm{S}_{d-1,w=1}, \mathrm{S}_{d-1,w=2} \xrightarrow{\Theta^{|\mathfrak{D}}{}_{d,w}} \ldots$$

are defined and, if so, evaluates the result

$$\ldots \xrightarrow{\Theta^{|\mathfrak{D}}{}_{d,w}} \mathrm{S}_{d,w}.$$

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 21 April 2025 doi:10.20944/preprints202504.1608.v1



The set of substates $\mathfrak{S}^{\mathfrak{x}}(\gamma^{|\mathfrak{G}})$ can be obtained from the simulation graph $\gamma(\Gamma^{|\mathfrak{G}}|\mathfrak{S}')$ by simply extracting the values from the nodes $S$ and combining them into the set $\mathfrak{S}^{\bar{X}|\mathfrak{G}}$. We implement this in the form of the method $\gamma^{|\mathfrak{G}}.\mathfrak{S}()$ (source code C.1.L179); next, $\mathfrak{S}^{\bar{X}|\mathfrak{G}}$ can be used to obtain the values of $\hat{\mathfrak{Y}} \in \bar{\mathfrak{y}}$ from the values of $\bar{\mathfrak{x}} \in \bar{\mathfrak{X}}$.

### *5.3. Constructing and Calculating Graph $C$ Using the Graphical Model $\hat{Y}(\bar{X}|\mathfrak{G})^{\Gamma}$*

As an example, we construct graph $C$ using the model $\hat{Y}(\bar{X}|\mathfrak{G})^{\Gamma}$ for mixing salt solutions. To implement it, we use the AKKA Streams library. There was a similar approach to implement the SwiftVis tool.

We can build an unoptimized version of graph $C$ by simply replacing the functions $\Theta^{|\mathfrak{P}}{}_{d,w=1}$ and $\Theta^{|\mathfrak{P}}{}_{d,w=2}$ with logical processors $LP^{eval}{}_{d,w=1}$ and $LP^{eval}{}_{d,w=2}$ and adding $LP^{collect}$, $LP^{init}{}_{d=0,w=1}$, and $LP^{init}{}_{d=0,w=2}$.

We represent the substates in the form of the messages $M_{d,w} = [\mathfrak{S}^{\mathfrak{x}}{}_{q,d,w}]$ produced by the corresponding $LP^{eval}{}_{d,w}$, where $q = w$. In particular, the substates from the set $\mathfrak{S}'$ can be represented as $M_{d=0,w=1} = [S_{d=0,w=1} = \mathfrak{S}'^{\mathfrak{x}}{}_{q=1}]$ and $M_{d=0,w=2} = [S_{d=0,w=2} = \mathfrak{S}'^{\mathfrak{x}}{}_{q=2}]$.

This will work as follows (see source code D.1): the initial messages $M_{d=0,w=1}$ and $M_{d=0,w=2}$ are sent by logical processors $LP^{init}{}_{d=0,w=1}$ and $LP^{init}{}_{d=0,w=2}$ to the processors $LP^{eval}{}_{d=1,w=1}$, $LP^{eval}{}_{d=1,w=2}$. Then, the messages will distribute throughout the graph, where a copy of each substate is fed into the processor $LP^{collect}$, which builds the resulting set of substates $\mathfrak{S}^{\bar{X}|\mathfrak{G}}$.

Since the standard blocks Zip, Flow.map, Broadcast, and Merge from the AKKA Streams library were used to construct graph $C$, the implementation of each $LP^{eval}{}_{d,w}$ will be a nested graph.

Since the obtained graph $C$ consists of recurring pairs $LP^{eval}{}_{d,w=1}$ and $LP^{eval}{}_{d,w=2}$, it can be optimized by implementing two cycles using 2 logical processors $LP^{eval}{}_{w=1}$ and $LP^{eval}{}_{w=2}$.

