



HOKKAIDO UNIVERSITY

Title	A Framework for Proximity-Based Federation of Smart Objects
Author(s)	JULIA, JEREMIE HENRI RAPHAEL
Degree Grantor	北海道大学
Degree Name	博士(情報科学)
Dissertation Number	甲第11972号
Issue Date	2015-09-25
DOI	https://doi.org/10.14943/doctoral.k11972
Doc URL	https://hdl.handle.net/2115/59958
Type	doctoral thesis
File Information	Jeremie_Julia.pdf



A FRAMEWORK FOR PROXIMITY-BASED FEDERATION OF
SMART OBJECTS

A THESIS SUBMITTED FOR THE DEGREE OF DOCTOR OF INFORMATION SCIENCE
IN THE GRADUATE SCHOOL OF INFORMATION SCIENCE AND TECHNOLOGY OF
HOKKAIDO UNIVERSITY

August 2015

Jérémie Julia

Division of Computer Science

Preface

After the mainframe era in the 60s, the personal computer era in the 80s, and the mobility era in the 2000-2010s, the ubiquity era will certainly arrive in the 2020s. Mark Weiser was one of the first who introduced the concept of ubiquitous computing in a seminal paper of 1991 [55]. His vision of ubiquitous computing was the creation of physical environments saturated with computer devices. However, these devices are supposed to be gracefully integrated in the environment, and thus disappear from the eyes of the users. In other words, a ubiquitous computing system should provide services to the user seamlessly with a minimum of distraction.

Today, 24 years after the seminal paper of Mark Weiser, our real world environments are filled with many embedded intelligent devices with communication capabilities. We call these devices smart objects (SOs). They include smart phones, PDAs, tablet PCs, embedded computers, sensor devices, and RFID tags. Most ubiquitous computing applications are limited to the scope of two stereotyped scenarios, i.e., the location-transparent service continuation, and the context-aware service provision. Some researchers think that the potential of the ubiquitous and/or pervasive computing is limited because of the lack of formal models. They also think that any single layer model cannot handle all the concepts needed to understand ubiquitous and/or pervasive computing, and therefore, a hierarchy of models is necessary. Some studies focus on process calculi models. Other studies provide middleware frameworks to help developers to implement solutions. These approaches are not sufficient to describe the interactions among SOs depending on SO location changes during the execution of an application scenario. Moreover, these solutions generally use a centralized control mechanism. It was already proposed that a decentralized solution is required to deal with spontaneous interoperation among mobile devices. Based on this, we believe that to realize the full potentiality of SOs, it is necessary to introduce a framework organised in a hierarchy of formal models. Each formal model in this hierarchy should focus on a specific concept of ubiquitous computing. This framework should allow the SOs to dynamically and autonomously reconstruct

their federation configurations depending on location and context change. Moreover, it should not use a centralized control.

The first major contribution of this thesis is to provide a formal model that allows the smart objects to dynamically and autonomously reconstruct their federation configurations depending on location and context change. No centralized control is used to achieve this. This formal model deals with the following three different levels, where each level is built on top of its previous level:

- At the first level, or at the bottom level, the port-matching model describes the federation and inter-operation mechanism between two SOs.
- At the second level, the graph-rewriting rules describe the dynamic change of federation structures among SOs.
- At the third level, the catalytic-reaction-network model describes complex application scenarios involving mutually related multiple federations.

A catalytic reaction network is a set of catalytic reactions. There are composition reactions and decomposition reactions. A composition reaction recombines a set of input materials into an output material. A decomposition reaction decomposes an input material into a set of output materials. An output material of a catalytic reaction may work as an input material of other reactions, or as a catalyst to enable or disable other reactions. If a material works as a catalyst to enable another reaction, it is called a promoter or a stimulus of this reaction. If it works as a catalyst to disable another reaction, it is called a repressor of this reaction. There are also contexts that can enable reactions. The main difference between a context and a stimulus is that a context is not mobile. Here we will consider only two types of catalytic reactions, i.e., reactions requiring both a context and a stimulus, and reactions requiring only a context. Reactions that can be disabled by a repressor, or reactions that do not require a context are not considered in this thesis.

We use this catalytic reaction network system to model scenarios involving complex federations of SOs where each material denotes an SO or a federation of SOs. These SOs are called application SOs. We present an implementation of our third level using the second level, and prove the validity of this model. This implementation is inspired by a biological RNA replication mechanism. In our framework, this replication mechanism is implemented using the graph-rewriting rules of our second level.

The second major contribution of this thesis is the proof of the validity of the implementation of catalytic-reactions using graph-rewriting rules. It has been shown

that proving properties such as confluence or termination of graph rewriting systems is not possible in general [37, 38]. However, our system only deals with a certain type of graphs. To prove the validity of our graph rewriting system, we needed to formally represent the class of graphs to which our graph rewriting rules are applied. These graphs all share the same type of structures that we called the double-strand structures. Proving the validity of our system for this type of graphs is possible, unlike proving it for the general case.

In ubiquitous computing, we need formal computation models and middlewares to describe and to develop complex applications. Process calculi can model the behaviour of mobile objects in more detail than our model. However one big difference between these models and our model is that they are not focused on the interconnection structure of mobile objects and the dynamical change of this structure.

In the middleware world, researchers try to provide middleware systems to simplify the development of pervasive environments. Conventional middleware systems provide many useful tools and services to develop ubiquitous systems. However, they lack high-level abstract-models necessary to describe complex scenarios. On the other hand, process calculi are too abstract to describe the dynamic interconnection changes among mobiles objects.

To summarize, existing solutions for modelling and implementing ubiquitous computing system either lack high-level-abstract model required to describe complex application scenarios, or do not focus on the interconnection structure of mobile objects and the dynamical change of this structure. Both are required to consider, to describe, and to implement innovative scenarios beyond the current scope of stereotyped applications of ubiquitous, pervasive, and/or mobile computing. Moreover, most of the existing solutions assume centralized control, which limits the flexibility of a system.

We cover the low-level abstractions describing federation mechanisms and interconnection configuration respectively with the first and second level of our formal modelling, i.e., the port matching model and the graph rewriting rule model. The high-level abstraction describing complex scenarios is addressed by the third level of our formal modelling, i.e., the catalytic reaction network model. Moreover, our solution does not assume centralized control.

A system based on our theory has several advantages. It is robust and scalable. Each scenario implemented by such a system is easy to modify. Even if there are several copies of the same context, each federation reaction uses only one of them. Multiple copies of the same context, however, may increase the robustness of the

federation mechanism. Even if one of them may malfunction and get terminated, other contexts will continue to make federations. For scalability, whenever a certain context is overloaded, you may add more copies of this context to immediately increase the concurrent operation capacity. Each scenario is easy to modify because different contexts deal with different federation reactions, and each of them can be added or removed independently from the others.

With this new approach based on the RNA replication mechanism, and with the representation of the interactions among smart objects based on catalytic-reaction-networks, we can consider, describe, and implement innovative scenarios beyond the current scope of stereotyped applications of ubiquitous, pervasive, and/or mobile computing.

Acknowledgments During my stay in the Meme Media Laboratory, I received the support and the help of many people. The first person I would like to express my gratitude to is my supervisor, Professor Yuzuru Tanaka, for his invaluable constructive criticisms and friendly advice. He did not just guide me through the difficulties of my doctor course, but he also encouraged me during moments of doubts and he helped me at critical moments. I am also extremely grateful to my second supervisor, Professor Shin-ichi Minato, for providing me with his assistance, suggestions, and help. I would also like to use this opportunity to thank Randy Goebel, Klaus P. Jantke, and Nicolas Spyrtos, as well as my colleagues and in particular Jonas Sjöbergh for their aspiring guidance and friendship.

Finally, I would like to give special thanks to my parents, my grand-mother, my uncle, and my girlfriend for their long patience, support, and understanding.

Thank you,
Jérémie Julia

Contents

Preface	i
1 Introduction	1
1.1 Background	1
1.2 Motivations	2
1.3 Contributions	3
1.4 Structure of the thesis	7
2 Related Works	9
2.1 Ubiquitous Computing	9
2.2 Graph Rewriting Rules	12
3 Formalization of our Framework	15
3.1 Smart Objects and their port-matching	16
3.2 Graph Rewriting Rules	17
3.2.1 Smart Object Graph (SOG)	17
3.2.2 Graphical Representation of an SOG	20
3.2.3 Definition of the Graph Rewriting Rules	21
3.2.4 Graphical Representation of Graph Rewriting Rules	23
3.2.5 Operations for Triggering and Applying a Rule	23
3.3 Catalytic-Reaction-Network Framework	25
3.3.1 A Reaction in the Catalytic-Reaction-Network Framework	25
3.3.2 Nucleotide Smart Objects	29
3.3.3 Strands of Nucleotide Smart Objects	30
3.3.4 Instance-SOG family of a reaction	31
3.4 Graph Rewriting Rules to Implement Reactions	35
3.4.1 Outline of the process implementing a reaction	35
3.4.2 States of a Nucleotide Smart Object	41

Contents

3.4.3	Graph Rewriting Rules for implementing Catalytic Reactions	41
3.4.4	Error Management Rules	43
3.5	Graph Rewriting rules to build a Context	47
4	Validity Proof of Rules for Reactions	53
4.1	Output-strand formation stage	54
4.2	Proof of the Validity of the Output-Strand Formation	59
4.2.1	Definitions	59
4.2.2	Assumptions “No rule 14” and “Simple Proximity”	60
4.2.3	Lemmas and Theorem	62
4.3	Reinitialization Stage	65
4.4	Proof of the Validity of the Reinitialization Stage	66
4.4.1	Definitions	66
4.4.2	Assumption “No rule 1”	69
4.4.3	Proof of the validity of the reinitialization of the context . . .	69
4.4.4	Proof of the validity of the reinitialization of the material strand for a composition reaction	70
4.4.5	Proof of the validity of the reinitialization of the material strands for a decomposition reaction	71
4.4.6	Proof of the validity of the reinitialization of the stimulus strand	73
4.4.7	Proof of the validity of the reinitialization	74
4.4.8	Confluence during the reinitialization stage	74
4.5	Correctness of the rewriting rules for implementing a Reaction . . .	77
4.6	Correctness of Error Management Rules	77
4.6.1	Broken Connection Management	78
4.6.2	Loss of a B connection for the context	79
4.6.3	Loss of a B connection for the material strands in a composi- tion reaction	80
4.6.4	Loss of a B connection for the material strands in a decompo- sition reaction	82
4.6.5	Improved Correctness proof of the rewriting rules for imple- menting a Reaction	84
4.7	Discussion about the assumptions	84
4.7.1	Assumption “No rule 14”	85
4.7.2	Assumption “No rule 1”	87
4.7.3	Assumption “Simple Proximity”	87
4.8	Implementation	92

4.9 Establishing the Connections among the Application Smart Objects	94
5 Conclusion	101

Contents

List of Figures

1.1	An example catalytic reaction network representing a scenario of smart-object federations.	5
1.2	An example of reaction representation in the third level of our formal modeling (a) and in the second level (b and c).	6
3.1	A graph representing a federation of smart object to illustrate a directed path. The nodes represent the smart objects. The directed edge represent the connections. The label over the edges represent the type of the connections.	19
3.2	The different elements for the graphical representation of SOGs. . .	21
3.3	Some example graph rewriting rules	23
3.4	An example of a composition reaction ρ	29
3.5	An example of a decomposition reaction ρ'	29
3.6	The hierarchy of the six different kinds of NSO types.	30
3.7	An example of material SOG for materials of type (E, F, G) and some of its instances. The states are not represented in the instances. . . .	32
3.8	The initial reaction-SOG and the final reaction-SOG of a composition reaction ρ (Fig. 3.4). The hidden ports are not shown.	36
3.9	The initial reaction-SOG and the final reaction-SOG of a decomposition reaction ρ' (Fig. 3.5). The hidden ports are not shown.	37
3.10	Beginning and ending stages.	39
3.11	Intra-double-strand propagation stage.	39
3.12	Inter-double-strand propagation stage.	40

List of Figures

3.13	Set of Graph Rewriting rules that implement the beginning stage, the intra-double-strand propagation stage, and the inter-double-strand propagation stage of the output-strand formation. When an NSO has no specified type, it means that it is of type T . For readability, the $+B$ ports of the context NSOs and the $-B$ ports of the material NSOs are not shown since they are always hidden.	44
3.14	Set of Graph Rewriting rules that implement the inter-double-strand propagation stage with a separator SS or a separator OS, and ending stage of the output-strand formation. When an NSO has no specified type, it means that it is of type T . For readability, the $+B$ ports of the context NSOs and the $-B$ ports of the material NSOs are not shown since they are always hidden.	45
3.15	Set of Graph Rewriting rules that process the reinitialization. When an NSO has no specified type, it means that it is of type T . For readability, the $+B$ ports of the context NSOs and the $-B$ ports of the material NSOs are not shown since they are always hidden . . .	46
3.16	Set of rules to take care of the loss of B connections	48
3.17	Examples of incompatible strands for a composition reaction.	49
3.18	Set of rules to take care of incompatible material strands. The hidden port are not represented.	50
3.19	Initial configuration of the NSOs.	51
3.20	Set of graph rewriting rules to interactively build a context.	51
4.1	Initialized double-strands of size n	54
4.2	k -th-pair-bridged double-strand of size n	55
4.3	Undocked double-strands of size n	55
4.4	The unbridged reaction-SOGs of the composition reaction ρ (Fig. 3.4) and of the decomposition reaction ρ' (Fig. 3.5). The hidden ports are not shown.	58
4.5	The bridged reaction-SOGs of the composition reaction ρ (Fig. 3.4) and of the decomposition reaction ρ' (Fig. 3.5). The hidden ports are not shown.	61
4.6	Illustration of Lemma 4.2.3.	63
4.7	Illustration of Lemma 4.2.4	64
4.8	Illustration of Lemma 4.4.2. The hidden port are not represented. .	70
4.9	Illustration of Lemma 4.4.7. The hidden port are not represented. .	72

4.10	An illustration of the confluence found in the reinitialization stage of a reaction ρ' (Fig. 3.5).	76
4.11	An illustration of the confluence found in the output-strand formation stage of a reaction ρ' (Fig. 3.5), if Assumption 4.2.1 does not hold. For readability, the hidden ports are not represented.	86
4.12	An illustration of the equivalence if Assumption 4.4.1 holds or not, for a reaction ρ' (Fig. 3.5). For readability, the hidden ports are not represented.	88
4.13	An illustration comparing to different types of system behaviour for a reaction ρ' (Fig. 3.5) depending on whether Assumption 4.2.2 or not. For readability, the hidden ports are not represented.	91
4.14	An illustration of an SOG G that may lead to a deadlock situation. For readability, the hidden ports are not represented.	93
4.15	An illustration of an SOG I that is in a deadlock situation. For readability, the hidden ports are not represented.	93
4.16	Example of program code for a composition reaction ρ . For readability, the hidden port are not shown.	95
4.17	Example of program code for a decomposition reaction ρ' . For readability, the hidden port are not shown.	97
4.18	The architecture of our implementation.	98
4.19	Example of a context build with some Sun SPOTs.	98

List of Figures

List of Tables

3.1	Primitive condition-checking operations	24
3.2	Primitive-action operations	26
3.3	States of a Nucleotide Smart Object	42

List of Tables

Chapter 1

Introduction

1.1 Background

After the mainframe era in the 60s, the personal computer era in the 80s, and the mobility era in the 2000-2010s, the ubiquity era will certainly arrive in the 2020s. Mark Weiser is one of the first who introduced the concept of ubiquitous computing in a seminal paper of 1991 [55]. He started to work on ubiquitous system in 1988 in the research program he created at Xerox PARC [54]. His vision of ubiquitous computing was the creation of physical environment saturated with computer devices. However, these devices are supposed to be gracefully integrated in the environment, and thus disappear to the eyes of the users. In other words, a ubiquitous computing system should provide services seamlessly to the user with a minimum of distraction. The computer devices should be integrated in the daily life of the user compared to traditional personal computers, e.g., desktop computers, laptop computers, and even mobile computers as mobile phone. Indeed, we still see these computers as devices that execute a certain type of software to perform a certain task to fulfil one need. They are not integrated into the daily life of the user in the way that the user needs to enter the virtual environment of the device to fulfil one need, and then leave this environment once this need is fulfilled. Ubiquitous computing presumes a physical environment in which the computer devices are integrated. That mean they should spontaneously support the user when he has a need.

Ubiquitous computing share problems identified, studied, and solved by related fields. The two mains fields are the field of distributed systems and the field of mobile computing [47, 44]. The field of distributed systems revealed many areas that are necessary for ubiquitous computing and are now well defined. These areas include remote communication, fault tolerance, high availability, remote information

access, and security. The field of mobile computing also has many areas that are fundamental for ubiquitous computing. These areas include mobile networking, mobile information access, support for adaptive applications, system-level energy saving techniques, and location sensitivity. However, ubiquitous computing has its own challenges, and the following were identified.

- **Localized scalability:** compared to web service, scalability in a ubiquitous system may be an issue. Scalability in the context of a web service ignores physical distance. Many servers can provide the same web service, independently on whether they are close or far away the users. In a ubiquitous computing, the services provide locally may be able to be scale only locally.
- **Heterogeneity:** different devices use different standard, e.g., different communication protocol.
- **Invisibility:** a ubiquitous computing system should require the minimal human interaction.
- **Context awareness:** ubiquitous computing requires devices and systems that perceive context.
- **Context management:** once a ubiquitous computing system can perceive the current context, it should be able to exploit this perception effectively. For example, a device will have different configuration depending on the context.

1.2 Motivations

Today, 24 years after the seminal paper of Mark Weiser, our real world environments are filled with many embedded intelligent devices with communication capabilities. We call these devices smart objects (SOs). They include smart phones, PDAs, tablet PCs, embedded computers, sensor devices, and RFID tags [51, 18]. Most ubiquitous computing applications are limited to the scope of two stereotyped scenarios, i.e., the location-transparent service continuation, and the context-aware service provision. Some researchers think that the potential of the ubiquitous and/or pervasive computing is limited because of the lack of formal models [27, 15, 7, 40]. They also think that any single layer model cannot handle all the concepts needed to understand ubiquitous and/or pervasive computing, and therefore, a hierarchy of models is necessary [29, 28]. Some studies [26, 25] focus on process calculi models. Other

studies [13, 36] and more recent ones [5, 48, 21, 14, 53, 49] provide middleware frameworks to help developers to implement solutions. These approaches are not sufficient to describe the interactions among SOs depending on SO location changes during the execution of an application scenario. These solutions, except Line, CORTEX and Pico/SeSCo, use a centralized control mechanism. It was already proposed in [39] that a decentralized solution is required to deal with spontaneous interoperation among mobile devices. Based on this, we believe that to realize the full potentiality of SOs, it is necessary to introduce a framework organised in a hierarchy of formal models. Each formal model in this hierarchy should focus on a specific concept of ubiquitous computing. This framework should allow the SOs to dynamically and autonomously reconstruct their federation configurations depending on location and context change. Moreover, it should not use a centralized control.

1.3 Contributions

The first major contribution of this thesis is to provide a formal model [19, 16, 17, 18] based on previous works [50, 51] that allows the smart objects to dynamically and autonomously reconstruct their federation configurations depending on their location and context change. To achieve this, no centralized control is used. This formal model deals with the following three different levels, where each level is built on top of its previous level:

- At the first level, or at the bottom level, the port-matching model describes the federation and inter-operation mechanism between two SOs.
- At the second level, the graph-rewriting rules describe the dynamic change of federation structures among SOs.
- At the third level, the catalytic-reaction-network model describes complex application scenarios involving mutually related multiple federations.

A catalytic reaction network is a set of catalytic reactions. Catalytic reaction network are inspired by collectively autocatalytic set [20]. There are composition reactions and decompositions reactions. A composition reaction recombines a set of input materials into an output material. A decomposition reaction decomposes an input material into a set of output materials. In a catalytic reaction network, the output material of a catalytic reaction may work as the input material of other reactions, or as a catalyst to enable or disable other reactions. If a material works

Chapter 1. Introduction

as a catalyst to enable another reaction, it is called a promoter or a stimulus of this reaction. If it works as a catalyst to disable another reaction, it is called a repressor of this reaction. There is also contexts that can enable reactions. The main difference between a context and a stimulus is that a context is not mobile. Here we will consider only two types of catalytic reactions, i.e., reactions requiring both a context and a stimulus, and reactions requiring only a context. Reactions that can be disabled by a repressor, or reactions that do not require a context are not considered in this thesis. We use this catalytic reaction network system to model scenarios involving complex federations of SOs where each material denotes an SO or a federation of SOs. These SOs are called application SOs.

Fig. 1.1 shows an example scenario with more than one reaction. This scenario is modelled as a catalytic reaction network. Application SOs and reaction contexts are represented by coloured circles including white ones. This scenario describes two different visitor-activity plans for museum visitors. In each of these plans, a visitor first needs to federate his or her PDA P with his or her headphone H to get the federated SO P-H. This is done automatically when the visitor passes through the gate G1. If the visitor does not bring both of these objects, he can borrow one or both at the entrance of the museum. Then, the visitor may have two plans. The first plan is a guided tour. In this plan, the federated SO P-H will be automatically federated with the tablet G of the guide to obtain the federated SO P-H-G. This is done when the visitor passes through the gate G2. To illustrate exhibits, the guide can send audio and text information directly to both the PDA and the headphone of the visitor using his or her favourite language. The second plan is an augmented reality tour. For this tour, the visitor needs to get a special premium ticket that controls the access to this activity plan. When the visitor enters the activity room, he can pick up VR glasses (Virtual Reality glasses) at the entrance. If he has a premium ticket when he passes through the gate G3, this ticket will work as a stimulus to enable the catalytic reaction that will automatically federate his federated SO P-H with the VR glasses. He will become able to control the VR glasses with his or her PDA. In this scenario, all the connections and federations are automatically done while the visitor passes through some gate, and no visitor operation is necessary.

In our model, each material denotes a federation of application SOs and the type of a material is defined by the sequence of the application SO types. For example, in the previous illustration, the federated SO P-H-G is a material that has for type the sequence of SO types (P,H,G). Thus, the material defined by (P,H,G) and the one defined by (H,G,P) are different, even if they use the same SO types, i.e., P, H,

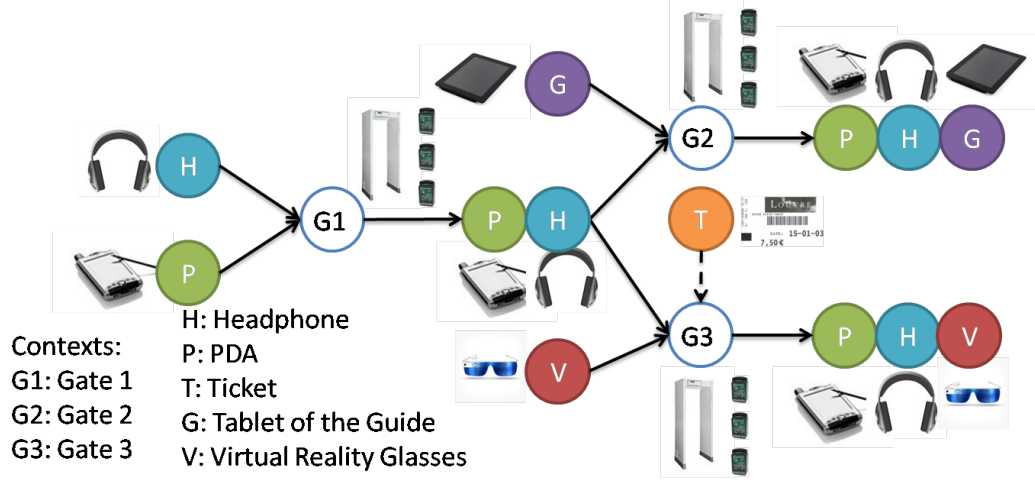


Figure 1.1: An example catalytic reaction network representing a scenario of smart-object federations.

and G. If two materials A and B have the same type, the connection among the SOs composing A and B may be different. For example, if both A and B have the type (P,H,G), the federated SO P-H-G corresponding to A may have a connection from H to G. In B, such a connection from H to G may not exist. However, in this thesis we focus on the sequence of SO types to determine the type of a material, and both A and B are considered to have the same type.