Since it is necessary in this case to determine which incoming messages refer to particular iterations of the cycle, we add the iteration (depth) counter $d$ to them, $M_{d,w} = [d, \mathfrak{S}^{\mathfrak{x}}{}_{q}]$, and modify the grouping function Zip so that it selects pairs of incoming $M_{d,w}$ with the same value $d$ (source code D.2). When we execute this code, we obtain the simulation result (see **Figure 11**), which was the same as our findings from the implementation of the $\hat{Y}(\bar{X}|\mathfrak{G})^{S}$ model (see **Figure 10**).

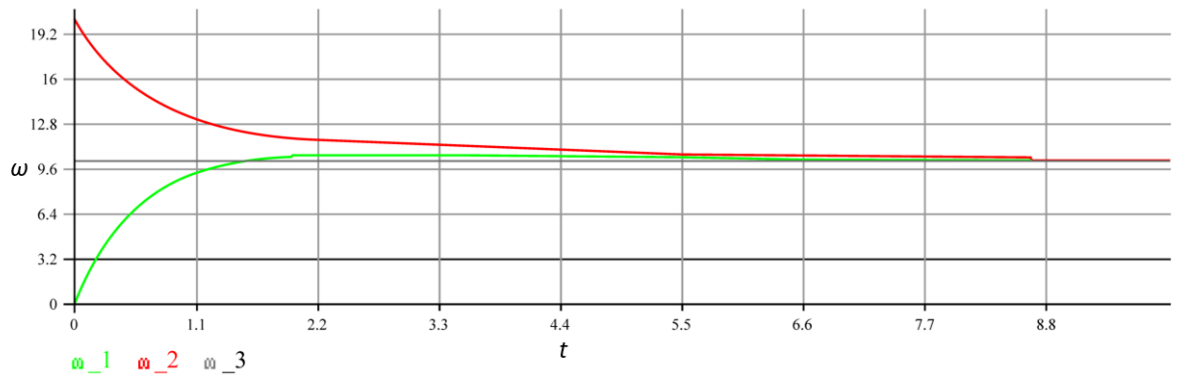



**Figure 11.** Simulation of the $\hat{Y}(\bar{X}|\mathfrak{G})^{\Gamma}$ model using graph $C$. Where X axis is time and Y axis is salt concentration, green line $\omega_1$ is salt concentration in tank 1 and red line $\omega_2$ is salt concentration in tank 2, gray line $\omega_3$ is saline solution concentration supplied to tank 1 constantly.

## 6. Discussion

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 21 April 2025 doi:10.20944/preprints202504.1608.v1



One of the primary contributions of this research is the synthesis of classical mathematical modeling techniques with the practical, high-performance synchronization mechanisms provided by reactive streams. Similar to earlier approaches such as the Time Warp algorithm[1,2] and actor-based frameworks used in HPC simulation platforms[9], our method decomposes the complete object state into substates with unique keys. This modular representation not only supports reuse and flexibility but also enables the direct mapping of transition functions to logical processors. The resulting computational graph is reminiscent of systems such as RxHLA[7] and CQRS + ES architectures[8], which emphasize decoupling and distributed processing.

Representing the model as a transition graph $\Gamma'^{|\mathfrak{G}}$ and initial set of sates $\mathfrak{S}'$ offers several benefits:

- Modularity and Reusability: By encapsulating transition rules as independent functional blocks, the approach supports reuse and flexibility. This modular structure is similar in spirit to block-diagram environments like Simulink [30–32] and has parallels in dataflow programming models discussed by Kuraj and Solar-Lezama [21].
- Scalability: Our implementation leverages the inherent parallelism of modern multi-core and distributed architectures. This approach aligns with the findings of actor-based models [9,25,26] and contemporary research on reactive programming in distributed systems [10,24].
- Interactive Simulation: The push and pool patterns introduced in our model are analogous to techniques used in recent studies on interactive and fault-tolerant reactive systems [23,33]. This design allows the simulation to respond in real time to external events or user inputs.

In summary, the proposed method of using reactive streams as a synchronization protocol for parallel simulation provides a compelling framework that unites rigorous mathematical modeling with practical, scalable implementation techniques. While challenges remain—particularly in optimization, continuous simulation, and fault tolerance, the initial results and conceptual clarity offer a solid foundation for further research and development. The integration of our approach with similar studies in the field [1–10,13,21–28,33,34] highlights its potential and provides clear directions for future work.

## 7. Future Work

Many unanswered questions remain, some of which we present for future research:

- Effective optimization of computational graph $C$ and simulation on it:
  Chapters 4.2 and 4.3 dealt with this topic. However, due to its complexity and vastness, it did not fit into this article. In general, this is a very important issue from a practical point of view. Solving it will significantly reduce the number of resources required to perform simulations. Another interesting question is the automation of the optimization of graph $C$. Say that, initially, we have non-optimal $C$, for example, obtained by the method described in Chapter 4.1. We want to automatically make $C$ compact and computationally easy, without loss of accuracy and consistency.
  To resolve the optimization task the ML technique can be used. For example, reinforcement rearming agents can be trained to explore various graph configurations (i.e., different ways to fold or collapse the computational graph) and learn which configurations yield the best performance in terms of latency, throughput, or resource consumption [36–40]. Also, techniques like neural architecture search (NAS) can be adapted to optimize the layout and parameters of the computational graph. This includes automatically deciding how to fold cyclic sequences, balance load among logical processors, and minimizing redundant computations [39–41].
- Accurate simulation of continuous-valued models:
  Many properties of modeled objects can be represented in continuous quantities, for example, values from the set $\mathbb{R}$. However, the simulation (the calculation of which is based on message forwarding) is inherently discrete. An open question remains as to how accurately continuous quantities can be calculated. The question is how to increase the accuracy of calculating such quantities without increasing the requirements for computer resources.

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 21 April 2025 doi:10.20944/preprints202504.1608.v1



- Fault-tolerance of reactive streams:
  We did not touch on fault tolerance of simulation in this work, but in most real/practical applications, fault tolerance is very important. This question was partially explored in [23] but we also suggest this for future work.
- Manual and automatic graph construction $C$:
  From a practical viewpoint, it is interesting to be able to use some IDE to manually construct a computational graph $C$, and to do this such that the corresponding graph $\Gamma^{|\mathfrak{G}}$ will be consistent and optimal. For example, this might be done similarly to the Simulink package, [30,42] SwiftVis tool, [25,43] or XFRP language [24,44]. It is also interesting to find ways to automate the construction of $C$. For example, the model can initially be defined as a certain set of rules by which graph $C$ can be automatically and even dynamically constructed. Specialized programming languages are also an interesting area to explore. For example, the EdgeC [33,35] language can be considered a tool to describe computational graphs.
  Also, ML technique can be applied wildly here. For example, graph learning techniques from graph neural networks (GNNs) can be applied to learn the structure of the optimal computational graph from historical data. The learned model can then suggest or automatically construct a more efficient graph based on current simulation requirements. Adaptive scheduling ML algorithms can dynamically adjust the scheduling of tasks across logical processors, optimizing the execution order and balancing the load [45,46]. This is particularly useful in interactive or real-time simulations where conditions may change frequently.
- Testing with complex models and comparing with other parallelizing approaches:
  This work provides a small, simple example of parallel simulation to show how the described approach can be implemented in practice. However, the questions of checking this approach with large and complex models and comparing its effectiveness with other parallelizing approaches remain open.

## Conclusions

The proposed method effectively integrates the reactive streams paradigm with classical mathematical modeling techniques to create a scalable framework for parallel simulation. By using a graph-based representation of object states and transition functions, this approach enhances modularity and reusability while supporting efficient computation through logical processors. The implementation using AKKA reactive streams demonstrates its scalability and practical feasibility for distributed systems. Despite its promise, the work highlights challenges such as graph optimization, continuous model simulation, fault tolerance, and automation of graph construction, which offer significant areas for future research and development. The study lays a strong foundation for advancing parallel simulation techniques, emphasizing both theoretical robustness and practical scalability.