Each catalytic reaction is modelled as a biological RNA-strand replication process. To emulate such a replication process, we introduced nucleotide smart objects (NSOs). NSOs behave like nucleotides in the RNA world [4] hypothesis. We assume that, for each application SO O, an NSO corresponding to the type of O is attached to O as its tag. Such NSOs are called NSO tags. In our second level of our formal model, when application SOs are to be federated together, their NSOs tag form a strand of NSOs. That is why we say that a federation of application SOs is represented by a strand of their tag NSOs. Such a strand representation is described in detail in Section 3.3.3. The context of a reaction is also modelled as a strand of NSOs called context strand. The replication process replicates the structure of the context strand with the NSO tags attached to the application SOs. As the result, these NSO tags are federated into a strand structure, and thus, the application SOs are indirectly federated together through these NSO tags. For example, the representation of a reaction is shown in Fig. 1.2a. This reaction requires a context C, two inputs I1 and I2, and a stimulus S. In our second level of our formal model,

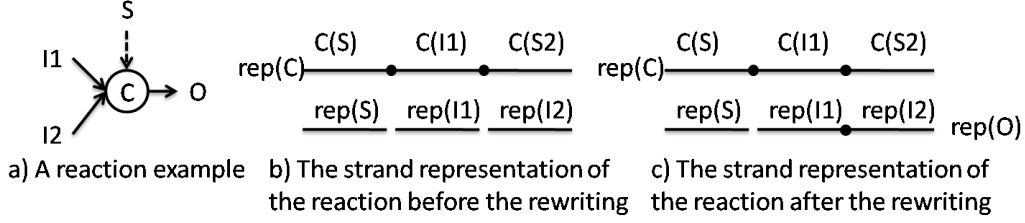


Figure 1.2: An example of reaction representation in the third level of our formal modeling (a) and in the second level (b and c).

S, I1 and I2 are represented by strands of NSOs, i.e., $\text{rep}(S)$, $\text{rep}(I1)$, and $\text{rep}(I2)$, respectively as shown in Fig. 1.2b. The context C is also represented by a strand $\text{rep}(C)$ of NSOs. The strand $\text{rep}(C)$ has three docking parts. Each part corresponds to the strand representation of one of the two inputs or the stimulus, i.e., these three parts C(S), C(I1), and C(I2) respectively correspond to $\text{rep}(S)$, $\text{rep}(I1)$, and $\text{rep}(I2)$. One condition for a catalytic reaction to be triggered is that the NSO tags of the application SOs of this reaction must enter the scope of the context strand NSOs of this reaction. This condition is formally described in Section 3.3.3 by the “proximity between compatible templates or instance strands” definition. If this condition holds true for our example in Fig. 1.2b, then our graph rewriting system will rewrite this graph of NSOs into the graph shown in Fig. 1.2c. After this rewriting, the strand representation of I1 and the strand representation of I2 are connected to form the strand representation of the output material O, i.e., $\text{rep}(O)$.

Such a system based on our theory has several advantages. It is robust and scalable. Each scenario implemented by such a system is easy to modify. Even if there are several copies of the same context, each federation reaction uses only one of them. Multiple copies of the same context, however, may increase the robustness of the federation mechanism. Even if one of them may malfunction and get terminated, the others can continue to make federation. For the scalability, whenever a certain context is overloaded, you may add more copies of this context to immediately increase the concurrent operation capacity. Each scenario is easy to modify because different contexts deal with different federation reactions, and each of them can be added or removed independently from the others.

The second major contribution of this thesis is the proof of the validity of the implementation of catalytic-reactions using graph-rewriting rules. It has been shown that proving properties such as confluence or termination of graph rewriting systems is not possible in general [37, 38]. However, our system only deals with a certain

type of graphs. To prove the validity of our graph rewriting system, we needed to formally represent the class of graphs to which our graph rewriting rules are applied. These graphs all share the same type of structures, as shown in Fig. 1.2, that we called the double-strand structures. Proving the validity of our system for this type of graphs is possible, unlike proving it for the general case.

1.4 Structure of the thesis

In the next chapter, we first present related work and basic concept useful for the understanding of this thesis. In Chapter 3, we present our formal model of smart objects, that includes the formalization of the smart objects, the port-matching model, the graph rewritings rules, and the catalytic-reaction-networks, as well as the set of graph rewriting rules that implements the catalytic-reaction networks. The mechanisms used to build the context of a reaction and to define the connections established by a reaction among the application smart objects are also presented in Chapter 3. We also present the management of the errors that may happen during the reaction process in that chapter. The validity proof of our system is given in Chapter 4. An implementation of our framework is also presented in this chapter as a proof of concept. Finally, in Chapter 5, we present the contribution summary of this thesis.

Chapter 1. Introduction

Chapter 2

Related Works

2.1 Ubiquitous Computing

In this section, we briefly review the history of ubiquitous computing and the common approaches to achieve it. As explain previously, Mark Weiser is one of the first who introduced the concept of ubiquitous computing in a seminal paper of 1991 [55]. He started to work on ubiquitous system in 1988 in the research program he created at Xerox PARC [54]. His vision of ubiquitous computing was the creation of physical environment saturated with computer devices. However, these devices are supposed to be gracefully integrated in the environment, and thus disappear to the eyes of the users. In other words, a ubiquitous computing system should provide services seamlessly to the user with a minimum of distraction. The computer devices should be integrated in the daily life of the user compared to traditional personal computers, e.g., desktop computers, laptop computers, and even mobile computers as mobile phone. Indeed, we still see these computers as devices that execute a certain type of software to perform a certain task to fulfil one need. They are not integrated into the daily life of the user. The user needs to enter the virtual environment of the device to fulfil one need. Once this need is fulfilled, the user leaves this environment. Ubiquitous computing presumes a physical environment in which the computer devices are integrated. That mean they should spontaneously support the user when he has a need, and he will not have to change of environment to fulfill a need.

Ubiquitous computing share problems identified, studied, and solved by related fields. The two mains fields are the field of distributed systems and the field of mobile computing [47, 44]. The field of distributed systems revealed many areas that are necessary for ubiquitous computing and are now well defined. These areas

Chapter 2. Related Works

include remote communication, fault tolerance, high availability, remote information access, and security. The field of mobile computing also has many areas that are fundamental for ubiquitous computing. These areas include mobile networking, mobile information access, support for adaptive applications, system-level energy saving techniques, and location sensitivity. However, ubiquitous computing has its own challenges, and the following were identified.

- **Localized scalability:** compared to web service, scalability in a ubiquitous system may be an issue. Scalability in the context of a web service ignores physical distance. Many servers can provide the same web service, independently on whether they are close or far away the users. In a ubiquitous computing, the services provide locally may be able to be scale only locally.
- **Heterogeneity:** different devices use different standard, e.g., different communication protocol.
- **Invisibility:** a ubiquitous computing system should require the minimal human interaction.
- **Context awareness:** ubiquitous computing requires devices and systems that perceive context.
- **Context management:** once a ubiquitous computing system can perceive the current context, it should be able to exploit this perception effectively. For example, a device will have different configuration depending on the context.

In ubiquitous computing, we need formal computation models and middlewares to describe and to develop complex applications. Some well-known formal computation models are process calculi as π -calculus [25], bigraph [26], and LMNtal[52]. All these are process calculi. Previous process calculi described the behaviour and interconnection of agents or processes. These newer process calculi, i.e., π -calculus, bigraph, and LMNtal try to also describe the behaviour and interconnection of mobile physical computing objects. The π -calculus model formally describes reconfiguration of connectivity¹ but does not describe spatial reconfiguration². On the other hand, process calculus as Ambient calculus [3] models spatial reconfiguration but does not model reconfiguration of connectivity. Bigraph and LMNtal can model both types of reconfiguration.

¹Reconfiguration of connectivity: links move in the virtual space of linked processes.

²Spatial reconfiguration: the processes move in the physical space of computing spaces.

These process calculi can model the behaviour of mobile objects in more detail than ours. However one big difference between these models and our model is that they are not focused on the interconnection structure of mobile objects and the dynamical change of this structure. For example, π -calculus uses channels to model communication between processes. Since π -calculus abstracts away network details, it is not obvious how to use those channels to model connections among SOs. In π -calculus, if two processes know the name of a channel, they can communicate through this channel without any limitation. There is no concept of scope or wireless connection range, no concept of closing a channel, and no concept of network topology or interconnection structure. All these concepts are however necessary to consider for practically modelling P2P interactions among SOs. The π -calculus model has been extended in [30] to consider network architectures. But this work focuses on software-defined networking³ that does not help us to represent the ad-hoc connections among SOs. Other process calculi also lack such concepts.

Example of middleware systems include Linda [13], Lime [36], Gaia [5], Aura [48], Oneworld [14], PICO/SESCO [21], CORTEX [53], and Activity-oriented computing (AoC)[49]. The goal of these middleware systems is to simplify the development of pervasive environments. Linda, Gaia, Aura, Oneworld, and AoC assume centralized control. Lime (that is the decentralized version of Linda), and CORTEX assume decentralized control. Pico/SeSCo used a hybrid approach in which resource-rich devices manage resource-poor devices. It was emphasized in [39] that a decentralized solution is required to deal with spontaneous interoperation among mobile devices. We have tried to provide such a solution in this thesis, namely, the second level of our formal model that deals with the graph rewriting rules.

One advanced feature of some of these middlewares (Gaia, Aura, CORTEX, and AoC) is that they can provide context awareness based on sensing the environment. The goal of context awareness is to collect raw data about the environment, and then to use these data in order to make situation-aware decisions.

In these approaches, federation mechanisms are generally based on the matching of a service-requesting message with a service-providing message. Such a message matching uses either a centralized repository-and-lookup service (e.g. in the case of Linda), or multiple distributed repository-and-lookup service (e.g. in the case of Lime).

These conventional middleware systems provide many useful tools and services

³Software-defined networking: approach that allow an administrator in the context of a computer networking to manage network services through an abstraction of lower level functionality by separating the network control plane from the forwarding plane.

to develop ubiquitous systems. However, they lack a high-level-abstract model necessary to describe complex scenarios. On the other hand, process calculi are too abstract to describe the dynamic interconnection changes among mobile objects. According to Milner [29, 28], there are too many concepts in ubiquitous computing to be covered by a single model. A hierarchy of models is needed. In such a hierarchy, each model should be implemented by a lower model.

2.2 Graph Rewriting Rules

Graph rewriting systems, also called graph grammars, are a natural generalization of formal language theory based on strings and term rewriting theories based on trees. They were first introduced in 1969 in [35], to solve picture processing problems and for pattern recognition. However, many other areas also take advantage of the modelling power that graph rewriting systems offer. They are now not only applied to pattern recognition but also to concurrent system modelling [6], database design [41], programming and visual languages [12], biology [42, 32], software specification and development [10], and many other areas. Graph rewriting systems are widely used, certainly because they are an easy and natural way to model and represent complex systems and/or problems. In this section, we briefly review some famous graph rewriting systems. For a more complete survey, the reader can consult [43].

There are many approaches to graph rewriting systems. Two of the most basic approaches are node replacement and edge replacement. Node replacement graph grammars were first introduced in [31]. In such systems, a node of a graph is replaced by a subgraph. This subgraph is attached to the remainder of the graph by new edges depending on how the replaced node was connected to it. Edge replacement graph grammars were first introduced in [11] and [33]. Similarly, to node replacement graph grammars, in edge replacement graph grammars, an edge of a given graph is replaced by a subgraph. However, the subgraph is not just attached to the remainder of the graph, but it is glued to the remainder by identifying and merging some nodes of the subgraph with some nodes of the remainder.

Another approach to graph rewriting systems is the algebraic approach. In the algebraic approach, a subgraph, rather than a node or an edge, is replaced by a new subgraph. Similarly, to edge replacement graph grammars, the new subgraph is not just attached to the remainder graph, but it is glued to the remainder by identifying and merging some nodes of the subgraph with some nodes of the remainder. There are different graph rewriting systems depending on how the subgraph is glued to

the remainder. Two of the most famous graph rewriting systems from the algebraic approach are the double-pushout approach and the single-pushout approach. The double-pushout approach was first introduced in [9]. The single-pushout approach was first introduced in [23].

Basically, in any graph rewriting system, we consider productions, or rules. Usually a rule is a pair of graphs (L, R) , where graphs L and R are called the left- and the right-hand side of the rule, respectively. Intuitively, applying a rule to a given graph G consists of finding a subgraph of G that is homomorphic to L , and then to replace this subgraph with the graph R to obtain a graph H . What differentiates the double-pushout approach from the single-pushout approach is how to identify and replace the subgraph of G . When we apply a rule r to a graph G in order to obtain a graph H , we also say that we rewrite G into H by applying the rule r , or that we have a direct derivation from G to H , and this direct derivation is denoted by $G \Rightarrow_r H$ or just by $G \Rightarrow H$. A derivation from G to H is a sequence of direct derivations $G_0 \Rightarrow G_1 \Rightarrow G_2 \Rightarrow \dots \Rightarrow G_n$ where $G_0 = G$ and $G_n = H$, and is denoted by $G \Rightarrow^* H$. We may also call a derivation an applicable sequence of rules.

For a graph G , we say that a set of rules P is terminating if there does not exist an infinite sequence of derivations $G \Rightarrow G_1 \Rightarrow G_2 \Rightarrow G_3 \dots$. For a graph G and a set of rules P , if there exists two derivations $G \Rightarrow^* G_1$ and $G \Rightarrow^* G_2$, we say that G is confluent if there exists two derivations $G_1 \Rightarrow^* H_1$ and $G_2 \Rightarrow^* H_2$ such that H_1 is isomorphic to H_2 . If the two derivations are actually two direct derivations $G \Rightarrow G_1$ and $G \Rightarrow G_2$, we say that G is local confluent.

Chapter 2. Related Works

Chapter 3

Formalization of our Framework

In this chapter, we first present in Section 3.1 the formal definition of the smart objects and their proximity federations mechanism based on port-matching, which constitute the basis of our work. A smart object is modelled as a tuple of sets and constants representing its properties. A smart object federation is modelled as an acyclic directed edge-labelled vertex-labelled graph that we call a smart object graph (SOG). SOGs are used

- to define the rules of the second level of our formal model, i.e., the graph-rewriting rules,
- to define the third level of our formal model, i.e., the catalytic-reaction-network model, and
- to prove the validity of our framework.

In Section 3.2, we give the formalization of the second level of our formal model, i.e., the graph-rewriting rules. In Section 3.3, we give the formalization of the third level of our formal model, i.e., the catalytic-reaction-network model. In the same section, we introduce nucleotide smart objects that are the basic elements used to implement catalytic reactions. We also present how to use these nucleotide smart objects to design the type of SOG that we use in our implementation of catalytic reactions. Finally, in Section 3.4, we give a set of graph rewriting rules that are used with this type of SOG to implements catalytic-reactions.

3.1 Smart Objects and their port-matching

In this section, we give the formal definition of the smart objects, their ports, and their connections. We first define a smart object system, which specifies all the sets required to define a smart object.

Definition 1 (Smart Object System). *A smart object system is a septuple*

$$S = (S_\Sigma, S_{Obj}, S_{Po}, S_{PoTyp}, S_{Typ}, S_{Sta}, S_{Rul})$$

where S_Σ is the alphabet of SO types, S_{Obj} is a set of SOs, S_{Po} a set of SO ports, S_{PoTyp} , S_{Typ} and S_{Sta} are a set of strings over S_Σ , and S_{Rul} a set of graph rewriting rules. The set S_{Typ} is a partially ordered set.

Definition 2 (Smart Object). *A smart object O in S_{Obj} is a quintuple*

$$O = (oid_O, type_O, States_O, Ports_O, Rules_O)$$

where $oid_O \in \mathbb{N}$ is its unique identifier, $type_O \in S_{Typ}$ is its type, $States_O \subseteq S_{Sta}$ is its finite set of states, $Ports_O \subseteq S_{Po}$ is its finite set of ports, and $Rules_O \subseteq S_{Rul}$ is its finite set of graph rewriting rules. We also define the following functions:

- $oid : S_{Obj} \rightarrow \mathbb{N}$, such that $oid(O) = oid_O$,
- $type : S_{Obj} \rightarrow S_{Typ}$, such that $type(O) = type_O$,
- $state : S_{Obj} \rightarrow S_{Sta}$, such that $state(O)$ returns the current state of O ,
- $ports : S_{Obj} \rightarrow 2^{S_{Po}}$, such that $ports(O) = Ports_O$.

Definition 3 (Smart Object Port). *A port P in S_{Po} is a pair $P = (poPo_P, poTyp_P)$ where $poPo_P \in \{+, -, \cdot\}$ is its polarity such that the $+$ ($-$) polarity means that the port is a service-providing port (a service-requesting port), and $poTyp_P \in S_{PoTyp}$ is its port type. We also define the following functions:*

- $polarity : S_{Po} \rightarrow \{+, -, \cdot\}$, such that $polarity(P) = poPo_P$,
- $portType : S_{Po} \rightarrow S_{PoTyp}$, such that $portType(P) = poTyp_P$,
- $portState : S_{Po} \rightarrow \{V, H\}$, such that $portState(P)$ returns the current state of the port P . The two possible states V and H of a port respectively denote its Visible and Hidden states. A port can participate in a connection only if it is in its Visible state, or, in other words, not in its Hidden state.

No two ports of the same smart object can share the same type and the same polarity. More formally, the following condition must always hold true.

$$\begin{aligned} \forall O \in S_{Obj} \forall P, Q \in Ports_O \ ((P \neq Q) \\ \subset ((polarity(P) \neq polarity(Q)) \vee (portType(P) \neq portType(Q)))) \end{aligned}$$

In other words, in each smart object, a port is uniquely identified by its type and its polarity. Therefore, a providing (requesting) port of type t is denoted by $+t$ ($-t$).

Definition 4 (Smart Object Connection). *A connection C is a triple $C = (sour_C, dest_C, typ_C)$ where $sour_C$ and $dest_C$ are a smart object, and $typ_C \in S_{PoTyp}$ is the type of the connection. We say that there exists a connection of type typ_C from an SO $sour_C$ to an SO $dest_C$. Such connection is possible if and only if the following three conditions all hold true.*

- (1) $oid(sour_C) \neq oid(dest_C)$
- (2) *The SO $sour_C$ has a port p_s and the SO $dest_C$ has a port p_d such that*
 - (a) $polarity(p_s) = '-'$,
 - (b) $polarity(p_d) = '+'$,
 - (c) $portType(p_s) = portType(p_d) = typ_C$, and
 - (d) $portState(p_s) = portState(p_d) = 'V'$.
- (3) *The SO $dest_C$ is in the scope of $sour_C$. The scope of an SO o is defined as the set of all the SOs with which o can initiate a communication, say, through wireless communication.*

Such a connection will be denoted by $sour_C \xrightarrow{typ_C} dest_C$. Each port can be connected to no more than one other port. There can be more than one connection from $sour_C$ to $dest_C$ using different ports.

3.2 Graph Rewriting Rules

3.2.1 Smart Object Graph (SOG)

In this section, we give the definition of a smart object graph (SOG). An SOG is an acyclic directed edge-labelled vertex-labelled graph. We use an SOG to represent a federation of smart objects. The vertices of a SOG represent the smart objects of

Chapter 3. Formalization of our Framework

a federation. A labelled edge between two vertices means a connection between the two smart objects represented by these vertices. A non-labelled edge between two vertices means a proximity relationship between the two smart objects represented by these vertices. We introduce three types of SOGs. The first type consists of instance SOGs. Instance SOGs are used to represent concrete federations that can be found in the real world environment. The second type consists of parameterized SOGs. Parameterized SOGs are used in the second level of our formal model, i.e., the graph rewriting rules in Section 3.2.3, to define our rules. The third type consists of template SOGs. Template SOGs are used to define classes of SOGs. Classes of SOGs are used in Section 3.3 to define basic concepts. We use these concepts to build the third level of our formal model, i.e., the catalytic-reaction-network model. We also reuse these concepts to prove the validity of our framework in Chapter 4. Moreover, we define two types of isomorphism of SOGs. The first one gives the conditions required by a parameterized SOG to be isomorphic with an instance SOG. We use such an isomorphism to define the applicability of graph rewriting rules in Section 3.2.3. The second type of isomorphism gives the conditions required by a template SOG to be isomorphic with an instance SOG. We use such an isomorphism to define the set of instances of a template SOG. Finally, we present the graphical representation of an SOG.

Definition 5. An SOG G is a triple $G = (V_G, C_G, P_G)$ where V_G is a finite set of vertices that represent SOs, $C_G(\subseteq V_G \times V_G \times S_{PoTyp})$ is a finite set of labelled-directed edges called the connection set, and $P_G(\subseteq V_G^2)$ is a finite set of directed edges called the proximity set.

By “a smart object A ” with $A \in V_G$, we denote the smart object represented by the vertex A . Vice versa, by “a node A ” with $A \in V_G$, we denote a smart object A . An edge $(s, d, t) \in C_G$ means that there is a connection of type t from the smart object s to the smart object d . An edge $(s, d) \in P_G$ means that the smart object d is in the scope of the smart object s . In a SOG, a directed path from a source smart object s to a destination smart object d is uniquely defined by the source smart object s and the sequence of edge labels σ of this path. This is because any requesting port has no more than one connection. For example, in pictures Fig. 3.1, the path from the black node to the grey node is given by the sequence t, t, u that starts from the black node. We call the destination smart object d the σ object of s . We denoted by $\sigma(s)$ the smart object d . In other words, $\sigma(s) = d$. As the special case, the length of σ can be zero, namely the σ object of s can be s itself.

There are three types of SOGs. The first type consists of instance SOGs. An

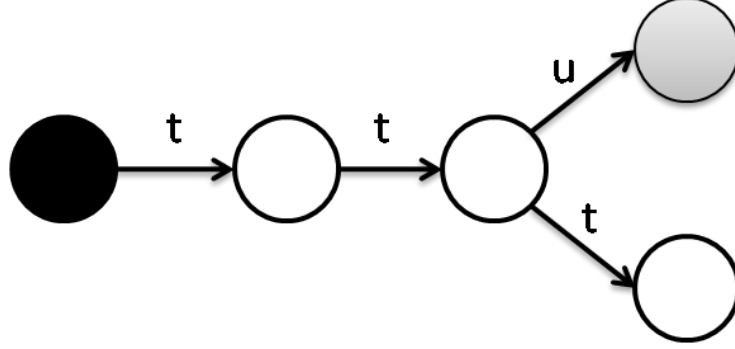


Figure 3.1: A graph representing a federation of smart object to illustrate a directed path. The nodes represent the smart objects. The directed edge represent the connections. The label over the edges represent the type of the connections.

instance SOG represents a federation of SOs with their connections. The second type consists of parameterized SOGs. A parameterized SOG is similar to an instance SOG but may have some SOs that have variables as their SO types and/or SO states. The third type consists of template SOGs. A template SOG is similar to an instance SOG but have all its SOs with their states unspecified.

Definition 6 (Isomorphic parameterized SOG). *A parameterized SOG G is said to be isomorphic to an instance SOG H if and only if there exists a bijection $f : V_G \rightarrow V_H$ that satisfies the following seven conditions.*

- (1) $\forall g \in V_G \left((type(g) \neq 'variable') \supset (type(g) \geq type(f(g))) \right)$
- (2) $\forall g \in V_G \left((state(g) \neq 'variable') \supset (state(g) = state(f(g))) \right)$
- (3) $\forall g \in V_G \forall p_g \in ports(g) \exists p_h \in ports(f(g))$
 $\left((polarity(p_g) = polarity(p_h)) \wedge (portType(p_g) = portType(p_h)) \right)$
 $\wedge (portState(p_g) = portState(p_h))$
- (4) $\forall (g_s, g_d, g_t) \in C_G (f(g_s), f(g_d), g_t) \in C_H$
- (5) $\forall (h_s, h_d, h_t) \in C_H \exists (g_s, g_d, g_t) \in C_G$
 $\left((h_s = f(g_s)) \wedge (h_d = f(g_d)) \wedge (h_t = f(g_t)) \right)$
- (6) $\forall (g_s, g_d) \in P_G (f(g_s), f(g_d)) \in P_H$
- (7) $\forall (h_s, h_d) \in P_H \exists (g_s, g_d) \in P_H \left((h_s = f(g_s)) \wedge (h_d = f(g_d)) \right)$

Definition 7 (Isomorphic template SOG). *A template SOG G is said to be isomorphic to an instance SOG H if and only if there exists a bijection $f : V_G \rightarrow V_H$ that satisfies the following four conditions.*

- (1) $\forall g \in V_G \left(type(g) = type(f(g)) \right)$
- (2) $\forall g \in V_G \forall p_g \in ports(g) \exists p_h \in ports(f(g))$
 $\left((polarity(p_g) = polarity(p_h)) \wedge (portType(p_g) = portType(p_h)) \right)$
 $\wedge (portState(p_g) = portState(p_h))$
- (3) $\forall (s, d, t) \in C_G (f(s), f(d), t) \in C_H$
- (4) $\forall (s, d) \in P_G (f(s), f(d)) \in P_H$

In both of the above definitions, we consider a bijection $f : V_G \rightarrow V_H$, which implies that $\forall h \in V_H \exists g \in V_G (f(g) = h)$, namely that f is an onto mapping. We say that an instance SOG H is an instance of a template SOG G if and only if G is isomorphic to H .