**Supplementary Materials:** The following supporting information can be downloaded at: Preprints.Org.

**Author Contributions:** O.S.: Conceptualization, methodology, formal analysis, writing, visualization, software. A.P.: Conceptualization, methodology, formal analysis, writing—original draft preparation, and visualization. I.P.: Methodology, software, formal analysis, data curation, and writing—original draft preparation. M.Y.: Investigation, resources and editing, and visualization. H.K.: Supervision, data curation, and writing—review and editing. V.A.: Supervision, data curation, visualization, and writing—review and editing. All authors have read and agreed to the published version of the manuscript.

**Funding:** No funding

**Data Availability Statement:** The raw data supporting the conclusions of this article will be made available by the authors upon request.

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 21 April 2025 doi:10.20944/preprints202504.1608.v1



**Acknowledgments:** The research was conducted as part of the projects 'Development of Methods and Means of Increasing the Efficiency and Resilience of Local Decentralized Electric Power Systems in Ukraine' and 'Development of Distributed Energy in the Context of the Ukrainian Electricity Market Using Digitalization Technologies and Systems', implemented under the state budget program 'Support for Priority Scientific Research and Scientific-Technical (Experimental) Developments of National Importance' (CPCEL 6541230) at the National Academy of Sciences of Ukraine.

**Conflicts of Interest:** The authors declare no conflicts of interest.

## References

1. Jeerson, D. R.; Sowizral, H. A. *Fast concurrent simulation using the time warp mechanism*, Rand Corp: Santa Monica CA, 1982; Part I: Local Control.
2. Richard, R.; M. Fujimoto, *Parallel and Distributed Simulation Systems*, Wiley: New York, USA, 2000.
3. Radhakrishnan, R.; Martin, D. E.; Chetlur, M.; Rao, D. M.; Wilsey, P. A. An Object-Oriented, Time Warp Simulation Kernel. in D. Caromel, R. R. Oldehoeft, and M. Tholburn, editors, *In Proc. of the International Symposium on Computing in Object-Oriented Parallel Environments (ISCOPE'98)* **1998**, 1505, 13-23.
4. Jefferson, D.; Beckman, B.; Wieland, F.; Blume, L.; Diloreto, M. Time warp operating system, in *Proc. of the eleventh ACM Symposium on Operating systems principles* **1987**, *21*, 77-93.
5. Aach, J; Church, G.M. Aligning gene expression time series with time warping algorithms, *Bioinformatics* **2001**, Volume 17, Issue 6, Pages 495–508, https://doi.org/10.1093/bioinformatics/17.6.495
6. Nicol, D.M.; Fujimoto, R.M.; Parallel simulation today, *Ann. Oper. Res.*, **1994**, *53*, 249, https://doi.org/10.1007/BF02136831.
7. Falcone, A.; Garro, A. Reactive HLA-based distributed simulation systems with rxhla, *In 2018 IEEE/ACM 22nd International Symposium on Distributed Simulation and Real Time Applications (DS-RT)* **2018**, 1-8.
8. Debski, A.; Szczepanik, B.; Malawski, M.; Spahr, S. In Search for a scalable & reactive architecture of a cloud application: CQRS and event sourcing case study, *IEEE Software (accepted preprint), copyright IEEE(99)*, **2016**.
9. Bujas, J.; Dworak, D.; Turek, W.; Byrski, A. High-performance computing framework with desynchronized information propagation for large-scale simulations, *J. Comp. Sci.* **2019**, *32*, 70-86. https://doi.org/10.1016/j.jocs.2018.09.004
10. Reactive stream initiative, https://www.reactive-streams.org (accessed on 15 April 2025).
11. Davis, A.L. *Reactive Streams in Java: Concurrency with RxJava, Reactor, and Akka Streams*, Apress, 2018.
12. Curasma H.P.; Estrella, J.C. Reactive Software Architectures in IoT: A Literature Review. *In Proceedings of the 2023 International Conference on Research in Adaptive and Convergent Systems (RACS '23)* **2023**, Association for Computing Machinery, New York, NY, USA, Article 25, 1–8. https://doi.org/10.1145/3599957.3606212
13. The implementation of reactive streams in AKKA: https://doc.akka.io/docs/akka/current/stream/stream-introduction.html (accessed on 15 April 2025).
14. Oeyen, B.; De Koster, J.; De Meuter, W. A Graph-Based Formal Semantics of Reactive Programming from First Principles. *In Proceedings of the 24th ACM International Workshop on Formal Techniques for Java-like Programs (FTfJP '22)* **2023**, Association for Computing Machinery, New York, NY, USA, 18–25. https://doi.org/10.1145/3611096.3601
15. Posa, R. *Scala Reactive Programming: Build Scalable, Functional Reactive Microservices with Akka, Play, and Lagom*, Packt Publishing: Germany, 2018
16. Baxter, C. *Mastering Akka*, Packt Publishing: Germany, 2016.
17. Nolte, D.D. The tangled tale of phase space, *Phys. Today*, **2010**, 63, 33-38.
18. Myshkis, A.D.; Classification of applied mathematical models - the main analytical methods of their investigation, *Elements of the Theory of Mathematical Models* **2007**, 9.
19. Briand, L.C.; Wust, J. Modeling development effort in object-oriented systems using design properties. *IEEE Transactions on Software Engineering* 27.11, **2001**, 963-986.
20. Briand, L.C.; Daly, J.W.; Wust. J.K. A unified framework for coupling measurement in object-oriented systems., *IEEE Transactions on software Engineering* 25.1, *1999*, 91-121.