We also define the following operation.

Definition 8 (SOG deletion). *Let G and H be two SOGs satisfying that H is a sub-graph of G . Then we define the operation SOG deletion as*

$$G - H = I$$

where I is a sub-graph of G satisfying the following three conditions.

- (1) $V_I = V_G - V_H$
- (2) $\forall (s, d, t) \in C_G ((s \notin V_H \wedge d \notin V_H) \supset ((s, d, t) \in C_I))$
- (3) $\forall (s, d) \in P_G ((s \notin V_H \wedge d \notin V_H) \supset ((s, d) \in P_I))$

3.2.2 Graphical Representation of an SOG

The different elements used to graphically represent an SOG are presented in Fig. 3.2. A node (representing a smart object) is represented by a circle (Fig. 3.2a). A non-connected visible $-p$ port of a node is represented by an outgoing arrow labelled with p and pointing to a black dot (Fig. 3.2b). A non-connected visible $+p$ port is represented by an incoming arrow (Fig. 3.2c). A hidden $-p$ port of a node is represented by an outgoing arrow labelled with p that connects this node to a black cross (Fig. 3.2d). A hidden $+p$ port is represented by an incoming arrow (Fig. 3.2e).

A connection p from a node s to a node d is represented by an arrow from the node s to a node d with the label p (Fig. 3.2f). The type of a node is specified by a label given around the node (Fig. 3.2g). The state of a node is shown by a label inside the node (Fig. 3.2h). If a node has a variable as its state and/or as its type, no label is given for the state and/or the type. A proximity edge from a node s to a node d denotes that d is in the scope of s , and is represented by a dashed arrow from s to d (Fig. 3.2i).

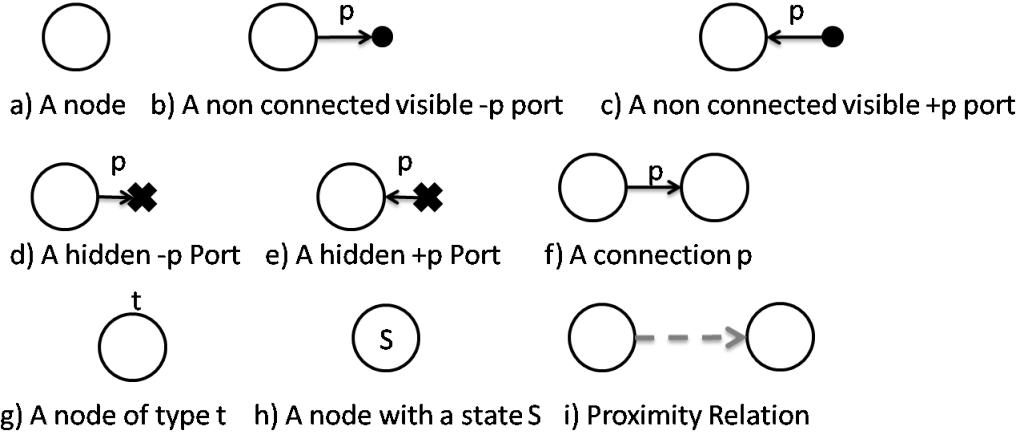


Figure 3.2: The different elements for the graphical representation of SOGs.

3.2.3 Definition of the Graph Rewriting Rules

Here we define the second level of our formal model introduced by [51], namely the graph-rewriting-rule modelling. It is an unordered graph rewriting system [2]. In this model, each graph rewriting rule $r = (L_r, R_r, a_r)$ is a triple that consists of an activated node $a_r \in L_r$ and a pair of two parameterized SOGs L_r and R_r such that $V_{L_r} = V_{R_r}$. The SOGs L_r and R_r are respectively called the left- and the right-hand side of the rewriting rule. The left-hand side L_r represents the condition of applying this rule, and the right-hand side R_r corresponds to the change result of the left-hand side after applying this rule. It is possible to apply the rule r to an instance SOG G if L_r is isomorphic to a sub-graph H of G . Let $f_l : V_{L_r} \rightarrow V_H$ be the bijection for this isomorphism. Then the node $f_l(a_r)$ is called an activated node of G . It is possible to have more than one activated node in G . For example, let another rule r' having its SOG L'_r isomorphic to a sub-graph H' of G , and let $f'_l : V_{L'_r} \rightarrow V_{H'}$ be the bijection for this isomorphism. Then the node $f'_l(a_{r'})$ is also an activated node. That is why, if H and H' share some nodes, the execution of the

Chapter 3. Formalization of our Framework

rule r may conflict with the execution of the rule r' . To avoid such a conflict, the node $f_l(a_r)$ will lock all the nodes of H to reserve them for the application of the rule r . If one of these nodes is already locked by another activated node other than $f_l(a_r)$ then the application of the rule r fails. Otherwise, the rule r is applied to G to obtain J through the rewriting of the sub-graph H to the sub-graph I satisfying the following three conditions.

- (1) R_r is isomorphic to I . Let $f_r : V_{R_r} \rightarrow V_I$ be the bijection for this isomorphism.
- (2) There is a bijection $o : V_G \rightarrow V_J$ such that $\forall g \in V_G \text{ } oid(g) = oid(o(g))$ and $\forall (s, d, t) \in C_G \text{ } ((s, d, t) \notin C_H) \supset ((o(s), o(d), t) \in C_J)$
- (3) $\forall e \in V_{L_r} \text{ } o(f_l(e)) = f_r(e)$

Once the rule r is applied, the node $f_l(a_r)$ unlocks the nodes.

Because we are considering distributed environments with many SOs, we cannot assume any central processing object or service that triggers and applies all the rules. Each graph rewriting rule is performed by some node. However, it cannot be performed by any arbitrary node, because a node b in C_H may not be able to access another node c in C_H . Such a situation may occur when there is no directed path from b to c in C_H . Hence, it is necessary for each rule to specify an activated node a_r such that its image $f_l(a_r)$ will execute this rule. This activated node must have a directed path in C_{L_r} to every other node of L_r , except the neighbouring nodes of L_r , where each neighbouring node n of L_r is defined as follows:

- (1) n is a node of L_r .
- (2) There exists no (undirected) path in C_{L_r} from a node of L_r to n .
- (3) $type(n)$ is a variable.
- (4) $state(n)$ is a variable.
- (5) The node n has only one port that is a service providing port of some type t in its state visible and not connected.
- (6) There exists a node $m \neq n$ in L_r such that
 - (a) m is the node a_r or there is a directed path in C_{L_r} from a_r to m ,
 - (b) m has a service requesting port of the type t in its state visible and not connected,
 - (c) the proximity set P_{L_r} of L_r has an edge (m, n) .

3.2.4 Graphical Representation of Graph Rewriting Rules

In Fig. 3.3 we graphically represent two examples of graph rewriting rules. Each rule is represented in a round-corner frame. The name of the rule is given on top of this frame. In this frame, the graph on the left-hand side (respectively on the right-hand side) of the bold big arrow corresponds to the parameterized SOG L_r (respectively to R_r) of the rule r . The activated node a_r is represented by a unique grey node. The rule $r2$ shows a case in which a neighbouring node is involved.

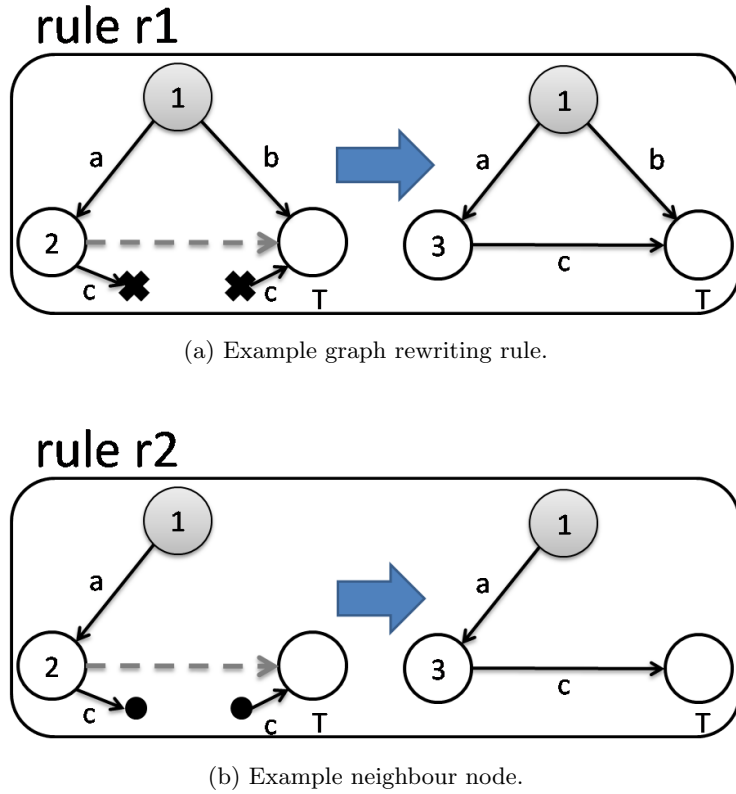


Figure 3.3: Some example graph rewriting rules

3.2.5 Operations for Triggering and Applying a Rule

Our system is a distributed system. There is no central process that evaluates and applies the rules. It is the role of SOs to execute the rules. We provide each SO with a list of primitive operations classified into two categories. The first category consists of the primitive condition-checking operations that are used to check the triggering conditions of rules. The second category consists of the primitive-action

Table 3.1: Primitive condition-checking operations

Names	Descriptions
isIdentical ($\sigma 1, \sigma 2$)	returns true if the oid of the $\sigma 1$ object is the same as the oid of the $\sigma 2$ object, or false otherwise.
hasType (σ, t)	returns true if the type of the σ object is t , or false otherwise.
hasSameType ($\sigma 1, \sigma 2$)	returns true if the type of the $\sigma 1$ object is the same than the type of the $\sigma 2$ object, or false otherwise.
hasState (σ, s)	returns true if the state of the σ object is s , or false otherwise.
isVisible (σ, po, p)	returns true if the σ object has a visible port with its polarity po and of type p , or false otherwise.
isHidden (σ, po, p)	returns true if the σ object has a hidden port with its polarity po and of type p , or false otherwise.
inScope ($\sigma 1, \sigma 2$)	returns true if the $\sigma 2$ object is in the scope of the $\sigma 1$ object, or false otherwise.
neighbor (σ, p)	returns true if at least one SO with a visible and non-connected $+p$ port is in the scope of the σ object, or false otherwise.

operations that are used to execute graph rewritings.

Primitive Conditions

We listed up in Table 3.1 primitive condition-checking operations. They are used by an SO to check the applicability of a rewriting rule. Each SO will use these condition-checking operations to check if it can activate itself to work as the activated node of some rule. In these condition-checking operations, if a parameter sigma object such as σ or σi does not exist, then the condition-checking operation returns false.

For example, when an SO tries to execute the rule $r1$ presented in Fig. 3.3a, it will execute the following conditions-checking operations in this order:

- (1) $hasState(' ', 1)$,
- (2) $hasState('a', 2)$,
- (3) $hasType('b', T)$,
- (4) $isHidden('b', -, c)$,
- (5) $isHidden('a', +, c)$, and
- (6) $inScope('a', 'b')$.

If all these operations return true, the SO will apply this rule.

Primitive Actions

We listed up in Table 3.2 primitive-action operations that each SO uses to rewrite the SOG in which it is involved. In these primitive-action operations, if a parameter sigma object σ or σi does not exist, then the action does not rewrite the graph. For example, the graph rewriting by the rule $r1$ shown in Fig. 3.3a is performed by the sequential execution of $setState('a', 3)$ and $span('a', 'b', c)$ in this order. The execution of the sequence of actions of each rule is atomic. Abstractly, that means that either all the actions in a sequence are executed, or none of the actions is executed. Concretely, if some action in the sequence fails, then the target sub-graph is initialized to the state before the rule application.

3.3 Catalytic-Reaction-Network Framework

Here we will formally define a reaction in the third level of our formal model, i.e., the catalytic reaction network model. Then, we will define a special type of smart objects called nucleotide smart objects (NSOs). NSOs will be used to establish the mapping between the second level of our formal model, i.e., the graph rewriting model, and the third level of our formal model.

3.3.1 A Reaction in the Catalytic-Reaction-Network Framework

In our framework, a material is an application SO, or a federation of application SOs. There are two types of catalytic reactions, i.e., composition reactions, and decomposition reactions. A composition reaction denotes the federation of the SOs of more than one material into a new material. A decomposition reaction denotes the decomposition of a material into more than one new material.

Table 3.2: Primitive-action operations

Names	Descriptions
expose (σ, po, p)	If the σ object has a port with its polarity po and of type p then this operation changes the state of this port to Visible, and returns true. Otherwise, it returns false.
hide (σ, po, p)	If the σ object has a port with its polarity po and of type p , then this operation changes the state of this port to Hidden, and return true. Otherwise, it returns false.
setState (σ, S)	If the state S is in the set of states of the σ object, the action changes the state of the σ object to S , and returns true. Otherwise, it returns false.
span ($\sigma1, \sigma2, p$)	If the $\sigma1$ object and the $\sigma2$ object respectively have a non-connected $-p$ port and a non-connected $+p$ port, and if the $\sigma2$ object is in the scope of the $\sigma1$ object, then this operation creates an edge labelled with p from the $\sigma1$ object to the $\sigma2$ object and returns true. Otherwise, it returns false.
spanNeighbor (σ, p)	If the σ object has a non-connected $-p$ port and if at least one SO with a non-connected visible $+p$ port is in the scope of the σ object, then this operation creates an edge labelled with p from the σ object to one of such SOs and returns true. Otherwise, it returns false. This operation is non-deterministic. Which object with the $+p$ port to choose is an implementation issue.
break (σ, p)	If $\sigma.p$ object exists then this operation removes the edge labelled with p from the σ object and returns true. Otherwise, it returns false.

For example, in a medical context, we can imagine a reaction ρ in which the first material is a biological implant (e.g., a pacemaker, or an insulin pump) in a patient and the second material is a tablet computer of a doctor. Both of these materials are called the input materials of the reaction. This reaction will occur only at the entrance of a consultation room. The entrance is called the context of this reaction. The reaction ρ occurs when both the patient and the doctor simultaneously enter the same consultation room. The tablet computer of the doctor will be automatically connected to the implant in the patient to form a composite material M called the output material of the reaction. Such reaction ρ is called a composition reaction.

Vice-versa, we can imagine a reaction ρ' in which the unique input material is a composite material M . This reaction will occur only at the exit of a consultation room. When both the patient and the doctor simultaneously exit the same consultation room the reaction occurs. The tablet computer of the doctor and the implant in the patient, which form the material M , will be automatically disconnected. Such reaction ρ' is called a decomposition reaction.

A reaction may require a stimulus to occur. A stimulus is another material. In this example, the reaction R may occur only in the presence of a stimulus that is the doctor's ID card.

Definition 9 (Materials and Material types). *A material is defined as a federation of application SOs ordered in a sequence $o = o_1, \dots, o_n$ where $n \geq 1$. Let t_i be the type of the application SO o_i for $i = 1, \dots, n$. Then, the type of this material is defined as a sequence $t = (t_1, \dots, t_n)$. For two materials of type $t = (t_1, \dots, t_n)$ and $u = (u_1, \dots, u_m)$, the type of their federation in this order is defined as the concatenation of t and u , i.e., $(t_1, \dots, t_n, u_1, \dots, u_m)$, and denoted by $t \rightarrow u$.*

Definition 10 (Composition Reactions). *A composition reaction ρ is defined as a pair $\rho = \langle S_\rho, B_\rho \rangle_C$ where S_ρ is a material type, and $B_\rho = (B_1, B_2, \dots, B_k)$ is a sequence of k material types. If the reaction ρ does not require a stimulus, S_ρ becomes nil. A material of type S_ρ is called the stimulus of the reaction, and materials of type $B_1, \dots, B_k$ are called the inputs of the reaction. The output of such a reaction ρ is the material of type $B_1 \rightarrow B_2 \rightarrow \dots \rightarrow B_k$.*

Definition 11 (Decomposition Reactions). *A decomposition reaction ρ' is defined as a pair $\rho' = \langle S_{\rho'}, B_{\rho'} \rangle_D$ where $S_{\rho'}$ is a material type, and $B_{\rho'} = (B_1, B_2, \dots, B_k)$ is a sequence of k material types. If the reaction ρ' does not require a stimulus, $S_{\rho'}$ becomes nil. A material of type $S_{\rho'}$ is called the stimulus of the reaction, and a material of type $B_1 \rightarrow B_2 \rightarrow \dots \rightarrow B_k$ is called the input of the reaction. The outputs of such a reaction ρ' are materials of types $B_1, \dots$, and B_k .*

Chapter 3. Formalization of our Framework

By $CReacts$ is denoted the set of all the possible composition reactions. By $DReacts$ is denoted the set of all the possible decomposition reactions.

For any $\rho \in CReacts \cup DReacts$, we define the following functions:

- $\alpha : CReacts \cup DReacts \rightarrow \{0, 1\}$ such that $\alpha(\rho)$ returns 1 if the material type S_ρ is nil, and 0 otherwise,
- $\omega : CReacts \cup DReacts \rightarrow \mathbb{N}$ such that $\omega(\rho)$ returns the size of the sequence B_ρ , and
- $context$ such that $context(\rho)$ returns some coordinates representing a place in the real world environment called context of the reaction ρ .

We say that the composition reaction ρ occurs if the two following conditions are satisfied.

- (1) There are at least k materials $M_1, M_2, \dots, M_k$ that physically exist in the real world environment at $context(\rho)$ such that the type of M_i is B_i for any $i \in \{1, \dots, k\}$.
- (2) If $S_\rho \neq nil$, there is at least one material of type S_ρ that physically exists in the real world environment at $context(\rho)$.

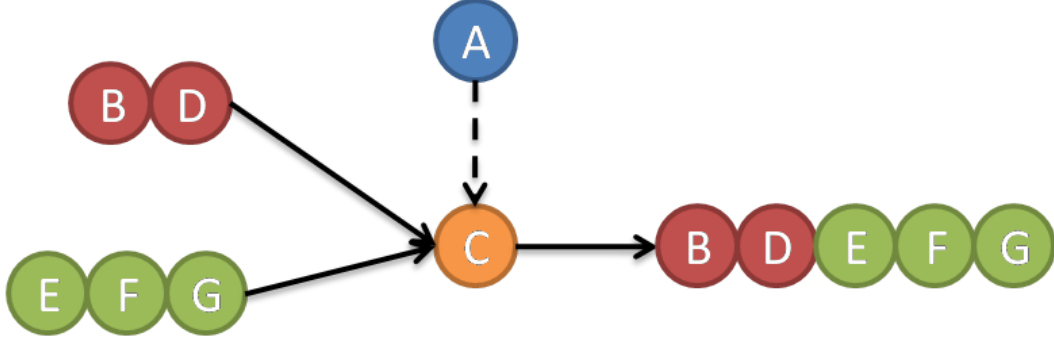
If the composition reaction ρ occurs, the k materials $M_1, M_2, \dots, M_k$ are federated together to form the output material of type $B_1 \rightarrow B_2 \rightarrow \dots \rightarrow B_k$.

We say that the decomposition reaction ρ' occurs if the two following conditions are satisfied.

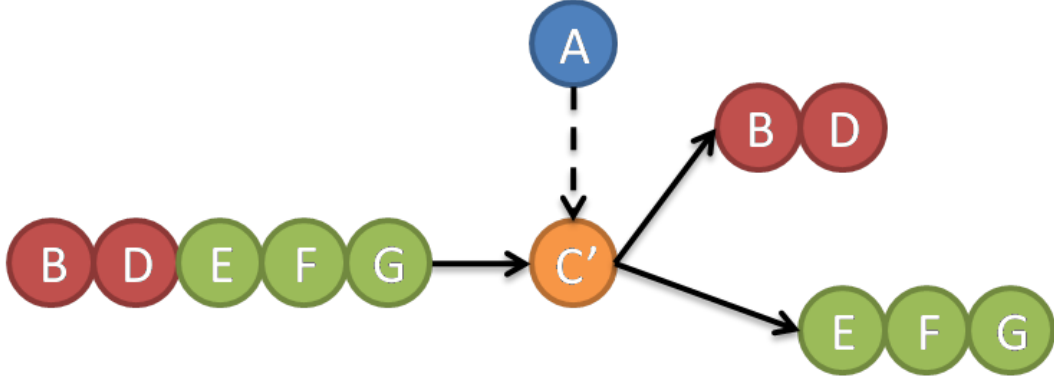
- (1) There is at least one material M that physically exists in the real world environment at $context(\rho')$ such that the type of M is $B_1 \rightarrow B_2 \rightarrow \dots \rightarrow B_k$.
- (2) If $S_{\rho'} \neq nil$, there is at least one material of type $S_{\rho'}$ that physically exist in the real world environment at $context(\rho')$.

If the decomposition reaction ρ' occurs, the material M is decomposed into k output materials $M_1, M_2, \dots, M_k$ such that the type of M_i is B_i for any $i \in \{1, \dots, k\}$.

For example, Fig. 3.4 shows a composition reaction $\rho = (A, B_\rho)$ where $B_\rho = (B_1, B_2)$, $B_1 = (B, D)$, and $B_2 = (E, F, G)$. The inputs of this reaction are materials of type $B_1 = B, D$ and $B_2 = E, F, G$. The outputs of this reaction are materials of type $(B_1 \rightarrow B_2) = (B, D, E, F, G)$. This reaction can only occurs in the context $context(\rho) = C$.

Figure 3.4: An example of a composition reaction ρ .

And Fig. 3.5 shows a decomposition reaction $\rho' = (A, B_{\rho'})$ where $B_{\rho'} = (B_1, B_2)$, $B_1 = (B, D)$, and $B_2 = (E, F, G)$. The inputs of this reaction are materials of type $(B_1 \rightarrow B_2) = (B, D, E, F, G)$. The outputs of this reaction are materials of type $B_1 = B, D$ and $B_2 = E, F, G$. This reaction can only occur in the context $context(\rho') = C$.

Figure 3.5: An example of a decomposition reaction ρ' .

3.3.2 Nucleotide Smart Objects

Our goal is to provide a generic framework for each catalytic reaction based on the graph rewriting rule model. This framework is based on the emulation of the replication process of RNAs as well as their regulation switch mechanisms. We use this idea to make more than one SO federate with each other. This framework uses seven different kinds of NSO types that are defined in the hierarchy as shown in Fig. 3.6. Each NSO has the type N , and two ports, i.e. a $-L$ port and a $+L$ port. For each type X of application SOs, we use NSOs of type N_X . Each NSO

of type N_X works as a tag for any SO of type X , and has two ports, i.e., $-B[N_X]$ and $+B[N_X]$. Later, these ports are simply denoted by the B ports of NSOs. A connection established by these ports is denoted by a B connection. The type T is defined as the abstract type of all different N_X and called the tag type. Subtypes SS , IS and OS of N are respectively called the stimulus separator type, the input separator type, and the output separator type. The subtype S is defined as the abstract type of SS , IS , and OS . This type S is called the separator type. The roles of these separator types will be explained later.

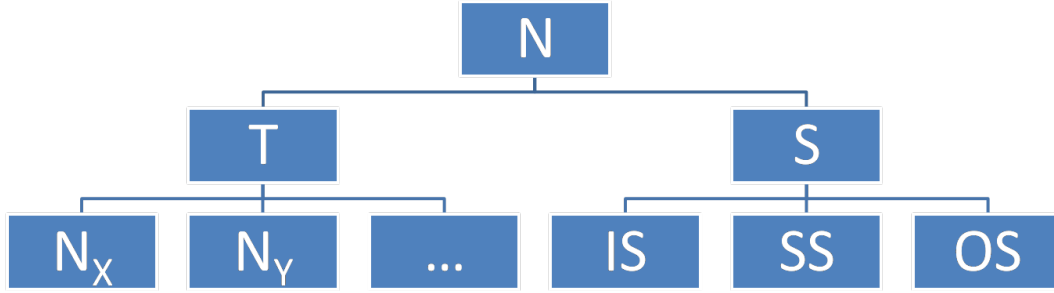


Figure 3.6: The hierarchy of the six different kinds of NSO types.