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 21 April 2025 doi:10.20944/preprints202504.1608.v1




21. Shibanai, K.; Watanabe, T. Distributed functional reactive programming on actor-based runtime, *Proc. of the 8th ACM SIGPLAN International Workshop on Programming Based on Actors, Agents, and Decentralized Control*, *2018*, 13 – 22.
22. Lohstroh, M.; Romeo, I.I.; Goens, A.; Derler, P.; Castrillon, J.; Lee, E.A.; Sangiovanni-Vincentelli, A. Reactors: A deterministic model for composable reactive systems, *In Cyber Physical Systems. Model-Based Design*, *Springer*, *Cham*, **2019**, 59-85.
23. Mogk, R.; Baumgärtner, L.; Salvaneschi, G.; Freisleben, B.; Mezini, M. Fault-tolerant distributed reactive programming, *32nd European Conference on Object-Oriented Programming (ECOOP 2018)* 2018.
24. About the graphs in AKKA streams: https://doc.akka.io/docs/akka/2.5/stream/stream-graphs.html (accessed on 15 April 2025).
25. Kurima-Blough, Z.; Lewis, M.C.; Lacher, L.; Modern parallelization for a dataflow programming environment, in *Proc. of the International Conference on Parallel and Distributed Processing Techniques and Applications* (PDPTA), The Steering Committee of the World Congress in Computer Science, Computer Engineering and Applied Computing (WorldComp), **2017**, 101-107.
26. Kirushanth, S.; Kabaso, B. Designing a cloud-native weigh-in-motion, i*n 2019 Open Innovations (OI)*, **2019**.
27. Prymushko, A.; Puchko, I.; Yaroshynskyi, M.; Sinko, D.; Kravtsov, H.; Artemchuk, V. Efficient State Synchronization in Distributed Electrical Grid Systems Using Conflict-Free Replicated Data Types. *IoT* **2025**, 6, 6. https://doi.org/10.3390/iot6010006
28. Oeyen, B.; De Koster, J.; De Meuter, W. Reactive Programming without Functions. *arXiv* **2024,** preprint arXiv:2403.02296.
29. Babaei, M.; Bagherzadeh, M.; Dingel, J. Efficient reordering and replay of execution traces of distributed reactive systems in the context of model-driven development. *Proceedings of the 23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems* **2020**.
30. Simulink https://www.mathworks.com/help/simulink/index.html?s_tid=CRUX_lftnav_(accessed on 15 April 2025).
31. Karris, S.T. Introduction to Simulink with engineering applications. Orchard Publications, *2006*.
32. Dessaint, L.-A.; Al-Haddad, K.; Le-Huy, H.; Sybille, G.; Brunelle P. A power system simulation tool based on Simulink. *IEEE Transactions on Industrial Electronics* **1999**, 46, 6, 1252-1254.
33. Kuraj, I.; Solar-Lezama, A. Aspect-oriented language for reactive distributed applications at the edge, *in Proceedings of the Third ACM International Workshop on Edge Systems, Analytics and Networking* **2020**, (EdgeSys '20). Association for Computing Machinery, New York, NY, USA, 67–72, https://doi.org/10.1145/3378679.3394531.