3.3.3 Strands of Nucleotide Smart Objects

To implement catalytic reactions, we want to mimic the self-replication mechanism of RNA. RNAs are a single-strand molecules. For this reason, we introduce strand of NSOs. Strands of NSOs are the basic SOG structure of our system. We define two types of strands, i.e., template strands and instance strands.

Definition 12 (Strands). *A template strand is defined as a template SOG consisting of a single acyclic-directed path satisfying all the followings. (1) all nodes are NSOs. (2) all edges have the same label L . (3) the state of each B port is unspecified. A strand V of length n is represented as $V : v_1 \rightarrow v_2 \rightarrow \dots \rightarrow v_n$ where $v_1, v_2, \dots, v_n$ denote the NSOs in the strand and each arrow denotes a directed edge labelled with L .*

An instance SOG U that is an instance of a template strand V is called an instance strand.

Let $V : v_1 \rightarrow v_2 \rightarrow \dots \rightarrow v_n$ be a template (or instance) strand. The nodes v_1 and v_n in V are respectively called the first NSO and the last NSO of the strand V .

For a strand of length one, the first NSO and the last NSO coincide. The NSO v_i is called the i -th NSO of the strand V . The size of V is defined to be the length of V and is denoted by $\text{size}(V)$.

We will now define some properties of template and instance strands.

Definition 13 (Terminated template (or instance) strand and concatenation). *A template (or an instance) strand is said to be terminated if and only if the $-L$ port of its first NSO and the $+L$ port of its last NSO are both hidden.*

Let $U : u_1 \rightarrow \dots \rightarrow u_n$ and $V : v_1 \rightarrow \dots \rightarrow v_m$ be two template (or instance) strands. An expression $U \rightarrow V$ denotes a template (or instance strand) that is a concatenation of U and V , i.e., $u_1 \rightarrow \dots \rightarrow u_n \rightarrow v_1 \rightarrow \dots \rightarrow v_m$. When the length of U or V is one, we may also use the notation $u \rightarrow V$ or $U \rightarrow v$.

Definition 14 (Template (or instance) strand compatibility). *Two template (or instance) strands $V : v_1 \rightarrow v_2 \rightarrow \dots \rightarrow v_n$ and $U : u_1 \rightarrow u_2 \rightarrow \dots \rightarrow u_m$ are said to be compatible, and denoted as $V \simeq U$ if and only if $n = m$ and $\text{type}(v_i) = \text{type}(u_i)$ for any $i = 1, \dots, n$.*

Definition 15 (Proximity between compatible strands). *For two compatible template or instance strands $V : v_1 \rightarrow v_2 \rightarrow \dots \rightarrow v_n$ and $U : u_1 \rightarrow u_2 \rightarrow \dots \rightarrow u_n$ that are sub-graphs of the same SOG G , V is said to be in the proximity of U in G if and only if for any $i = 1, \dots, n$ there is a directed edge from u_i to v_i in the proximity set P_G of G .*

3.3.4 Instance-SOG family of a reaction

Now we will use the strands previously defined to model catalytic reactions. For this purpose, we will first define the input-strand representation, the stimulus-strand representation and the context-strand representation. After that, we will define the SOG representation of each catalytic reaction called the instance-SOG family of this reaction. Then, we will provide our graph rewriting rules applied to this SOG representation and show how each reaction is implemented by this SOG representation with these rewriting rules.

Definition 16 (Material-strand representation and Material SOG). *Let m be a material of type $t = (t_1, \dots, t_n)$. The strand representation of m is a template strand $U : u_1 \rightarrow \dots \rightarrow u_n$ such that $\text{type}(u_i) = N_{t_i}$ for any $i = 1, \dots, n$. We called such a strand a material strand. It is assumed that each NSO u_i is previously connected to an SO o_i of type t_i through a special connection P as shown in Fig. 3.7a*

Chapter 3. Formalization of our Framework

for making itself work as the tag of o_i . The template SOG composed of the SOs o_i and of the NSOs u_i is called the material SOG of the material m , and is denoted by $mat(t)$. The connection set of $mat(t)$ is neglected. All the NSOs in a material strand have their $-B$ ports hidden. The NSOs of a material strand and of an instance of a material strand are called material NSOs. If m is one of the inputs, the output or the stimulus of a reaction ρ , U is called an input strand, the output strand, or the stimulus strand of the reaction ρ , respectively.

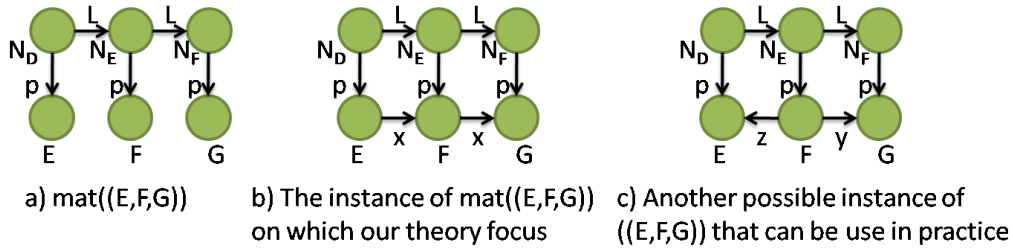


Figure 3.7: An example of material SOG for materials of type (E, F, G) and some of its instances. The states are not represented in the instances.

A material m of type t in the third level of our formal model is represented by any instance of $mat(t)$, in the second level of our formal model. For example, a material of type (E, F, G) is represented by an instance of the SOG $mat((E, F, G))$ (shown in Fig. 3.7a). They may be extra connections between the application SOs in a material-SOG instance, i.e., between the SOs E , F , and G in the example. However, these connections are neglected to determine if an instance SOG is a representation of a material. For example, the SOGs shown in Fig. 3.7b and in Fig. 3.7c are both instance of $mat((E, F, G))$, and thus are both valid representation of a material of type (E, F, G) , even if the connection among the SOs E , F , and G are different.

Definition 17 (Basic Material strands of a reaction). *For any reaction $\rho \in CReacts \cup DReacts$ such that $\rho = \langle S_\rho, B_\rho \rangle$, and $B_\rho = (B_1, B_2, \dots, B_k)$. The basic material strands of the reaction ρ is a sequence $(M_{\alpha(\rho)}, \dots, M_k)$ of template material strands such M_i is the strand representation of a material of type B_i for any $i = 1, \dots, k$. If $\alpha(\rho) = 0$, M_0 is the strand representation of a material of type S_ρ .*

Definition 18 (Context strand representation). *For any reaction $\rho \in CReacts \cup DReacts$, let the sequence $(M_{\alpha(\rho)}, \dots, M_k)$ be the basic material strands of the reaction ρ . The context of this reaction is represented in the second level of our formal*

model by a terminated template strand

$$C : C_{\alpha(\rho)} \rightarrow s_{\alpha(\rho)} \rightarrow \cdots \rightarrow C_{k-1} \rightarrow s_{k-1} \rightarrow C_k$$

called the context strand. Each C_i is a strand satisfying that $C_i \simeq M_i$ for any $i = \alpha(\rho), \dots, k$. Each s_i is an NSO satisfying that

(1) $\text{type}(s_i) = IS$ for any $i = 1, \dots, k-1$ if $\rho \in CReacts$, or

(2) $\text{type}(s_i) = OS$ for any $i = 1, \dots, k-1$ if $\rho \in DReacts$,

and $\text{type}(s_0) = SS$ if $\alpha(\rho) = 0$. Each C_i in this context strand is called its i -th docking strand. All the NSOs of a docking strand have their $+B$ ports hidden. The NSOs of a context strand and of its instances are called context NSOs.

Definition 19 (Template SOG and instance-SOG family of a reaction). For any reaction $\rho \in CReacts \cup DReacts$, the template SOG of the reaction ρ is a template SOG G consisting only of the context strand $C : C_{\alpha(\rho)} \rightarrow s_{\alpha(\rho)} \rightarrow \cdots \rightarrow C_{k-1} \rightarrow s_{k-1} \rightarrow C_k$ and the basic material strands $(M_{\alpha(\rho)}, \dots, M_k)$ of this reaction. In G , for any $i = \alpha(\rho), \dots, k$, M_i is assumed to be in the proximity of C_i . Each pair (C_i, M_i) of strands for $\alpha(\rho) \leq i \leq k$ is called the i -th double-strand of G , and is denoted by D_i . The size of the i -th double-strand of a reaction ρ is defined to be the length of C_i and denoted by $\text{size}(\rho, i)$. The strand C , each C_i , M_i , D_i , and each s_j (for $i = \alpha(\rho), \dots, k$ and for $j = \alpha(\rho), \dots, k-1$) are called a component of G . For any component X of G , $X(\rho)$ refers to the template-SOG X . Let G' be an instance SOG such that G is isomorphic to G' , and let $f : V_G \rightarrow V_{G'}$ be the bijection for this isomorphism. For any G' , and for any component X of G , $X(G')$ refers to a sub-graph in G' with its node set $V_{X(G')}$ equal to $f(V_X)$.

The instance-SOG family of a reaction ρ , denoted by $ISOGF(\rho)$, is the set of all the possible instance SOGs that are instances of the template SOG of this reaction. Each of these instances are called an instance SOG of the reaction ρ . Let $ReactSOGs$ be the set of all the possible instance SOGs of all the possible reactions formally defined as follows:

$$ReactSOGs = \{G | \exists \rho \in CReacts \cup DReacts \ G \in ISOGF(\rho)\}.$$

Definition 20 (Initialized context-strands). For any reaction $\rho \in CReacts \cup DReacts$, the initialized context-strand of ρ is an instance of $C(\rho)$ satisfying that each of its NSOs is in the state i_c and has its $-B$ port hidden.

Chapter 3. Formalization of our Framework

Definition 21 (Initialized visible-strands). *Let $U = u_1 \rightarrow u_2 \rightarrow \dots \rightarrow u_p$ be a template strand for some $p \geq 1$. The initialized material-strand of U is an instance of U satisfying that each of its NSOs is in the state i and has its $+B$ port hidden, except the first NSO for which the $+B$ port is visible but not connected.*

Definition 22 (Initialized hidden-strands). *Let $U = u_1 \rightarrow u_2 \rightarrow \dots \rightarrow u_p$ be a template strand for some $p \geq 1$. The initialized material-strand of U is an instance of U satisfying that each of its NSOs is in the state i and has its $+B$ port hidden.*

Definition 23 (Initial reaction-SOG). *For any reaction $\rho \in CReacts$, the initial reaction-SOG of ρ is defined as an SOG $G \in ISOGF(\rho)$ satisfying that*

- (1) *each $M_i(G)$, for any $i = \alpha(\rho), \dots, \omega(\rho)$, is a terminated strand and the initialized visible-strand of $M_i(\rho)$, and*
- (2) *the strand $C(G)$ is the initialized context-strand of ρ .*

For any reaction $\rho' \in DReacts$, the initial reaction-SOG of ρ' is defined as an SOG $G' \in ISOGF(\rho')$ satisfying that

- (1) *the strands $M_1(G'), \dots, M_{\omega(\rho')}(G')$ form a terminated strand $M : M_1(G') \rightarrow M_2(G') \rightarrow \dots \rightarrow M_{\omega(\rho')}(G')$,*
- (2) *the strand M is the initial visible-strand of the template-strand $M_1(\rho') \rightarrow M_2(\rho') \rightarrow \dots \rightarrow M_{\omega(\rho')}(\rho')$, and*
- (3) *if $\alpha(\rho') = 0$, $M_0(G')$ is a terminated strand that is the initial visible-strand of $M_0(\rho')$,*
- (4) *the strand $C(G')$ is the initialized context-strand of ρ' .*

Definition 24 (Final reaction-SOG). *For any reaction $\rho \in CReacts$, the initial reaction-SOG of ρ is defined as an SOG $G \in ISOGF(\rho)$ satisfying that*

- (1) *the strands $M_1(G), \dots, M_{\omega(\rho)}(G)$ form a terminated strand $M : M_1(G) \rightarrow M_2(G) \rightarrow \dots \rightarrow M_{\omega(\rho)}(G)$,*
- (2) *the strand M is the initial visible-strand of the template-strand $M_1(\rho) \rightarrow M_2(\rho) \rightarrow \dots \rightarrow M_{\omega(\rho)}(\rho)$,*
- (3) *if $\alpha(\rho) = 0$, $M_0(G)$ is a terminated strand that is the initial visible-strand of $M_0(\rho)$,*

(4) the strand $C(G)$ is the initialized context-strand of ρ .

The strand M is called the output strand of the reaction.

For any reaction $\rho' \in DReacts$, the initial reaction-SOG of ρ' is defined as an SOG $G' \in ISOGF(\rho')$ satisfying that

- (1) each $M_i(G')$, for any $i = \alpha(\rho'), \dots, \omega(\rho')$, is a terminated strand and the initialized visible-strand of $M_i(\rho')$,
- (2) if $\alpha(\rho') = 0$, $M_0(G')$ is a terminated strand that is the initialized visible-strand of $M_0(\rho')$, and
- (3) $C(G)$ is the initialized context-strand of ρ .

The strands $M_1(G'), \dots, M_{\omega(\rho')}(G')$ are called the output strands of the reaction.

We define the two functions $initial : CReacts \cup DReacts \rightarrow ReactSOGs$ and $final : CReacts \cup DReacts \rightarrow ReactSOGs$ that return, for any reaction $\rho \in CReacts \cup DReacts$, its initial reaction-SOG and its final reaction-SOG, respectively.

The list of all possible states of NSOs is given in Section 3.4.2.

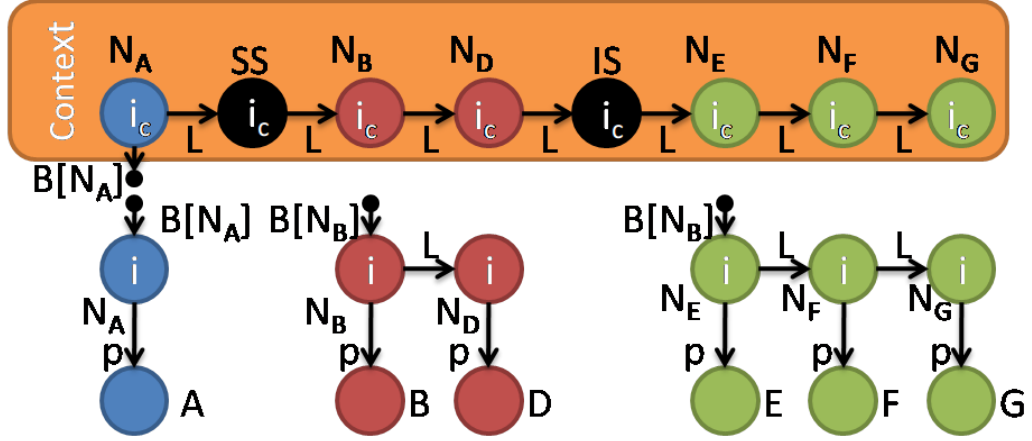
An example of the initial reaction-SOG (Fig. 3.8a) and the final reaction-SOG (Fig. 3.8b) of a composition reaction ρ (Fig. 3.4) are shown in Fig. 3.8, and an example of the initial reaction-SOG (Fig. 3.9a) and the final reaction-SOG (Fig. 3.9b) of a decomposition reaction ρ' (Fig. 3.4) are shown in Fig. 3.9.

Now we will give a set of graph rewriting rules that are locally executed by each NSO to implement catalytic reactions based on our modelling. This set of rules rewrite for any reaction ρ its initial reaction-SOG into its final reaction-SOG. In the following, we first explain informally the logic of our rules. We then define the NSO states used by this set of rules. After that, we list up the required set of rules.

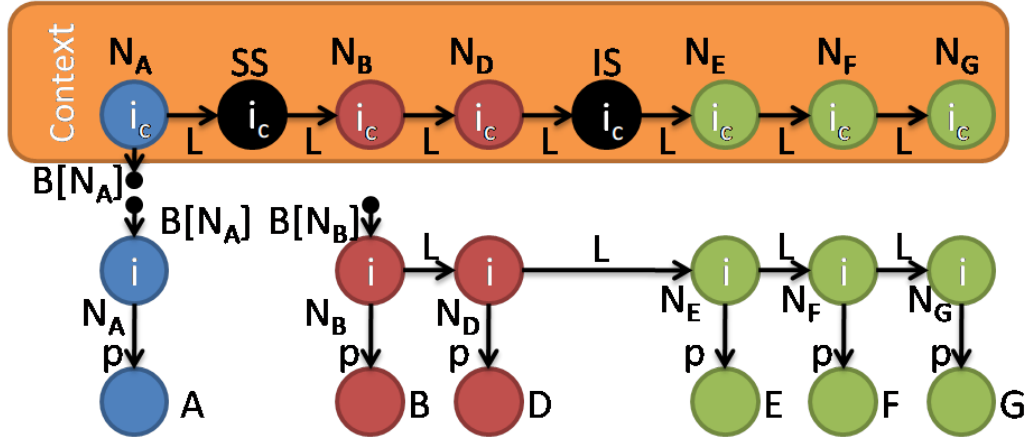
3.4 Graph Rewriting Rules to Implement Reactions

3.4.1 Outline of the process implementing a reaction

The process implementing a reaction is described by a set of graph rewriting rules locally executed by each NSO. Before presenting the states and the graph rewriting rules used by the NSOs to implement this process, we informally explain the general logic of this process. The process implementing a reaction is divided into two stages, i.e., the first stage called the output-strand formation stage, and the

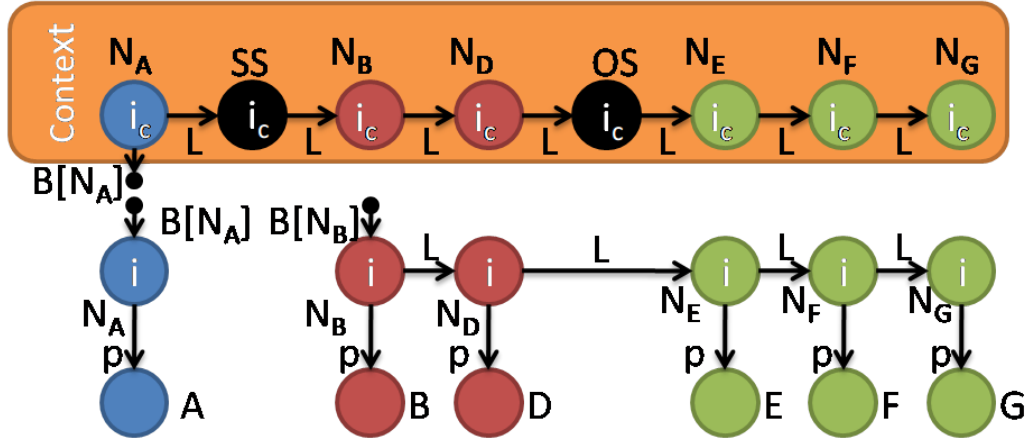


(a) The initial reaction-SOG of the composition reaction ρ

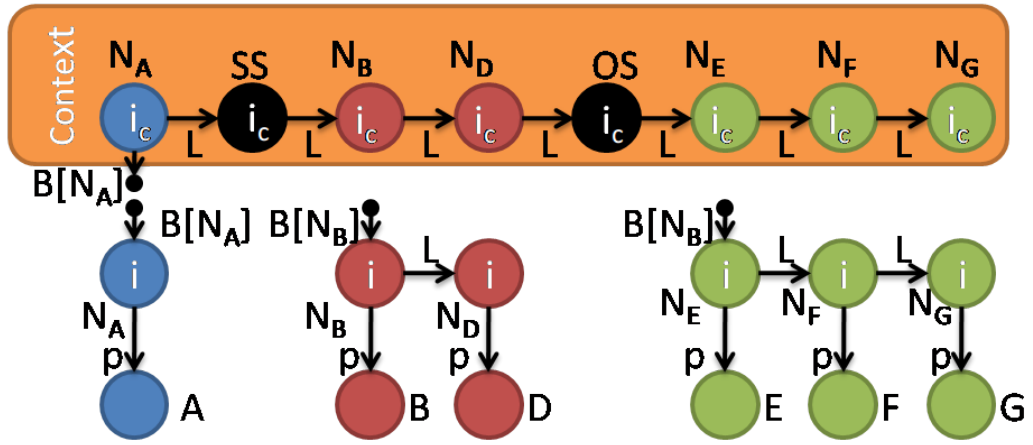


(b) The final reaction-SOG of the composition reaction ρ

Figure 3.8: The initial reaction-SOG and the final reaction-SOG of a composition reaction ρ (Fig. 3.4). The hidden ports are not shown.



(a) The initial reaction-SOG of the decomposition reaction ρ'



(b) The final reaction-SOG of the decomposition reaction ρ'

Figure 3.9: The initial reaction-SOG and the final reaction-SOG of a decomposition reaction ρ' (Fig. 3.5). The hidden ports are not shown.

Chapter 3. Formalization of our Framework

second stage called the reinitialization stage. The first stage itself is divided into four sub-stages, i.e., the beginning stage, the intra-double-strand propagation, the inter-double-strand propagation, and the ending stage. To informally explain each of these sub-stages, we use minimalist representation of the SOG of a reaction. They are minimalist representations because they do not represent the states and the types of the NSOs, nor the types of the connections. However, in this SOG representation, connections represented by horizontal arrows are connections of type L , and connections represented by vertical arrows are connections of type B .

Informally, the first stage corresponds to the propagation of a B connection among the SOG of a reaction. The beginning stage create this B connection to start the propagation. For example, for the reactions ρ and ρ' , the beginning stage rewrites the SOGs shown in Figs. 3.8a and 3.9a into the SOGs shown in Figs. 3.10a and 3.10c, respectively. This B connection is established from the first NSO of the context. After the full propagation, the B connection should be established from the last NSO of the context, as shown in Figs. 3.10b and 3.10d. There are two types of propagation, i.e., the intra-double-strand propagation and the inter-double-strand propagation.

At the beginning of the intra-double strand propagation, the B connection is established from the first NSO of a double-strand. After the propagation, the B connection should be established from the last NSO of this double-strand, as shown in Fig. 3.11. At the beginning of the intra-double strand propagation, the B connection is established from the last NSO of a double-strand. After the propagation, the B connection should be established from the first NSO of the next double-strand, as shown in Fig. 3.12. However, depending of the type of the separator between these two double-strands, this propagation has a different behavior that will be explained in details in Section 4.2. Once the B connection is established from the last NSO of the context, it is necessary to break it, and this is done by the ending stage. After this, the reinitialization stage is executed.

By propagating the B connection, each NSO of the context were connected to its partner NSO in a material strand. This allows us to verify that for each sub-context-strand C_i of a reaction, we have a compatible material-strand M_i . Since we are in a distributed environment and there is no NSO that has knowledge of the full set of NSOs, such a mechanism that are purely based on neighbor interaction is mandatory.

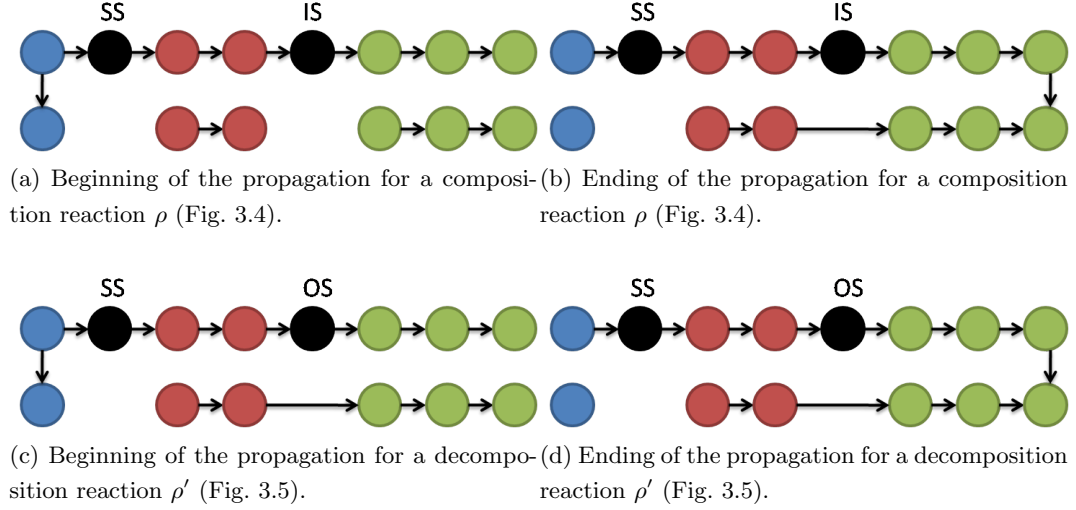


Figure 3.10: Beginning and ending stages.

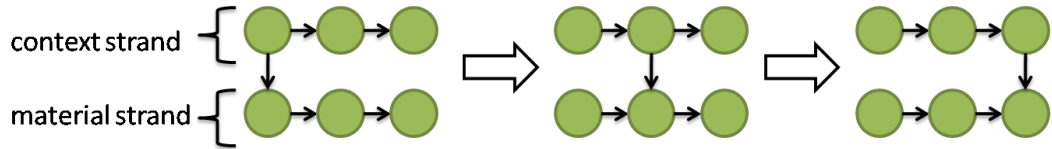


Figure 3.11: Intra-double-strand propagation stage.

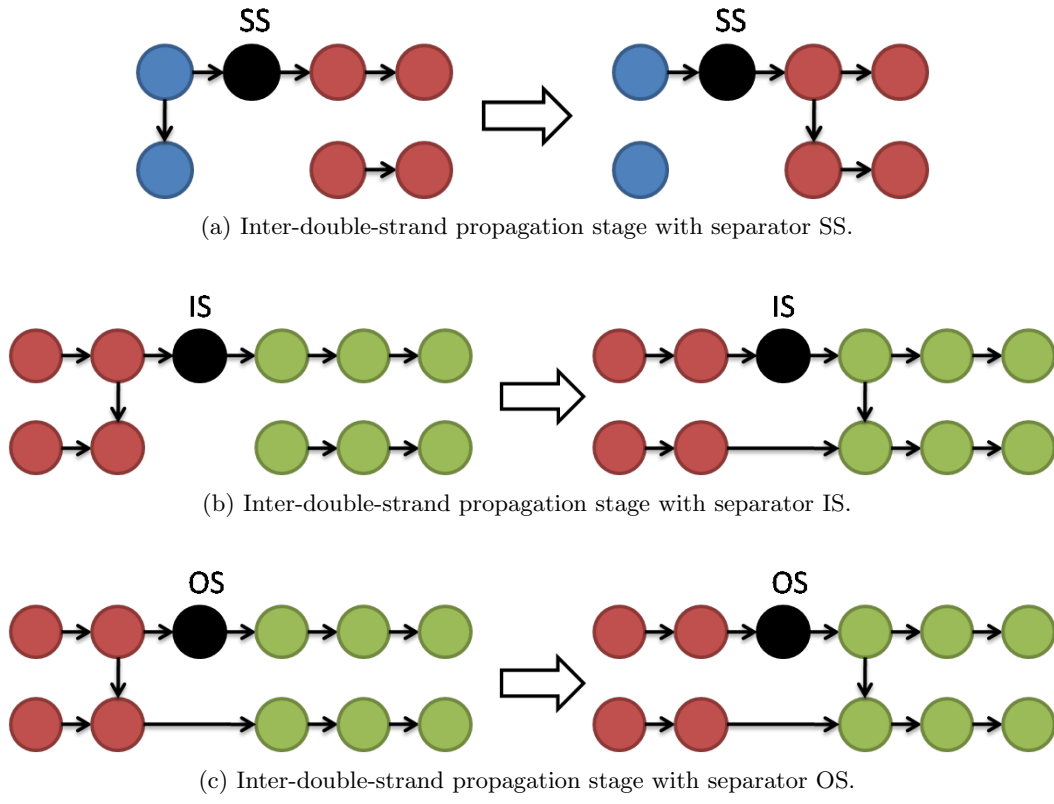


Figure 3.12: Inter-double-strand propagation stage.

3.4.2 States of a Nucleotide Smart Object

In this section, we define the NSO states used by our set of rules. Table 3.3 lists up all the possible states of NSOs. All the NSOs in an initial reaction-SOG are in their initial states, i.e., the state i_c for the context NSOs and the state i for the others. If a context NSO has a connection B to a material NSO, we say that this material NSO is docked to this context NSO. While they are docked, these two NSOs are in the docked state, i.e., the state d_c for the context NSO and the state d for the material NSO. When such a B connection is broken between two NSOs, we say that these NSOs are undocked. The state of a context NSO changes to the state u_c , and the state of a material NSO changes to u , u_e , or u_b . The state n_c is used by the first NSO of a docking-strand instance to denote that it is ready for docking. Finally, once the output strand is formed, it is necessary to reinitialize all the NSOs to their initial states. The state r_c of the context NSOs, and the state r of the material NSOs are used to respectively trigger their reinitialization. There is another resetting state e that is used when an error is detected and it becomes necessary to reset the reaction (see Section 3.4.4).

During a composition reaction process, some L connections are created by the inter-strand-propagation among the basic material strands to form the output strand. In case of some error, it is required to break only these L connections for resetting the reaction. To keep track of these L connections, we used the state u_e . This state has the same role as the states u . However, the state u_e also means that an NSO in this state did not have its $-L$ port connected at the beginning of the reaction. Thus, if the reaction has to be reset (see Section 3.4.4), thanks to these state, we know which L connections we need to break.

During a decomposition reaction process, some L connections have to be broken among the input material strand to form the output strands. In case of some error, these L connections should be memorized to reset the reaction. That is why, during the output-strand formation, the inter-double-strand propagation marks these L connections by using the state u_b . The state u_b has the same role as the state u . However, the states u_b also means that an NSO in this state should break the L connection from itself during the reinitialization stage.

3.4.3 Graph Rewriting Rules for implementing Catalytic Reactions

In Fig. 3.13 and Fig. 3.14, we give the complete list of graph rewriting rules that each NSO repeatedly tries to execute for implementing catalytic reactions in a generic

Table 3.3: States of a Nucleotide Smart Object

For the context NSOs	
i_c :	initialized state
d_c :	docked state
u_c :	undocked state
n_c :	state of the first NSO in a docking strand for initializing the docking of this strand
r_c :	reinitializing state
For the material NSOs	
i :	initialized state
d :	docked state of an NSO that is not the first NSO in a basic material strand
u :	undocked state of an NSO that is not the first NSO, nor the last NSO, of a basic material strand
u_e :	undocked state of the first NSO in a basic material strand
u_b :	undocked state of the last NSO in a basic material strand
r :	reinitializing state
e :	error state of an NSO that is not the first NSO in a basic material strand

way. The process of implementing a catalytic reaction is divided into two stages. The first stage is called the output-strand formation stage. The second stage, is called the reinitialization stage, and it realizes the reinitialization. Some additional rules for the error handling will be given in Chapter 3.4.4.

3.4.4 Error Management Rules

There are two different cases in which the reaction process may fail. The first case may occur due to the loss of a B connection because of the instability of proximity-based connections, and the second case may occur due to the incompatibility between a pair of strands. We assume that the L connections are all stable. In the following two subsections, we will discuss how to deal with these cases.

Broken Connection Management

From the set of rules given in Fig. 3.13 and Fig. 3.14, it is obvious that a B connection can be spanned between two NSOs only when they are in specific states. All the possible combinations are shown in Fig. 3.16a. It is also obvious that at any moment during a reaction process, there exist only one or two B connections.

For the context, the set of rules in Fig. 3.16b repairs all the possible losses of B connections. The rule e1 takes care of such a failure by resetting the NSO to its previous situation just after the execution of the rule 6. The rule e2 takes care of the case in which there are two B connections at the same time, namely the one from the NSO in d_c and the other from the NSO in n_c , by resetting the NSO in n_c and then by changing the state of the NSO in d_c to r_c . The rules e3 and e4 take care of the loss of a connection from an NSO in the state d_c when it is assured that there is only one B connection. They change the state d_c of the NSO to r_c . Changing the state of the NSO to r_c will start the reinitialization phase, and applies the rule 12 repeatedly, and then the rule 11 only once to the focused NSOs.

If a B connection is lost during the output-strand formation, it is necessary to reset all the NSOs of the strand and eventually to break the L connections that may have been already established by the rules 5, 6 and 7. That is why the rules 7 set each NSO from which the L connection is made at the state u_e instead of u . The states u_e indicates the NSOs that did not have a $-L$ connection in the beginning of the reaction. The set of rules in Fig. 3.16c resets the material strands when a B connection is lost. The rule e5 takes care of the case when the B connection is lost at the first NSO of the first material strand, i.e., the instance of M_1 if it is a reaction without a stimulus, or the instance of M_0 otherwise. In this case, it is necessary

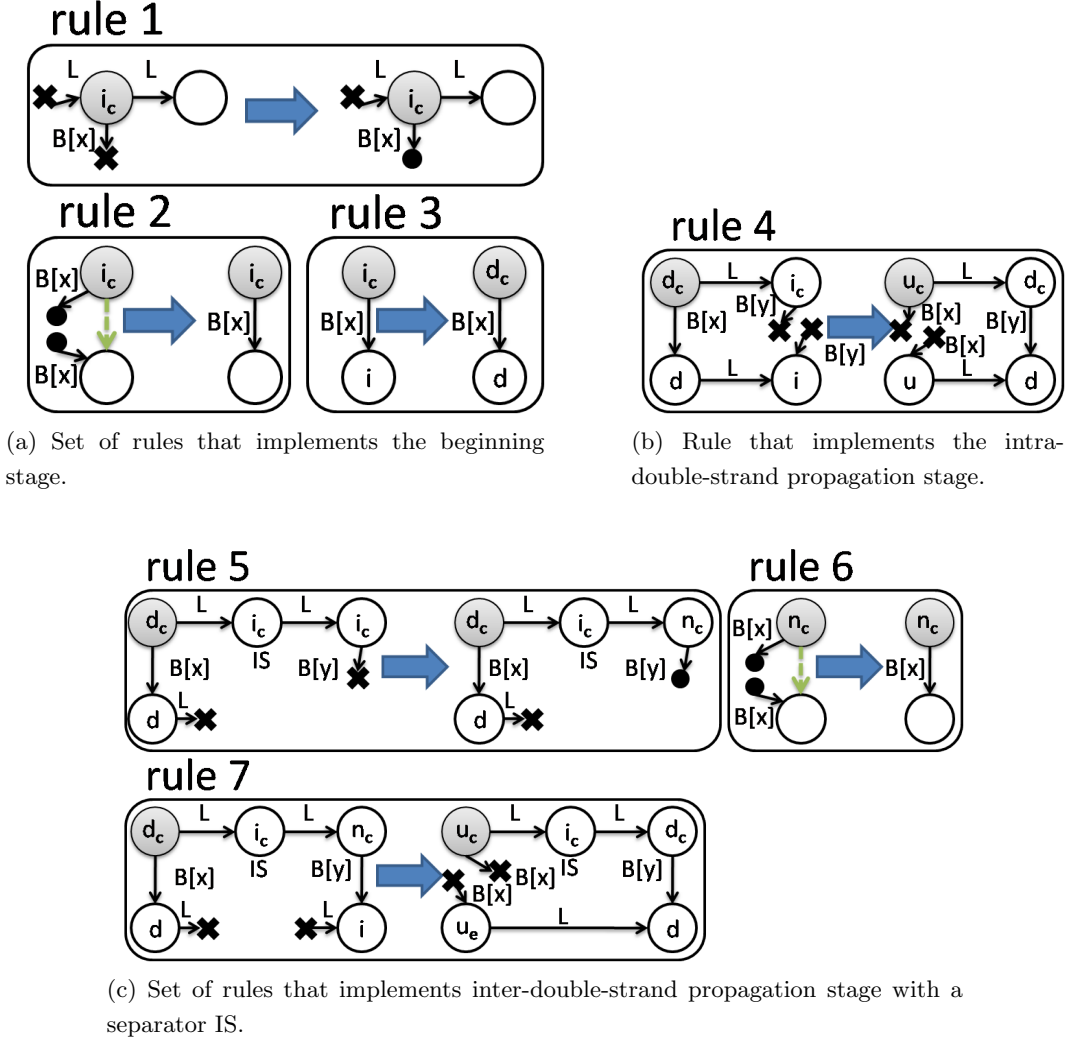
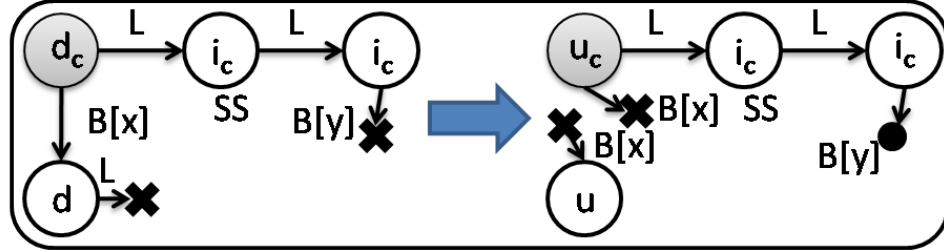


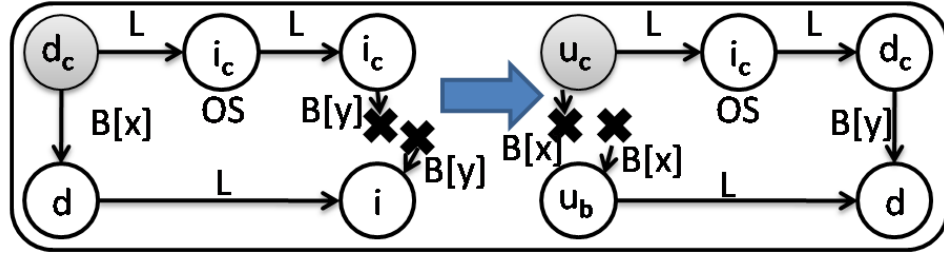
Figure 3.13: Set of Graph Rewriting rules that implement the beginning stage, the intra-double-strand propagation stage, and the inter-double-strand propagation stage of the output-strand formation. When an NSO has no specified type, it means that it is of type T . For readability, the $+B$ ports of the context NSOs and the $-B$ ports of the material NSOs are not shown since they are always hidden.

rule 8



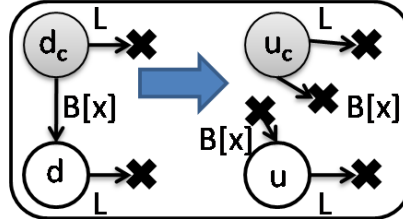
(a) Rule that implements inter-double-strand propagation stage with a separator SS.

rule 9



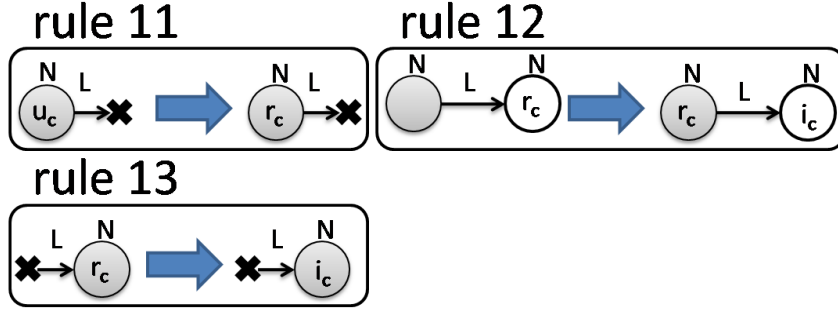
(b) Rule that implements inter-double-strand propagation stage with a separator OS.

rule 10

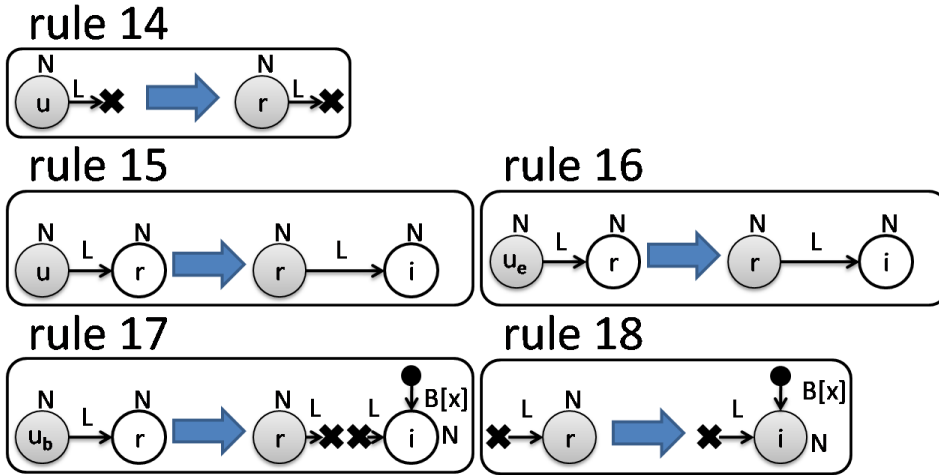


(c) Rule that implement the ending stage of the output-strand formation.

Figure 3.14: Set of Graph Rewriting rules that implement the inter-double-strand propagation stage with a separator SS or a separator OS, and ending stage of the output-strand formation. When an NSO has no specified type, it means that it is of type T . For readability, the $+B$ ports of the context NSOs and the $-B$ ports of the material NSOs are not shown since they are always hidden.



(a) Set of rules that processes the reinitialization of the context



(b) Set of rules that processes the reinitialization of the input strands and of the context

Figure 3.15: Set of Graph Rewriting rules that process the reinitialization. When an NSO has no specified type, it means that it is of type T . For readability, the $+B$ ports of the context NSOs and the $-B$ ports of the material NSOs are not shown since they are always hidden

only to set the B port visible. The rule e6 take care of the loss of the B connection in the other cases. The NSOs will be set to their resetting state e , that will trigger a resetting process which then iteratively selects and applies an appropriate rule out of the rules e7, e8, or e9, and finally applies rule e10. This process resets all the NSOs and breaks all the L connections that did not exist at the beginning of the reaction. The program codes, that supposed to span necessary connections among application SOs, must be deleted when the state of a NSO change its state to e .

Incompatible Material-Strands

For any reaction $\rho \in CReacts$, and for each docking strand $C_i(\rho) = c_1^i \rightarrow \dots \rightarrow c_{k_i}^i$ where $\alpha(\rho) \leq i \leq \omega(\rho)$, there are three possibilities of having an incompatible material-strand $V : v_1 \rightarrow \dots \rightarrow v_n$. The first case may occur for such a V satisfying that $\exists l (l \leq k_i \wedge l \leq n \wedge type(c_l^i) \neq type(v_l))$. We say that V is a type incompatible material strand. An example of this case is shown in Fig. 3.17a. The second case may occur for such a V satisfying that $k_i > n$. We say that V is a shorter material strand. An example of this case is shown in Fig. 3.17b. The third case may occur for such a V satisfying that $n > k_i$. We say that V is a longer material strand. Two example of this case are shown in Fig. 3.17c and 3.17d.

For any reaction $\rho' \in DReacts$, such that $M = M_1 \rightarrow \dots \rightarrow M(\omega(\rho')) = m_1 \rightarrow m_2 \rightarrow \dots \rightarrow m_k$, there are three possibilities of having an incompatible material-strand $V : v_1 \rightarrow \dots \rightarrow v_n$. The first case may occur for such a V satisfying that $\exists l (l \leq k \wedge l \leq n \wedge type(m_l) \neq type(v_l))$. We say that V is a type incompatible material strand. The second case may occur for such a V satisfying that $k > n$. We say that V is a shorter material strand. The third case may occur for such a V satisfying that $n > k$. We say that V is a longer material strand.

Our rules shown in Fig. 3.18 take care of all these kinds of incompatible material-strands by breaking the B connections that is established from an NSO of the context to the last NSO of an incompatible. Such a breaking operation triggers the set of rules given in Section 3.4.4 and resets the context and the material strands to their initial states.

3.5 Graph Rewriting rules to build a Context

Previously, we assumed that the context NSOs of a reaction are already federated to form the context strand of this reaction. We also assumed that this strand is initialized before processing a reaction. We gave the set of rules in Fig. 3.15a that

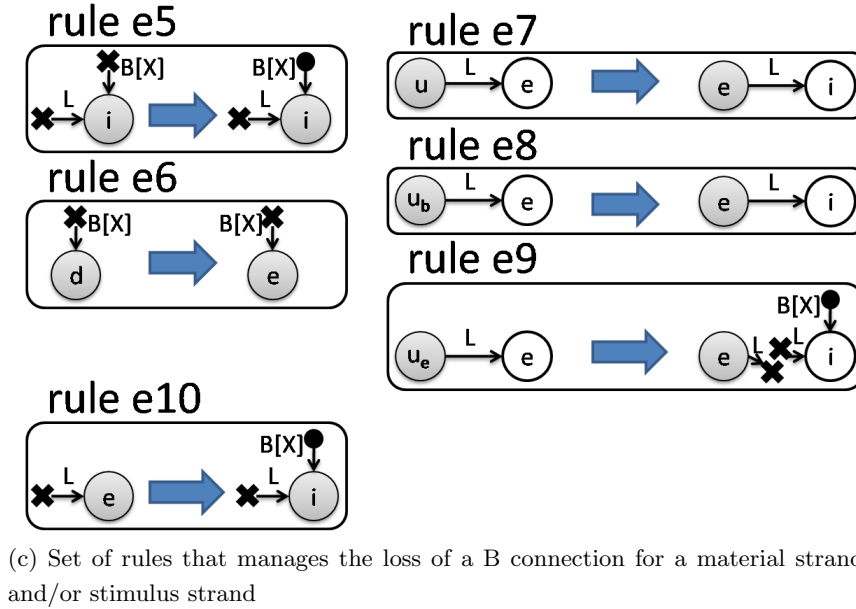
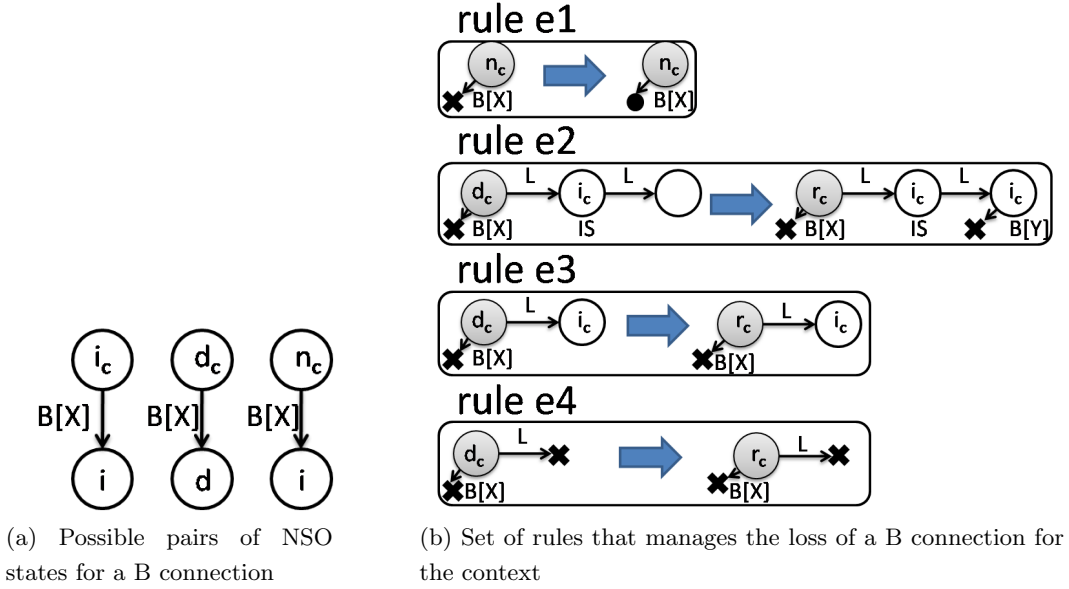


Figure 3.16: Set of rules to take care of the loss of B connections

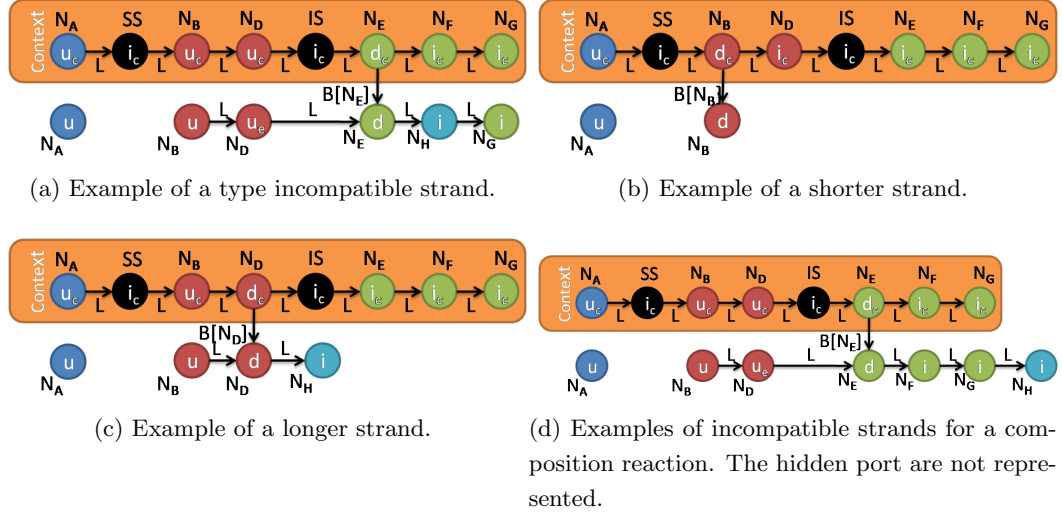


Figure 3.17: Examples of incompatible strands for a composition reaction.

reinitializes a context strand to be able to process the reaction more than once. However, we did not explain how to build a context. When an NSO of type T is turned on, it is in the configuration shown in the Fig. 3.19a. This configuration corresponds to an initialized material-strand of size 1. When an NSO of type S is turned on, it is in the configuration shown in the Fig. 3.19b. Then by applying the rules show in Fig. 3.20, it is possible for a user to build a context just by interacting physically with the smart objects.