34. Babbie, E.R. *The practice of social research*, Wadsworth Publishing, 2009, ISBN 0-495-59841-0.
35. Broekhoff, J. Programming Languages For Programs For Stateful Distributed Systems.
36. Zoph, B.; Le, Q. V. Neural Architecture Search with Reinforcement Learning, *arXiv* **2016**, preprint arXiv:1611.01578.
37. Nakata, T.; Chen, S.; Saiki, S.; Nakamura M. Enhancing Personalized Service Development with Virtual Agents and Upcycling Techniques. *Int J Netw Distrib Comput* **2025**, 13, 5, https://doi.org/10.1007/s44227-024-00043-y
38. Liu, M.; Zhang, L.; Chen, J.; Chen W.-A.; Yang, Z.; Lo, L.J.; Wen, J.; O'Neil, Z. Large language models for building energy applications: Opportunities and challenges. *Build. Simul.* **2025**, 18, 225–234, https://doi.org/10.1007/s12273-025-1235-9
39. Kipf, T. N.; Welling, M. Semi-Supervised Classification with Graph Convolutional Networks. *International Conference on Learning Representations (ICLR)* **2017**
40. Nie, M.; Chen, D.; Chen, H.; Wang, D. AutoMTNAS: Automated meta-reinforcement learning on graph tokenization for graph neural architecture search, *Knowledge-Based Systems* **2025**, Volume 310, 113023, https://doi.org/10.1016/j.knosys.2025.113023
41. Kuş, Z.; Aydin, M.; Kiraz, B. Kiraz, A. Neural Architecture Search for biomedical image classification: A comparative study across data modalities, *Artificial Intelligence in Medicine* **2025**, Volume 160, 103064, https://doi.org/10.1016/j.artmed.2024.103064
42. Chaturvedi, D.K. Modeling and simulation of systems using MATLAB and Simulink. CRC press, *2017*

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 21 April 2025 doi:10.20944/preprints202504.1608.v1




43. Lewis, M.C.; Lacher, L.L. Swiftvis2: Plotting with spark using scala. *International Conference on Data Science (ICDATA'18)* **2018**, Vol. 1. No. 1.
44. Yoshitaka, S.; Watanabe, T. Towards a statically scheduled parallel execution of an FRP language for embedded systems. *Proceedings of the 6th ACM SIGPLAN International Workshop on Reactive and Event-Based Languages and Systems* **2019**, 11 – 20, https://doi.org/10.1145/3358503.3361276.
45. Bassen, J.; Balaji, B.; Schaarschmidt, M.; Thille, C.; Painter, J.; Zimmaro, D.; Games, A.; Fast, E.; Mitchell, J.C. Reinforcement learning for the adaptive scheduling of educational activities. *Proceedings of the 2020 CHI conference on human factors in computing systems* **2020**, CHI '20: Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems, 1 – 12, https://doi.org/10.1145/3313831.337651
46. Long, L.N.B.; You, S.-S.; Cuong, T.N.; Kim, H.-S. Optimizing quay crane scheduling using deep reinforcement learning with hybrid metaheuristic algorithm, *Engineering Applications of Artificial Intelligence* **2025**, Volume 143,110021, https://doi.org/10.1016/j.engappai.2025.110021.


**Disclaimer/Publisher's Note:** The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.