This set of graph rewriting rules extends the definition that we gave in Section 3.2 with a new condition that corresponds to a user interaction. Such a condition can be for example to press a button. This user interaction should only happen on the activated node and is represented graphically by a label on top of the big arrow that separates the left-hand side from the right hand side of the rule. The two different user interactions used in this set of rules are “action 1” and “action 2”. We keep these interactions abstract since a user interaction depends on the hardware implementing the smart object. For example, if a smart object has two buttons, the easiest ways to implement these two interactions would be to map each interaction to one button.

When we are building a context, the NSOs are in their building state, i.e., the state b . The rules B1, B2, B3, B4, B5, and B6 allow us to put an L port in the state visible. By putting visible the $-L$ port of a first NSO and the $+L$ port of a second NSO, we will be able to trigger the rule B7 to create a connection L from the first

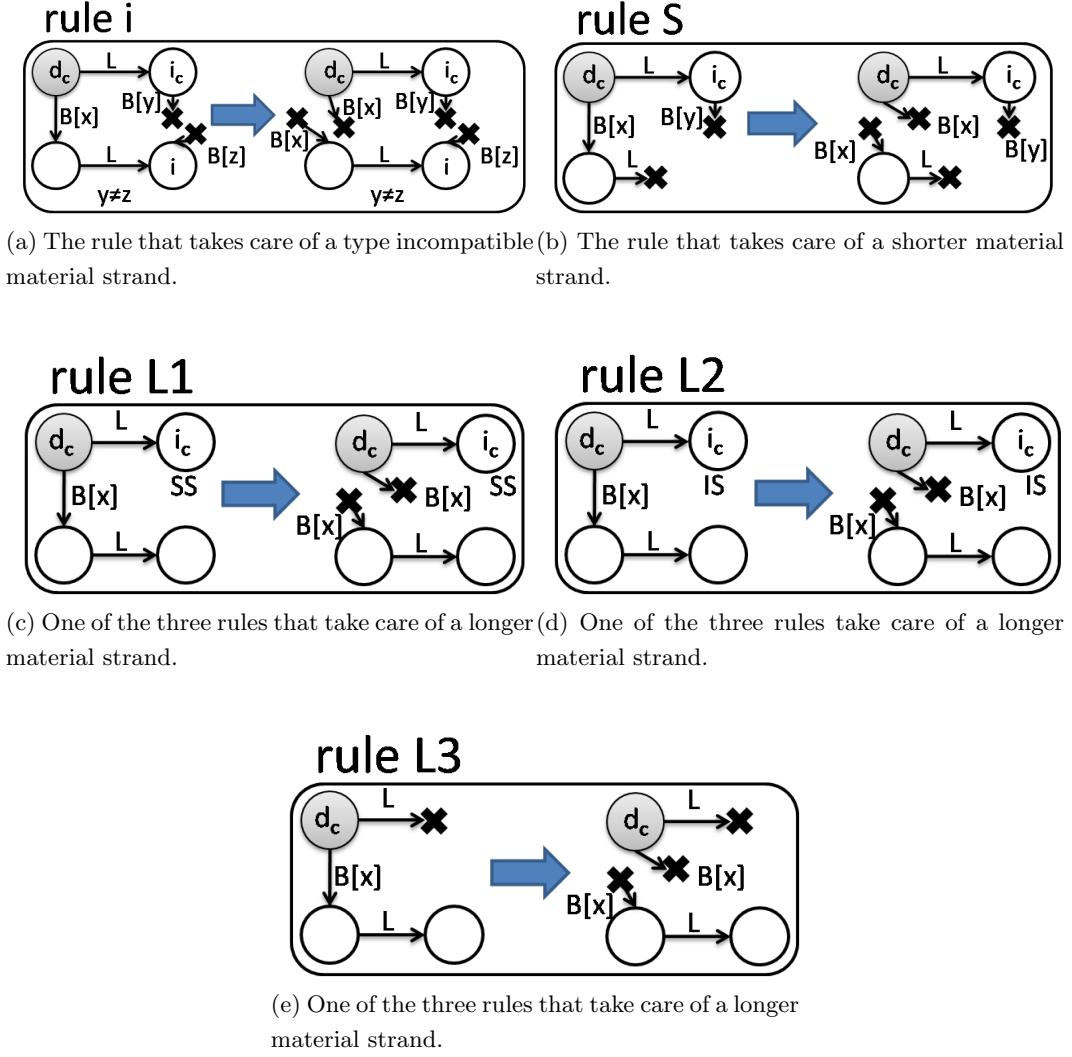


Figure 3.18: Set of rules to take care of incompatible material strands. The hidden port are not represented.



(a) Initial configuration of an NSO of type T . (b) Initial configuration of an NSO of type S .

Figure 3.19: Initial configuration of the NSOs.

NSO to the second NSO. Once we created all the necessary L connections and we obtain a terminated strand corresponding to the context strand of a reaction, we can initialize the context by using the “action 1” on the last NSO of this strand. The rule B8 will then trigger and put the last NSO of the context strand into the state

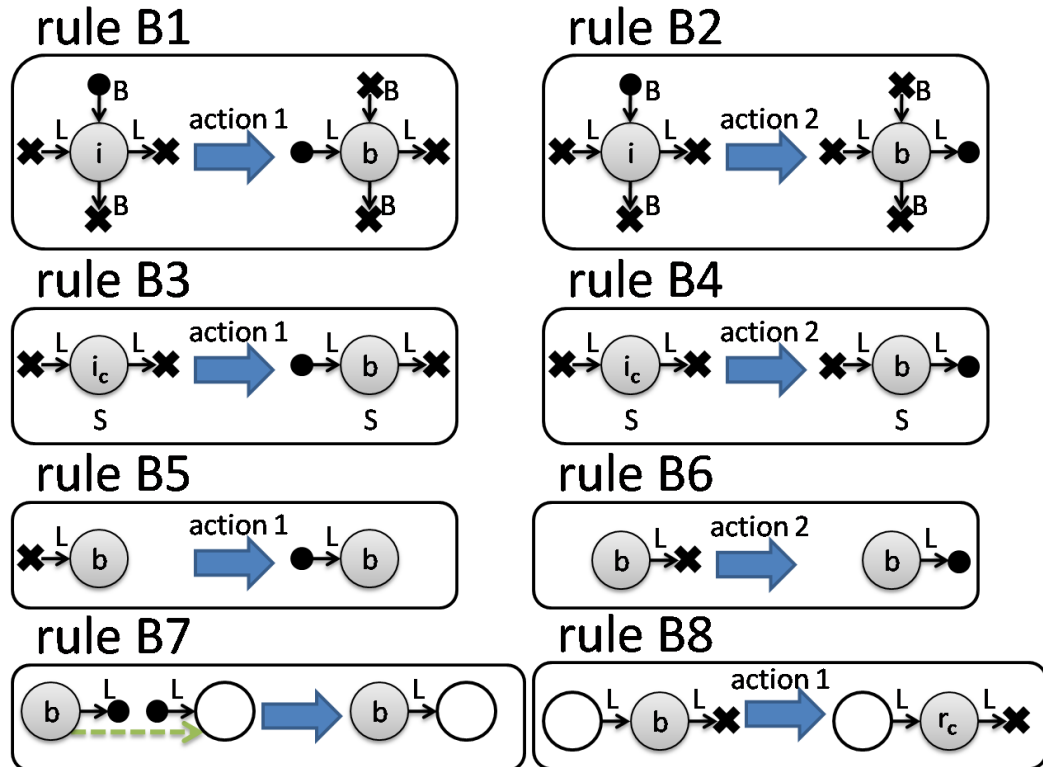


Figure 3.20: Set of graph rewriting rules to interactively build a context.

Chapter 3. Formalization of our Framework

r_c . This will have for effect to trigger the set of rules presented in Fig. 3.15a that will initialize the context strand. It is possible to come with a different mechanism to build a context. However, the one presented here is simple and can be implemented with simple smart objects that do not have a screen or a keyboard. This mechanism allows us to quickly create a context strand, and thus to add a reaction in a catalytic reaction network. It is not necessary to shut down or to modify the other contexts in the catalytic reaction network.

A set of rules to build material strands is not needed. Indeed, when we turn on NSOs they correspond to an initialized material strand of size 1. Bigger material strand can be built by using composition reactions using these NSOs.

Chapter 4

Validity Proof of Rules for Reactions

It is rather easy to prove that our graph rewriting system presented in Section 3.4.3 works correctly for each specific catalytic reaction. Here, we prove that our system works correctly for any catalytic reaction. For proving this, we need to show that there exists a sequence of rule applications that, for any reaction $\rho \in CReacts \cup DReacts$, rewrites the initial reaction-SOG $initial(\rho)$ into the final reaction-SOG $final(\rho)$. We also need to show that this sequence is deterministic, namely, there is no other possible sequence of rule applications. As mentioned before, the reaction process is divided into two stages, i.e., the first stage called the output-strand formation stage, and the second stage called the reinitialization stage. In the following, we will explain and prove these two stages for any reaction.

In the following, we assume that the L connections are stable. They are assumed to use, for example, mobile phone connections or wired connections. We assume that the B connections are proximity-based and may be broken due to the mobility of the SOs, for example when a material strand NSO goes out of the scope of a context NSO to which it was once docked. However, for proving the correctness of rules for the reaction process we will first assume that the B connections are also stable. Then, in Section 4.6, we will prove the process handling the errors that may occur during the reaction process, including the loss of a B connection.

4.1 Output-strand formation stage

Before defining the output-strand formation, we will define some specific key instances of a double-strand that will help us to describe and prove this process, and give the definition of the unbridged reaction-SOG of a reaction.

Definition 25 (Double-strand instances). *For any reaction $\rho \in CReacts \cup DReacts$, let D be a double-strand of size $n \geq 1$ of the template SOG of ρ . Then, an instance of this double-strand D is an instance SOG D' satisfying that D is isomorphic to D' . The size of an instance D' of a double-strand D is the size of D .*

We consider the following possible instances of a double-strand, namely, ‘visible-initialized’, ‘hidden-initialized’, ‘simple-unbridged’, ‘established-unbridged’, ‘breaking-unbridged’, ‘ k -th-pair-bridged’, and ‘unbridged’ double-strand instances. An SOG G' that is the ‘visible-initialized’ instance, the ‘hidden-initialized’ instance, the ‘simple-unbridged’ instance, the ‘established-unbridged’ instance, the ‘breaking-unbridged’ instance, the ‘ k -th-pair-bridged’ instance, or the ‘unbridged’ instance of a double-strand D is called an ‘visible-initialized’, ‘hidden-initialized’, ‘simple-unbridged’, ‘established-unbridged’, ‘breaking-unbridged’, ‘ k -th-pair-bridged’, and ‘unbridged’ double-strand, respectively. Figs. 4.1, 4.2 and 4.3 shows these instances.

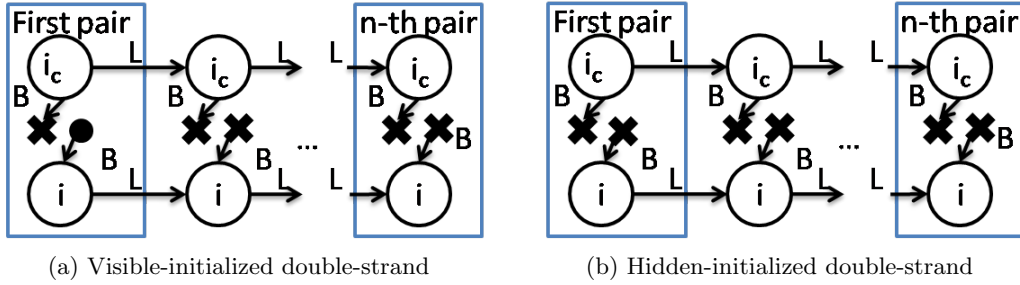


Figure 4.1: Initialized double-strands of size n

More formally, the different instances of double-strand are defined as follows.

Definition 26 (Visible-initialized double-strand). *A visible-initialized double-strand is a double-strand instance $D = \langle C, M \rangle$ of size n such that the following hold.*

- (1) *All the NSOs of the strand C are in the state i_c and have their $-B$ port hidden.*
- (2) *All the NSOs of the strand M are in the state i and have their $+B$ port hidden except for the first one that has its $+B$ port visible.*

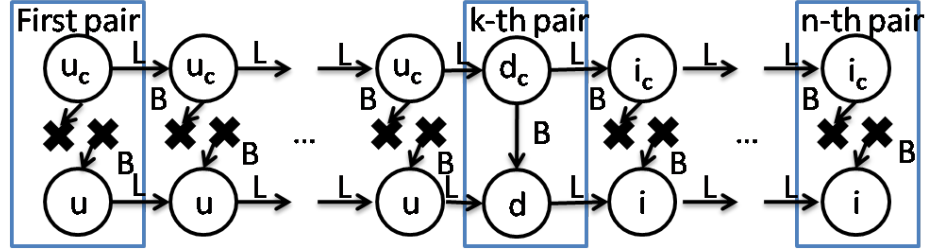


Figure 4.2: k -th-pair-bridged double-strand of size n .

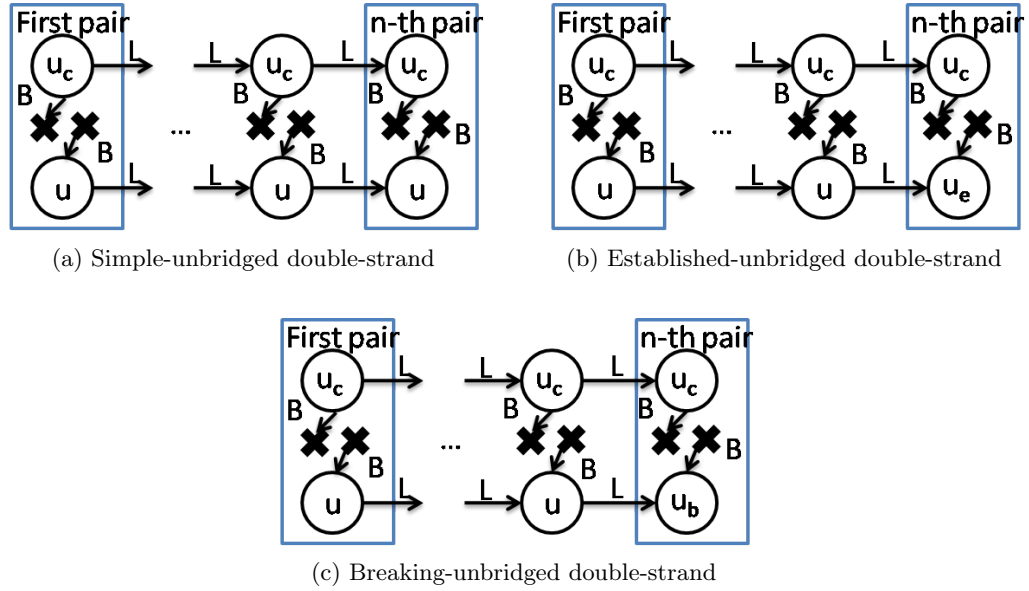


Figure 4.3: Undocked double-strands of size n

Chapter 4. Validity Proof of Rules for Reactions

Definition 27 (Hidden-initialized double-strand). *A hidden-initialized double-strand is a double-strand instance $D = \langle C, M \rangle$ of size n such that the following hold.*

- (1) *All the NSOs of the strand C are in the state i_c and have their $-B$ port hidden.*
- (2) *All the NSOs of the strand M are in the state i and have their $+B$ port hidden.*

Definition 28 (k -th-pair-bridged double-strand). *A k -th-pair-bridged double-strand is a double-strand instance $D = \langle C, M \rangle$ of size $n \geq k$ where $C = c_1 \rightarrow c_2 \rightarrow \dots \rightarrow c_n$ and $M = m_1 \rightarrow m_2 \rightarrow \dots \rightarrow m_n$ such that the following hold.*

- (1) *The NSO c_k is in the state d_c .*
- (2) *The NSO m_k is in the state d .*
- (3) *Each NSO c_i for $i = 1, \dots, k-1$ is in the state u_c .*
- (4) *Each NSO m_i for $i = 1, \dots, k-1$ is in the state u .*
- (5) *Each NSO c_j for $j = k+1, \dots, n$ is in the state i_c .*
- (6) *Each NSO m_j for $j = k+1, \dots, n$ is in the state i .*
- (7) *All the NSOs of the strand C have their $-B$ port hidden except for the NSO c_k that has its $-B$ port visible and connected.*
- (8) *All the NSOs of the strand M have their $+B$ port hidden except for the NSO m_k that has its $+B$ port visible and connected.*
- (9) *There is a connection of type B from the NSO c_k to the NSO m_k .*

Definition 29 (Simple-unbridged double-strand). *A simple-unbridged double-strand is a double-strand instance $D = \langle C, M \rangle$ of size n satisfying the followings.*

- (1) *All the NSOs of the strand C are in the state u_c and have their $-B$ port hidden.*
- (2) *All the NSOs of the strand M are in the state u and have their $+B$ port hidden.*

Definition 30 (Established-unbridged double-strand). *An established-unbridged double-strand is a double-strand instance $D = \langle C, M \rangle$ of size n satisfying the followings.*

- (1) *All the NSOs of the strand C are in the state u_c and have their $-B$ port hidden.*

- (2) All the NSOs of the strand M are in the state u and have their $+B$ port hidden except for the last one that is in the state u_c .

Definition 31 (Breaking-unbridged double-strand). A breaking-unbridged double-strand is a double-strand instance $D = \langle C, M \rangle$ of size n satisfying the followings.

- (1) All the NSOs of the strand C are in the state u_c and have their $-B$ port hidden.
- (2) All the NSOs of the strand M are in the state u and have their $+B$ port hidden except for the last one that is in the state u_b .

It is easy to observe that:

- There is no activated node in an initialized double-strand, unless the NSO in the state i_c of the first pair has its $+L$ port hidden.
- There is only one activated node in a k -th-pair-bridged double-strand. This activated node is the NSO in the state d_c .
- There is no activated node in an unbridged double-strand, unless the NSO in the state u_c of the pair n has its $-L$ port hidden.

The n -th-pair-bridged instance of a double-strand of n is also called the last-pair-bridged instance.

Definition 32 (The unbridged reaction-SOG of a reaction). For all reaction $\rho \in CReacts \cup DReacts$, the unbridged reaction-SOG of ρ is an SOG $G \in ISOGF(\rho)$ satisfying the following conditions.

- (1) If $\rho \in CReacts$, each double-strand $D_1(G), \dots, D_{\omega(\rho)-1}(G)$ is an established-unbridged double-strand. Otherwise, if $\rho \in DReacts$, each double-strand $D_1(G), \dots, D_{\omega(\rho)-1}(G)$ is a breaking-unbridged double-strand.
- (2) The double-strand $D_{\omega(\rho)}(G)$ is a simple-unbridged double-strand,
- (3) If $\alpha(\rho) = 0$, then $D_0(G)$ is a simple-unbridged double-strand, and $M_0(G)$ is a terminated strand,
- (4) all the separators in G are in the state i_c , and
- (5) the strands $M_1(G), \dots, M_{\omega(\rho)}(G)$ form a terminated strand $M_1(G) \rightarrow \dots \rightarrow M_{\omega(\rho)}(G)$.

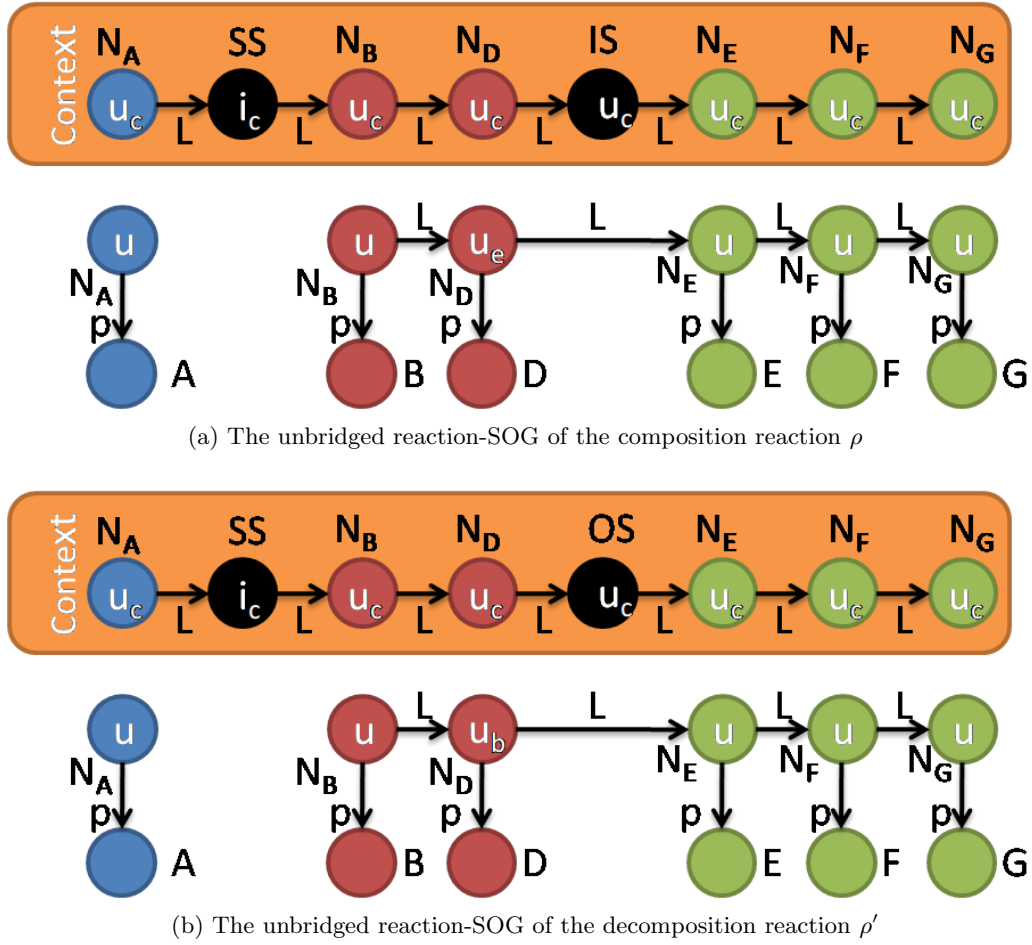


Figure 4.4: The unbridged reaction-SOGs of the composition reaction ρ (Fig. 3.4) and of the decomposition reaction ρ' (Fig. 3.5). The hidden ports are not shown.

Such an SOG G is called the unbridged reaction-SOG of the reaction ρ . An example of the unbridged reaction-SOG of a composition reaction and of a decomposition reaction are shown in Figs. 4.4a and 4.4a, respectively.

We define a function *unbridged* : $CReacts \cup DReacts \rightarrow ReactSOGs$ that returns, for any reaction $\rho \in CReacts \cup DReacts$, its unbridged reaction-SOG.

From the definition of the double-strands, it is easy to observe that there is only one activated node in any unbridged reaction-SOG G , and this node is the last NSO of the strand $C(G)$.

The output-strand formation rewrites the initial reaction-SOG *initial*(ρ) into the unbridged reaction-SOG *unbridged*(ρ) for any reaction $\rho \in CReacts \cup DReacts$.

4.2 Proof of the Validity of the Output-Strand Formation

Before proving the validity of the output-strand formation itself, we will first define the bridged reaction-SOG of a reaction $\rho \in CReacts \cup DReacts$, give the following assumptions, and prove the following lemmas.

4.2.1 Definitions

Definition 33 (Bridged reaction-SOGs of a reaction).

For any reaction $\rho \in CReacts$, for any $h = \alpha(\rho), \dots, \omega(\rho)$, and for any $l = 1, \dots, size(\rho, h)$, $\rho_l^h \in ISOGF(\rho)$ denotes an SOG defined as follows:

- (1) If $\alpha(\rho) = 0$, and if $h > 0$, then $D_0(\rho_l^h)$ is a simple-unbridged double-strand, and $M_0(\rho_l^h)$ is a terminated strand.
- (2) Each double-strand $D_1(\rho_l^h), \dots, D_{h-1}(\rho_l^h)$ is an established-unbridged double-strand.
- (3) The strands $M_1(\rho_l^h), \dots, M_h(\rho_l^h)$ form a terminated strand $M_1(\rho_l^h) \rightarrow \dots \rightarrow M_h(\rho_l^h)$.
- (4) The double-strand $D_h(\rho_l^h)$ is an l -th-pair-bridged double-strand.
- (5) Each double-strands $D_{h+1}(\rho_l^h), \dots, D_{\omega(\rho)}(\rho_l^h)$ is a visible-initialized double-strand.
- (6) All the separators in ρ_l^h are in the state i_c .

Chapter 4. Validity Proof of Rules for Reactions

For any reaction $\rho \in DReacts$, for any $h = \alpha(\rho), \dots, \omega(\rho)$, and for any $l = 1, \dots, size(\rho, h)$, $\rho_l^h \in ISOGF(\rho)$ denotes an SOG defined as follows:

- (1) If $\alpha(\rho) = 0$, and if $h > 0$, then $D_0(\rho_l^h)$ is a simple-unbridged double-strand, and $M_0(\rho_l^h)$ is a terminated strand.
- (2) Each double-strand $D_1(\rho_l^h), \dots, D_{h-1}(\rho_l^h)$ is a breaking-unbridged double-strand.
- (3) The double-strand $D_h(\rho_l^h)$ is an l -th-pair-bridged double-strand.
- (4) Each double-strand $D_{h+1}(\rho_l^h), \dots, D_{\omega(\rho)}(\rho_l^h)$ is a hidden-initialized double-strand.
- (5) The strands $M_1(\rho_l^h), \dots, M_{\omega(\rho)}(\rho_l^h)$ form a terminated strand $M_1(\rho_l^h) \rightarrow \dots \rightarrow M_{\omega(\rho)}(\rho_l^h)$.
- (6) All the separators in ρ_l^h are in the state i_c .

Such an SOG ρ_l^h is called a bridged reaction-SOG of the reaction ρ . If $l = 1$, ρ_l^h is also called a leftmost-bridged reaction-SOG. If $l = size(\rho, h)$, ρ_l^h is also called a rightmost-bridged reaction-SOG. An example of the unbridged reaction-SOG of a composition reaction and of a decomposition reaction are shown in Figs. 4.4a and 4.4a, respectively.

4.2.2 Assumptions “No rule 14” and “Simple Proximity”

For the simplicity of our following discussion, let us temporally assume that the following assumptions hold true.

Assumption 4.2.1 (No rule 14). *For any reaction $\rho \in CReacts \cup DReacts$, for any $h = \alpha(\rho), \dots, \omega(\rho)$, for any $l = 1, \dots, size(\rho, h)$, and for any $\rho_l^h \in ISOGF(\rho)$, let us assume that the rule 14 (Fig. 3.15) cannot rewrite the bridged reaction-SOG ρ_l^h .*

The rule 14 triggers the reinitialization of a material strand. In other words, if Assumption 4.2.1 holds, the stimulus strands of a reaction will be able to be reinitialized only after the output-strand formation of this reaction is completed. In the following proof of the validity of the output-strand formation, we will assume that Assumption 4.2.1 holds. We discuss in Section 4.7 what happens if this assumption does not hold.

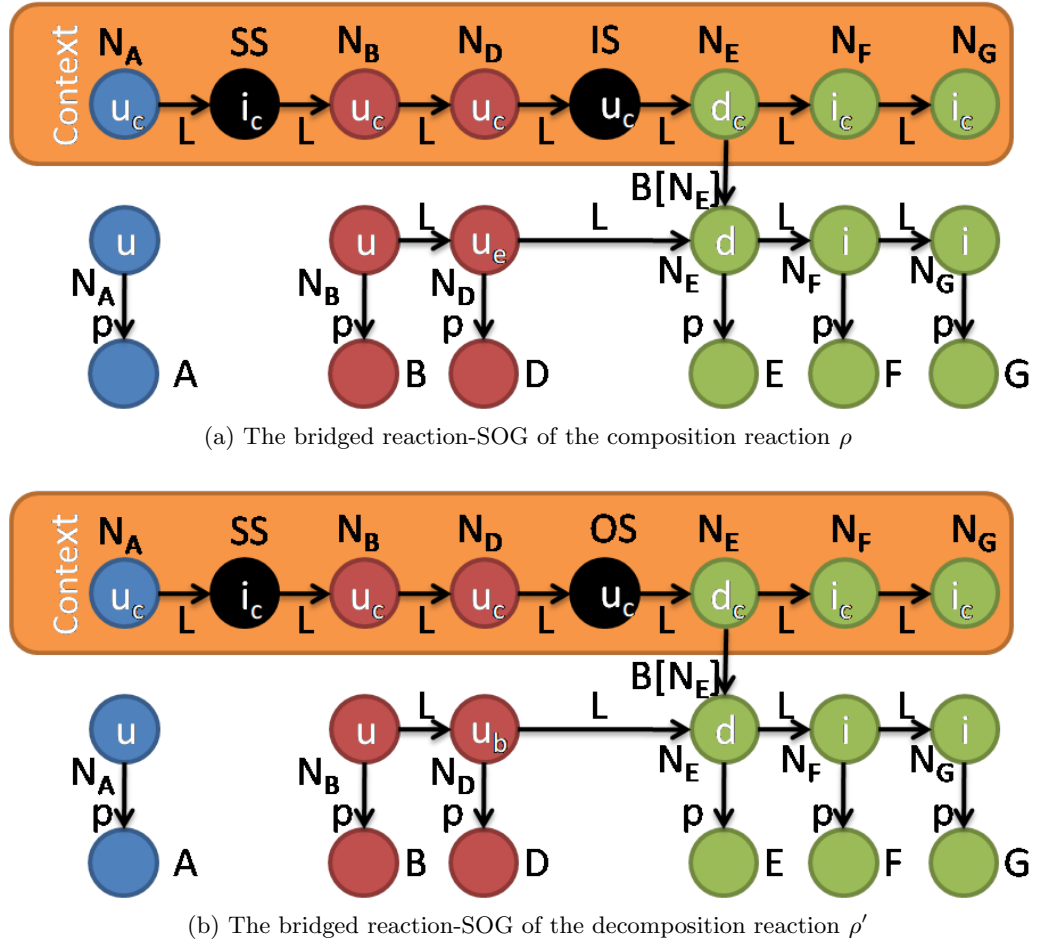


Figure 4.5: The bridged reaction-SOGs of the composition reaction ρ (Fig. 3.4) and of the decomposition reaction ρ' (Fig. 3.5). The hidden ports are not shown.

Assumption 4.2.2 (Simple Proximity). *For any reaction $\rho \in CReacts \cup DReacts$, for any instance SOG $G \in ISOGF(\rho)$, and for any $i \in \{\alpha(\rho), \dots, \omega(\rho)\}$, let us assume that the j -th NSO of each $M_i(\rho)$ is only in the proximity of the j -th NSO of $C_i(\rho)$, and that the j -th NSO of each $C_i(\rho)$ has in its proximity only the j -th NSO of $M_i(\rho)$, for any $j \in \{1, \dots, size(M_i)\}$.*

We also discuss in Section 4.7 what happens if this assumption does not hold.

From the definition of the double-strands, if Assumption 4.2.1 holds, it is easy to observe that there is only one activated node in any bridged reaction-SOG ρ_l^h , and this node is the l -th NSO of the strand $C_h(G)$.

4.2.3 Lemmas and Theorem

Lemma 4.2.3. *Suppose that each connection is stable once it is established, and that both Assumptions 4.2.1 and 4.2.2 hold. Then, for any reaction $\rho \in CReacts \cup DReacts$, for any $h = \alpha(\rho), \dots, \omega(\rho)$, and for any $n = 1, \dots, size(\rho, h)$ the bridged reaction-SOG ρ_1^h is always uniquely rewritten into the reaction-SOG ρ_n^h by the set of rewriting rules in Figs. 3.13, 3.14, 3.15, 3.16, and 3.17.*

Proof. We prove Lemma 4.2.3 by induction.

When $n = 1$, it is obvious that Lemma 4.2.3 holds.

Suppose that Lemma 4.2.3 is true for $n = k$, where $1 \leq k \leq size(\rho, h) - 1$. Then for $n = k + 1$, ρ_1^h is always uniquely rewritten into ρ_k^h . It is obvious that the only activated node in ρ_k^h is the k -th NSO of $C_h(\rho_k^h)$. This implies that only the rule 4 can be applied. By applying the rule 4, $D_h(\rho_k^h)$ is uniquely rewritten into $D_h(\rho_{k+1}^h)$ as shown in Fig. 4.6, and hence ρ_k^h is always uniquely rewritten into ρ_{k+1}^h . Therefore, Lemma 4.2.3 is true for $n = k + 1$.

By the principle of induction, Lemma 4.2.3 is true for all $n = 1, \dots, size(\rho, h)$. $\square$

Lemma 4.2.4. *Suppose that each connection is stable once it is established, and that both Assumptions 4.2.1 and 4.2.2 hold. Then, for any reaction $\rho \in CReacts \cup DReacts$, and for any $h = \alpha(\rho), \dots, \omega(\rho) - 1$, the rightmost-bridged reaction-SOG ρ_m^h where $m = size(\rho, h)$ is always uniquely rewritten into the leftmost-bridged reaction-SOG ρ_1^{h+1} by the set of rules in Figs. 3.13, 3.14, 3.15, 3.16, and 3.17.*

Proof. There are three different cases depending on whether it holds that $s_h(\rho_m^h) = IS$, $s_h(\rho_m^h) = SS$, or $s_h(\rho_m^h) = OS$. Since $D_h(\rho_m^h)$ is a rightmost-bridged reaction-SOG, it is obvious for each of these cases that the only activated node in ρ_m^h is the

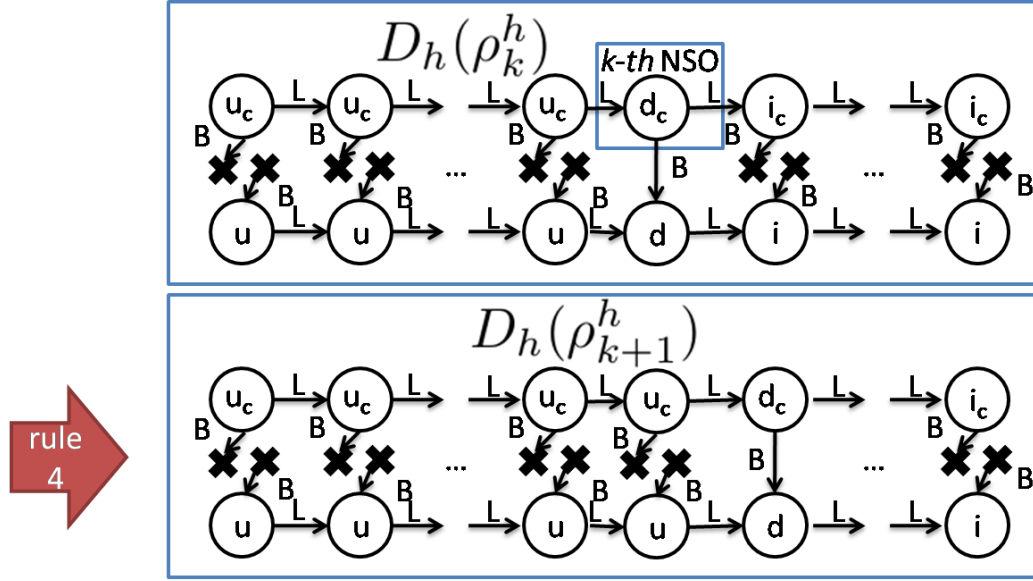


Figure 4.6: Illustration of Lemma 4.2.3.

last node of the strand $C_h(\rho_m^h)$ of the double-strand $D_h(\rho_m^h)$. Because this node is the only activated node, it is obvious that for each of these cases there is only one possible sequence of rule applications. Since this sequence uniquely rewrites $D_h(\rho_m^h)$ into $D_h(\rho_1^{h+1})$, and $D_{h+1}(\rho_m^h)$ into $D_{h+1}(\rho_1^{h+1})$, ρ_m^h is uniquely rewritten into ρ_1^{h+1} . We now show the three different cases together with their unique rule application sequences.

- (1) For the case when $s_h(\rho_m^h) = IS$, the only applicable sequence of rules is 5, 6, and 7 in this order.
- (2) For the case when $s_h(\rho_m^h) = SS$, the only applicable sequence of rules is 8, 2, and 3 in this order.
- (3) For the case when $s_h(\rho_m^h) = OS$, the only applicable rule is 9.

In each of these cases, the rule application sequence is unique. Therefore, Lemma 4.2.4 holds. $\square$

Lemma 4.2.5. *Suppose that each connection is stable once it is established, and that both Assumptions 4.2.1 and 4.2.2 hold. Then, for any reaction $\rho \in CReacts \cup DReacts$, and for any $n = \alpha(\rho), \dots, \omega(\rho)$, the leftmost-bridged reaction-SOG $\rho_1^{\alpha(\rho)}$*

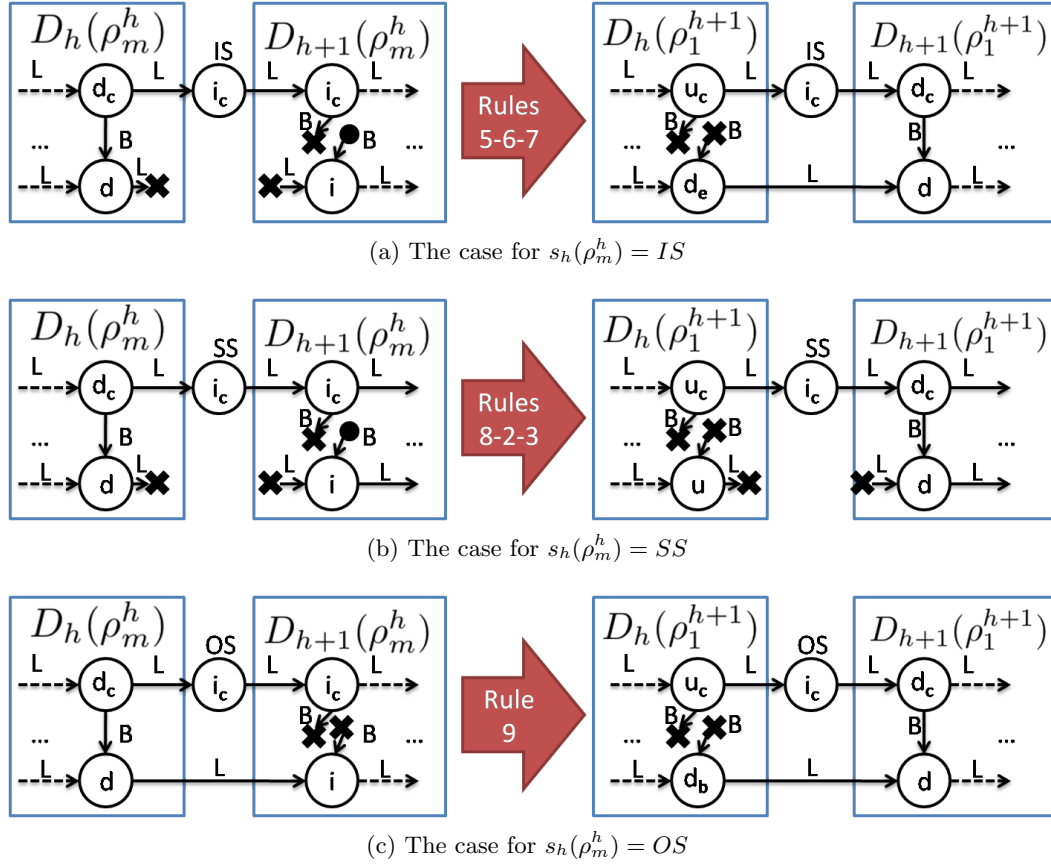


Figure 4.7: Illustration of Lemma 4.2.4

is always uniquely rewritten into its leftmost-bridged reaction-SOG ρ_1^n by the set of rules in Figs. 3.13, 3.14, 3.15, 3.16, and 3.17.

Proof. We prove Lemma 4.2.5 by induction.

When $n = \alpha(\rho)$, it is obvious that Lemma 4.2.5 is true.

Suppose that Lemma 4.2.5 is true for $n = k$, where $\alpha(\rho) \leq k \leq \omega(\rho) - 1$. Then for $n = k + 1$, $\rho_1^{\alpha(\rho)}$ is always uniquely rewritten into ρ_1^k . From Lemma 4.2.3, ρ_1^k is always uniquely rewritten into a rightmost-bridged reaction-SOG $\rho_{size(\rho,n)}^k \in ISOGF(\rho)$. From Lemma 4.2.4, $\rho_{size(\rho,n)}^k$ is always uniquely rewritten into a leftmost-bridged reaction-SOG ρ_1^{k+1} . Therefore, Lemma 4.2.5 is true for $n = k + 1$.

By the principle of induction, Lemma 4.2.5 is true for all $n = \alpha(\rho), \dots, \omega(\rho)$. $\square$

Now, we prove the correctness of the output-strand formation stage.

Theorem 4.2.6. *Suppose that each connection is stable once it is established, and that both Assumptions 4.2.1 and 4.2.2 hold. Then, for any reaction $\rho \in CReacts \cup DReacts$, its initial reaction-SOG $initial(\rho)$ is always uniquely rewritten into its unbridged reaction-SOG $unbridged(\rho)$.*

Proof. It is obvious that the only activated node in $initial(\rho)$ is the first NSO of $C(initial(\rho))$. It is also obvious that the only possible sequence of rules that can be applied to such a graph is the sequence of rules 1, 2, and 3 in this order. Thus, $initial(\rho)$ is always uniquely rewritten into the leftmost-bridged reaction-SOG $\rho_1^{\alpha(\rho)}$. Then, from Lemma 4.2.5, $\rho_1^{\alpha(\rho)}$ is always uniquely rewritten into the leftmost-bridged reaction-SOG ρ_1^k for $k = \omega(\rho)$. Then, from Lemma 4.2.3, ρ_1^k is always uniquely rewritten into the rightmost-bridged reaction-SOG ρ_m^k for $m = size(\rho, k)$. Since there is only one activated node in ρ_m^k , it is obvious that the only rule that can be applied to ρ_m^k is the rule 10. By applying such a rule, ρ_m^k is uniquely rewritten into the unbridged reaction-SOG $unbridged(\rho)$. Therefore, Theorem 4.2.6 holds. $\square$

4.3 Reinitialization Stage

For any reaction $\rho \in CReacts \cup DReacts$, the reinitialization stage rewrites the SOG $G = unbridged(\rho)$ into the SOG $G' = final(\rho)$. Because all the B ports in an unbridged reaction-SOG are hidden, the different terminated strands in G can be considered individually in this stage. In other words, this indicates that the reinitialization stage consists of the following three independent processes.

Chapter 4. Validity Proof of Rules for Reactions

- (1) The rewriting of the strand $C(G)$ into the strand $C(G')$.
- (2) The rewriting of the strand $M_0(G)$ into the strand $M_0(G')$.
- (3) The rewriting of the strand $M_1(G) \rightarrow \cdots \rightarrow M_{\omega(\rho)}(G)$.

The rewriting of the strand $M_1(G) \rightarrow \cdots \rightarrow M_{\omega(\rho)}(G)$ depends on the type of reaction.

- If $\rho \in CReacts$, then the strand $M_1(G) \rightarrow \cdots \rightarrow M_{\omega(\rho)}(G)$ is rewritten into the strand $M_1(G') \rightarrow \cdots \rightarrow M_{\omega(\rho)}(G')$.
- If $\rho \in DReacts$, then the strand $M_1(G) \rightarrow \cdots \rightarrow M_{\omega(\rho)}(G)$ is rewritten into the $\omega(\rho)$ strands $M_1(G')$, $\dots$, and $M_{\omega(\rho)}(G')$.

4.4 Proof of the Validity of the Reinitialization Stage

Before presenting the proof for each reinitialization, we will give the following definitions, and assumption.

4.4.1 Definitions

Definition 34 (Unbridged context-strands). *For any reaction $\rho \in CReacts \cup DReacts$, the unbridged context-strand of ρ is the instance of $C(\rho)$ satisfying that*

- (1) *each NSO of type T in this instance is in the state u_c ,*
- (2) *each NSO of type S is in the state i_c , and that*
- (3) *all the B ports in this instance are hidden.*

Definition 35 (Unbridged strands). *Let U be a template strand of size n .*

- *The simple-unbridged strand of U is an instance of U satisfying that each of its NSOs is in the state u and has its B port hidden.*
- *The established-unbridged strand of U is an instance of U satisfying that each of its NSOs is in the state u and has its B port hidden, except the last NSO that is in the state u_e .*
- *The breaking-unbridged strand of U is an instance of U satisfying that each of its NSOs is in the state u and has its B port hidden, except the last NSO that is in the state u_b .*

Definition 36 (Unbridged stimulus-strands). *For any reaction $\rho \in CReacts \cup DReacts$ such that $\alpha(\rho) = 0$, the unbridged stimulus-strand of ρ is the terminated instance-strand that is the simple-unbridged strand of $M_0(\rho)$.*

Definition 37 (Unbridged material-strands). *For any reaction $\rho \in CReacts$, the unbridged material-strand of ρ is the terminated-instance strand $M' : M'_1 \rightarrow \dots \rightarrow M_{\omega(\rho)'}'$ satisfying that*

- (1) *each strand M'_i is the established-unbridged strand of $M_i(\rho)$ for $i = 1, \dots, \omega(\rho) - 1$, and*
- (2) *the strand $M'_{\omega(\rho)'}$ is the simple-unbridged strand of $M_{\omega(\rho)}$.*

For any reaction $\rho \in DReacts$, the unbridged material-strand of ρ is the terminated-instance strand $M' : M'_1 \rightarrow \dots \rightarrow M_{\omega(\rho)'}'$ satisfying that

- (1) *each strand M'_i is the breaking-unbridged strand of $M_i(\rho)$ for any $i = 1, \dots, \omega(\rho) - 1$, and that*
- (2) *the strand $M'_{\omega(\rho)'}$ is the simple-unbridged strand of $M_{\omega(\rho)}$.*

Definition 38 (Reinitializing context-strands). *For any reaction $\rho \in CReacts \cup DReacts$, the reinitializing context-strand of degree n ($1 \leq n \leq p$) of ρ is the instance $C : c_1 \rightarrow \dots \rightarrow c_p$ of the context strand $C(\rho)$ of size p satisfying that*

- (1) *each NSO c_i for $i < n$ is in the state u_c or i_c ,*
- (2) *the NSO c_n is in the state r_c ,*
- (3) *each NSO c_j for $j > n$ is in the state i_c , and that*
- (4) *all the B ports in C are hidden.*

It is easy to observe that if $n > 1$ there is only one activated node in a reinitializing context strand, and this node is the NSO c_{n-1} . Otherwise, if $n = 1$, the only activated node is c_1 .

Definition 39 (Reinitializing material-strands). *Let $U = u_1 \rightarrow u_2 \rightarrow \dots \rightarrow u_p$ be a template strand for any $p \geq 1$. The reinitializing material-strand of degree n of U ($1 \leq n \leq p$) is the instance $V = v_1 \rightarrow v_2 \rightarrow \dots \rightarrow v_p$ of U satisfying that*

- (1) *each NSO v_j for $j < n$ is in the state u ,*
- (2) *the NSO v_n is in the state r ,*

Chapter 4. Validity Proof of Rules for Reactions

(3) each NSO v_k for $k > n$ is in the state i , and that

(4) all the B ports in V are hidden.

Definition 40 (Reinitializing output-strand). *For any reaction $\rho \in CReacts$, for any $h = 1, \dots, \omega(\rho)$, and for any $n = 1, \dots, \text{size}(M_h(\rho))$, $\mathcal{R}c_n^h(\rho)$ denotes the instance-strand $M'_1 \rightarrow \dots \rightarrow M'_{\omega(\rho)}$, called a reinitializing output-strand of ρ , satisfying that*

(1) each strand M'_i is the established-unbridged strand of $M_i(\rho)$ for $i = 1, \dots, h-1$,

(2) the strand M'_h is the reinitializing material-strand of degree n of $M_h(\rho)$, and that

(3) each strand M'_j is the visible-initialized strand of $M_j(\rho)$ for $j = h+1, \dots, \omega(\rho)$.

For any $k = \{1, \dots, \omega(\rho)\}$, $M'_k(\mathcal{R}c_n^h(\rho))$ refers to the strand M'_k .

It is easy to observe that if $n > 1$ there is only one activated node in a reinitializing output-strand, and this node is the $(n-1)$ -th NSO of M'_h . Otherwise, if $n = 1$ and $h > 1$, the only activated node is the last NSO of M'_{h-1} . And finally, if $n = 1$ and $h = 1$, the only activated node is the first NSO of M'_1 .

Definition 41 (Reinitializing output-SOG). *For any reaction $\rho \in DReacts$, for any $h = 1, \dots, \omega(\rho)$, and for any $n = 1, \dots, \text{size}(M_h(\rho))$, $\mathcal{R}d_n^h(\rho)$ denotes an instance-SOG, called the reinitializing output-SOG of ρ and consisting of $\omega(\rho)$ instance-strand $M'_1, \dots, M'_{\omega(\rho)}$, satisfying that*

(1) each strand M'_i is the breaking-unbridged strand of $M_i(\rho)$ for $i = 1, \dots, h-1$,

(2) the strand M'_h is the reinitializing material-strand of degree n of $M_h(\rho)$,

(3) each strand M'_j is the visible-initialized strand of $M_j(\rho)$ for $j = h+1, \dots, \omega(\rho)$, and that

(4) the strands M'_i and M'_h form a strand $M'_1 \rightarrow M'_2 \rightarrow \dots \rightarrow M'_h$, for $i = 1, \dots, h-1$.

For any $k = \{1, \dots, \omega(\rho)\}$, $M'_k(\mathcal{R}d_n^h(\rho))$ refers to the strand M'_k .

For the simplicity of our following discussion, let us temporally assume that the following assumption holds true.

4.4.2 Assumption “No rule 1”

Assumption 4.4.1 (No rule 1). *For any reaction $\rho \in CReacts \cup DReacts$, let us assume that the rule 1 (Fig. 3.13) can rewrite only either the initialized reaction-SOG of ρ , or the final reaction-SOG of ρ .*

We discuss in Section 4.7 what happens if this assumption does not hold.

4.4.3 Proof of the validity of the reinitialization of the context

Lemma 4.4.2. *Suppose that each connection is stable once it is established. Then, for any reaction $\rho \in CReacts \cup DReacts$, and for any $n = 1, \dots, p$, the reinitializing context-strand $C : c_1 \rightarrow \dots \rightarrow c_p$ of degree n of the context strand of ρ is always uniquely rewritten into the reinitializing context-strand C' of degree 1 of ρ by the set of rules in Figs. 3.13, 3.14, 3.15, 3.16, and 3.17.*

Proof. We prove Lemma 4.4.2 by induction.

When $n = 1$, it is obvious that Lemma 4.4.2 is true.

Suppose that Lemma 4.4.2 holds for $n = k$, where $1 \leq k \leq p - 1$. Then for $n = k + 1$, since the only activated node in C is the NSO c_k it is obvious that only the rule 12 can be applied to C . By applying the rule 12, C is uniquely rewritten into the reinitializing strand C'' of degree k of the context strand of ρ as shown in Fig. 4.8. Thus, C'' is always uniquely rewritten into C' . Therefore, Lemma 4.4.2 is true for $n = k + 1$.

By the principle of induction, Lemma 4.4.2 holds. □

Lemma 4.4.3. *Suppose that each connection is stable once it is established, and that Assumption 4.4.1 holds. Then, for any reaction $\rho \in CReacts \cup DReacts$, let $G_u = \text{unbridged}(\rho)$ and $G_f = \text{final}(\rho)$. The strand $C(G_u)$ is always uniquely rewritten into the strand $C(G_f)$.*

Proof. The strand $C(G_u)$ corresponds to the unbridged context-strand of ρ .

Let $m = \text{size}(C(G_u))$. It is obvious that only the rule 11 can be applied to $C(G_u)$. By applying the rule 11 to $C(G_u)$, $C(G_u)$ is rewritten into the reinitializing context-strand C' of degree m of ρ . Then, from Lemma 4.4.2, C' is rewritten into the reinitializing context-strand C'' of degree 1 of ρ . Then, it is obvious that only the rule 13 can be applied to C'' . By applying the rule 13 to C'' , C'' is rewritten into the $C(G_f)$, and therefore Lemma 4.4.3 holds. □

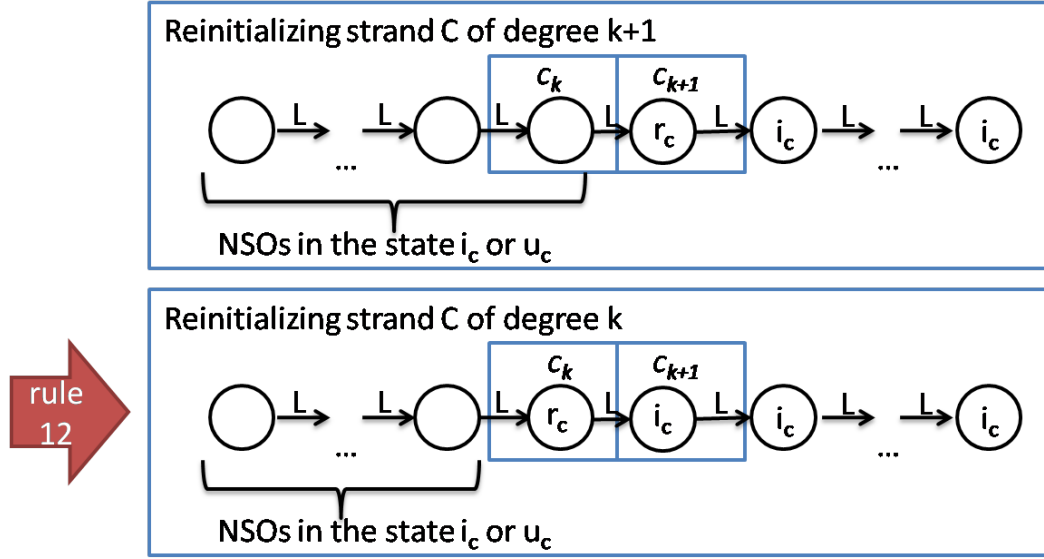


Figure 4.8: Illustration of Lemma 4.4.2. The hidden port are not represented.

4.4.4 Proof of the validity of the reinitialization of the material strand for a composition reaction

Lemma 4.4.4. *Suppose that each connection is stable once it is established. Then, for any reaction $\rho \in CReacts$, for any $h = 1, \dots, \omega(\rho)$, and for any $n = 1, \dots, \text{size}(M_h)$, the reinitializing output-strand $\mathcal{R}c_n^h(\rho)$ is always uniquely rewritten into the reinitializing output-strand $\mathcal{R}c_1^h(\rho)$ by the set of rules in Figs. 3.13, 3.14, 3.15, 3.16, and 3.17.*

Proof. We prove Lemma 4.4.4 by induction.

When $n = 1$, it is obvious that Lemma 4.4.4 is true.

Suppose that Lemma 4.4.4 holds for $n = k$, where $1 \leq k \leq \text{size}(M_h(\rho)) - 1$. Then for $n = k + 1$, since in $\mathcal{R}c_{k+1}^h(\rho)$ the only activated node is the $(k + 1)$ -th NSO of $M_h'(\mathcal{R}c_{k+1}^h(\rho))$, it is obvious that only the rule 15 can be applied to $\mathcal{R}c_{k+1}^h(\rho)$. By applying the rule 15, $\mathcal{R}c_{k+1}^h(\rho)$ is uniquely rewritten into $\mathcal{R}c_k^h(\rho)$. And since $\mathcal{R}c_k^h(\rho)$ is always uniquely rewritten into $\mathcal{R}c_1^h(\rho)$, $\mathcal{R}c_{k+1}^h(\rho)$ is always uniquely rewritten into $\mathcal{R}c_1^h(\rho)$. Therefore, Lemma 4.4.4 holds for $n = k + 1$.

By the principle of induction, Lemma 4.4.4 holds. □

Lemma 4.4.5. *Suppose that each connection is stable once it is established. Then, for any reaction $\rho \in CReacts$, for any $h = 1, \dots, \omega(\rho)$, the reinitializing output-*

strand $\mathcal{R}c_1^h(\rho)$ is always uniquely rewritten into the reinitializing output-strand $\mathcal{R}c_1^1(\rho)$ by the set of rules in Figs. 3.13, 3.14, 3.15, 3.16, and 3.17.

Proof. We prove Lemma 4.4.5 by induction.

When $h = 1$, it is obvious that Lemma 4.4.5 is true.

Suppose that Lemma 4.4.5 holds for $h = k$, where $1 \leq k \leq \omega(\rho) - 1$. Then for $h = k + 1$, since there is only one activated node in a reinitializing output-strand, it is obvious that only the rule 16 can be applied to $\mathcal{R}c_1^{k+1}(\rho)$. By applying the rule 16 to $\mathcal{R}c_1^{k+1}(\rho)$, $\mathcal{R}c_1^{k+1}(\rho)$ is rewritten into $\mathcal{R}c_{size(M_k(\rho))}^k(\rho)$. From Lemma 4.4.4, $\mathcal{R}c_{size(M_k(\rho))}^k(\rho)$ is always uniquely rewritten into $\mathcal{R}c_1^k(\rho)$. And since $\mathcal{R}c_1^k(\rho)$ is always uniquely rewritten into $\mathcal{R}c_1^1(\rho)$, that means $\mathcal{R}c_1^{k+1}(\rho)$ is always uniquely rewritten into $\mathcal{R}c_1^1(\rho)$. Therefore, Lemma 4.4.5 holds for $h = k + 1$.

By the principle of induction, Lemma 4.4.5 holds. □

Lemma 4.4.6. *Suppose that each connection is stable once it is established, and that Assumption 4.2.2 holds. Then, for any reaction $\rho \in CReacts$, let $G_u = unbridged(\rho)$ and $G_f = final(\rho)$. The strand $M_1(G_u) \rightarrow \cdots \rightarrow M_{\omega(\rho)}(G_u)$ is always uniquely rewritten into the output strand of ρ , i.e., $M_1(G_f) \rightarrow \cdots \rightarrow M_{\omega(\rho)}(G_f)$.*

Proof. The strand $M_1(G_u) \rightarrow \cdots \rightarrow M_{\omega(\rho)}(G_u)$ corresponds to the unbridged material-strand of ρ . It is obvious that only the rule 14 can be applied to $M_1(G_u) \rightarrow \cdots \rightarrow M_{\omega(\rho)}(G_u)$. By applying the rule 14 to $M_1(G_u) \rightarrow \cdots \rightarrow M_{\omega(\rho)}(G_u)$, $M_1(G_u) \rightarrow \cdots \rightarrow M_{\omega(\rho)}(G_u)$ is rewritten into the reinitializing output-strand $\mathcal{R}c_{size(M_{\omega(\rho)}(\rho))}^{\omega(\rho)}(\rho)$. From Lemma 4.4.4, $\mathcal{R}c_{size(M_{\omega(\rho)}(\rho))}^{\omega(\rho)}(\rho)$ is always uniquely rewritten into $\mathcal{R}c_1^{\omega(\rho)}(\rho)$. Then, from Lemma 4.4.5, $\mathcal{R}c_1^{\omega(\rho)}(\rho)$ is uniquely rewritten into $\mathcal{R}c_1^1(\rho)$. It is obvious that only the rule 18 can be applied to $\mathcal{R}c_1^1(\rho)$. By applying the rule 18, $\mathcal{R}c_1^1(\rho)$ is rewritten into $M_1(G_f) \rightarrow \cdots \rightarrow M_{\omega(\rho)}(G_f)$, and therefore Lemma 4.4.6 holds. □

4.4.5 Proof of the validity of the reinitialization of the material strands for a decomposition reaction

Lemma 4.4.7. *Suppose that each connection is stable once it is established, and that Assumption 4.2.2 holds. Then, for any $h = 1, \dots, \omega(\rho)$, and for any $n = 1, \dots, size(M_h)$, the reinitializing output-SOG $\mathcal{R}d_n^h(\rho)$ is always uniquely rewritten into the reinitializing output-SOG $\mathcal{R}d_1^h(\rho)$ by the set of rules in Figs. 3.13, 3.14, 3.15, 3.16, and 3.17.*

Proof. We prove Lemma 4.4.7 by induction.

When $n = 1$, it is obvious that Lemma 4.4.7 is true.

Suppose that Lemma 4.4.7 holds for $n = k$, where $1 \leq k \leq \text{size}(M_h(\rho)) - 1$. Then for $n = k + 1$, since in $\mathcal{R}d_{k+1}^h(\rho)$ the only activated node is the k -th NSO of $M_h'(\mathcal{R}d_{k+1}^h(\rho))$ it is obvious that only the rule 15 can be applied to $\mathcal{R}d_{k+1}^h(\rho)$. By applying the rule 15, $\mathcal{R}d_{k+1}^h(\rho)$ is uniquely rewritten into $\mathcal{R}d_k^h(\rho)$ as shown in Fig. 4.9. And since $\mathcal{R}d_k^h(\rho)$ is always uniquely rewritten into $\mathcal{R}d_1^h(\rho)$, $\mathcal{R}d_{k+1}^h(\rho)$ is always uniquely rewritten into $\mathcal{R}d_1^h(\rho)$. Therefore, Lemma 4.4.7 holds for $n = k + 1$.

By the principle of induction, Lemma 4.4.7 holds.

□

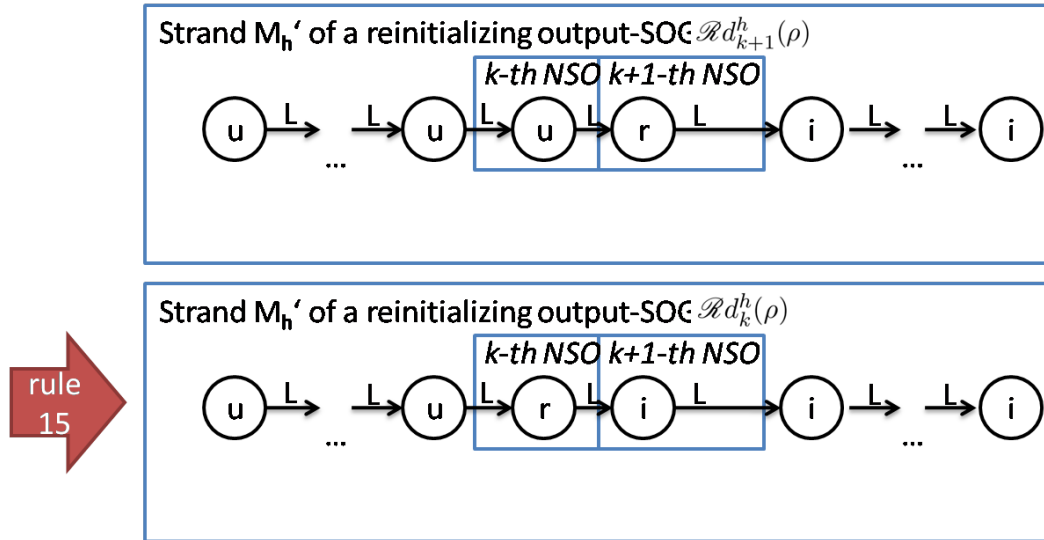


Figure 4.9: Illustration of Lemma 4.4.7. The hidden port are not represented.

Lemma 4.4.8. Suppose that each connection is stable once it is established, and that Assumption 4.2.2 holds. Then, for any $h = 1, \dots, \omega(\rho)$, the reinitializing output-SOG $\mathcal{R}d_1^h(\rho)$ is always uniquely rewritten into the reinitializing output-SOG $\mathcal{R}d_1^1(\rho)$ by the set of rules in Figs. 3.13, 3.14, 3.15, 3.16, and 3.17.

Proof. We prove Lemma 4.4.8 by induction.

When $h = 1$, it is obvious that Lemma 4.4.8 is true.

Suppose that Lemma 4.4.8 holds for $h = k$, where $1 \leq k \leq \omega(\rho) - 1$. Then for $h = k + 1$, since there is only one activated node in a reinitializing output-strand, it is obvious that only the rule 17 can be applied to $\mathcal{R}d_1^{k+1}(\rho)$. By applying the rule

17 to $\mathcal{R}d_1^{k+1}(\rho)$, $\mathcal{R}d_1^{k+1}(\rho)$ is rewritten into $\mathcal{R}d_{size(M_k(\rho))}^k(\rho)$. From Lemma 4.4.7, $\mathcal{R}d_{size(M_k(\rho))}^k(\rho)$ is always uniquely rewritten into $\mathcal{R}d_1^k(\rho)$. And since $\mathcal{R}d_1^k(\rho)$ is always uniquely rewritten into $\mathcal{R}d_1^1(\rho)$, that means $\mathcal{R}d_1^{k+1}(\rho)$ is always uniquely rewritten into $\mathcal{R}d_1^1(\rho)$. Therefore, Lemma 4.4.8 holds for $h = k + 1$.

By the principle of induction, Lemma 4.4.8 holds. □

Lemma 4.4.9. *Suppose that each connection is stable once it is established, and that Assumption 4.2.2 holds. Let $G_u = \text{unbridged}(\rho)$ and $G_f = \text{final}(\rho)$. Then, the strand $M_1(G_u) \rightarrow \dots \rightarrow M_{\omega(\rho)}(G_u)$ is always uniquely rewritten into $\omega(\rho)$ strands $M_1(G_f), \dots, M_{\omega(\rho)}(G_f)$. This $\omega(\rho)$ strands correspond to the output strands of ρ .*

Proof. The strand $M_1(G_u) \rightarrow \dots \rightarrow M_{\omega(\rho)}(G_u)$ corresponds to the unbridged material-strand of ρ . It is obvious that only the rule 14 can be applied to $M_1(G_u) \rightarrow \dots \rightarrow M_{\omega(\rho)}(G_u)$. By applying the rule 14 to $M_1(G_u) \rightarrow \dots \rightarrow M_{\omega(\rho)}(G_u)$, $M_1(G_u) \rightarrow \dots \rightarrow M_{\omega(\rho)}(G_u)$ is rewritten into the reinitializing output-SOG $\mathcal{R}d_{size(M_{\omega(\rho)}(\rho))}^{\omega(\rho)}(\rho)$. From Lemma 4.4.7, $\mathcal{R}d_{size(M_{\omega(\rho)}(\rho))}^{\omega(\rho)}(\rho)$ is always uniquely rewritten into $\mathcal{R}d_1^{\omega(\rho)}(\rho)$. Then, from Lemma 4.4.8, $\mathcal{R}d_1^{\omega(\rho)}(\rho)$ is uniquely rewritten into $\mathcal{R}d_1^1(\rho)$. It is obvious that only the rule 18 can be applied to $\mathcal{R}d_1^1(\rho)$. By applying the rule 18, $\mathcal{R}d_1^1(\rho)$ is rewritten into $\omega(\rho)$ strand $M_1(G_f), \dots, M_{\omega(\rho)}(G_f)$, and therefore Lemma 4.4.9 holds. □

4.4.6 Proof of the validity of the reinitialization of the stimulus strand

Lemma 4.4.10. *Suppose that each connection is stable once it is established, and that Assumption 4.2.2 holds. For any reaction $\rho \in CReacts \cup DReacts$ such that $\alpha(\rho) = 0$, let $G_u = \text{unbridged}(\rho)$ and $G_f = \text{final}(\rho)$. The strand $M_0(G_u)$ is always uniquely rewritten into the strand $M_0(G_f)$.*

Proof. The strand $M_0(G_u)$ corresponds to the unbridged stimulus-strand of ρ . It is obvious that only the rule 14 can be applied to $M_0(G_u)$. By applying the rule 14 to $M_0(G_u)$, $M_0(G_u)$ is rewritten into the equivalent of a reinitializing output-SOG $\mathcal{R}c_{size(M_1(\rho'))}^1(\rho')$ of a reaction ρ' such that $\alpha(\rho') = \omega(\rho') = 1$ and $M_0(\rho) = M_1(\rho')$. From Lemma 4.4.7, $\mathcal{R}c_{size(M_1(\rho'))}^1(\rho')$ is always uniquely rewritten into $\mathcal{R}c_1^1(\rho')$. It is obvious that only the rule 18 can be applied to $\mathcal{R}c_1^1(\rho')$. By applying the rule 18, $\mathcal{R}c_1^1(\rho')$ is rewritten into $M_0(G_u)$, and therefore Lemma 4.4.10 holds. □

4.4.7 Proof of the validity of the reinitialization

Theorem 4.4.11. *Suppose that each connection is stable once it is established, and that both Assumptions 4.4.1 and 4.2.2 holds. Then, the unbridged reaction-SOG $G_u = \text{unbridged}(\rho)$ is always uniquely rewritten into the final reaction-SOG $G_f = \text{final}(\rho)$.*

Proof. Let $\rho \in CReacts$. The unbridged reaction-SOG $G_u = \text{unbridged}(\rho)$ is only composed of the strands $C(G_u)$, $M_1(G_u) \rightarrow \dots \rightarrow M_{\omega(\rho)}(G_u)$, and, possibly, $M_0(G_u)$. The final reaction-SOG $G_f (= \text{final}(\rho))$ is only composed of the strands $C(G_f)$, $M_1(G_f) \rightarrow \dots \rightarrow M_{\omega(\rho)}(G_f)$, and, possibly, $M_0(G_f)$. Since, from Lemma 4.4.3, $C(G_u)$ is always uniquely rewritten into $C(G_f)$, from Lemma 4.4.10, if $\alpha(\rho) = 0$, $M_0(G_u)$ is always uniquely rewritten into $M_0(G_f)$, and from Lemma 4.4.6, if $\rho \in CReacts$, $M_1(G_u) \rightarrow \dots \rightarrow M_{\omega(\rho)}(G_u)$ is always uniquely rewritten into $M_1(G_f) \rightarrow \dots \rightarrow M_{\omega(\rho)}(G_f)$, it is true that for any reaction $\rho \in CReacts$, the unbridged reaction-SOG $G_u (= \text{unbridged}(\rho))$ is always uniquely rewritten into the final reaction-SOG $G_f (= \text{final}(\rho))$.

Now, let $\rho \in DReacts$. The unbridged reaction-SOG $G_u (= \text{unbridged}(\rho))$ is only composed of the strands $C(G_u)$, $M_1(G_u) \rightarrow \dots \rightarrow M_{\omega(\rho)}(G_u)$, and, possibly, $M_0(G_u)$. The final reaction-SOG $G_f (= \text{final}(\rho))$ is only composed of the strand $C(G_f)$, the $\omega(\rho)$ strands $M_1(G_f), \dots, M_{\omega(\rho)}(G_f)$, and, possibly, $M_0(G_f)$. Since, from Lemma 4.4.3, $C(G_u)$ is always uniquely rewritten into $C(G_f)$, from Lemma 4.4.10, if $\alpha(\rho) = 0$, $M_0(G_u)$ is always uniquely rewritten into $M_0(G_f)$, and from Lemma 4.4.9, if $\rho \in DReacts$, $M_1(G_u) \rightarrow \dots \rightarrow M_{\omega(\rho)}(G_u)$ is always uniquely rewritten into $\omega(\rho)$ strands $M_1(G_f), \dots, M_{\omega(\rho)}(G_f)$, it is true that for any reaction $\rho \in DReacts$, the unbridged reaction-SOG $G_u (= \text{unbridged}(\rho))$ is always uniquely rewritten into the final reaction-SOG $G_f (= \text{final}(\rho))$.

Therefore, Theorem 4.4.11 holds. □

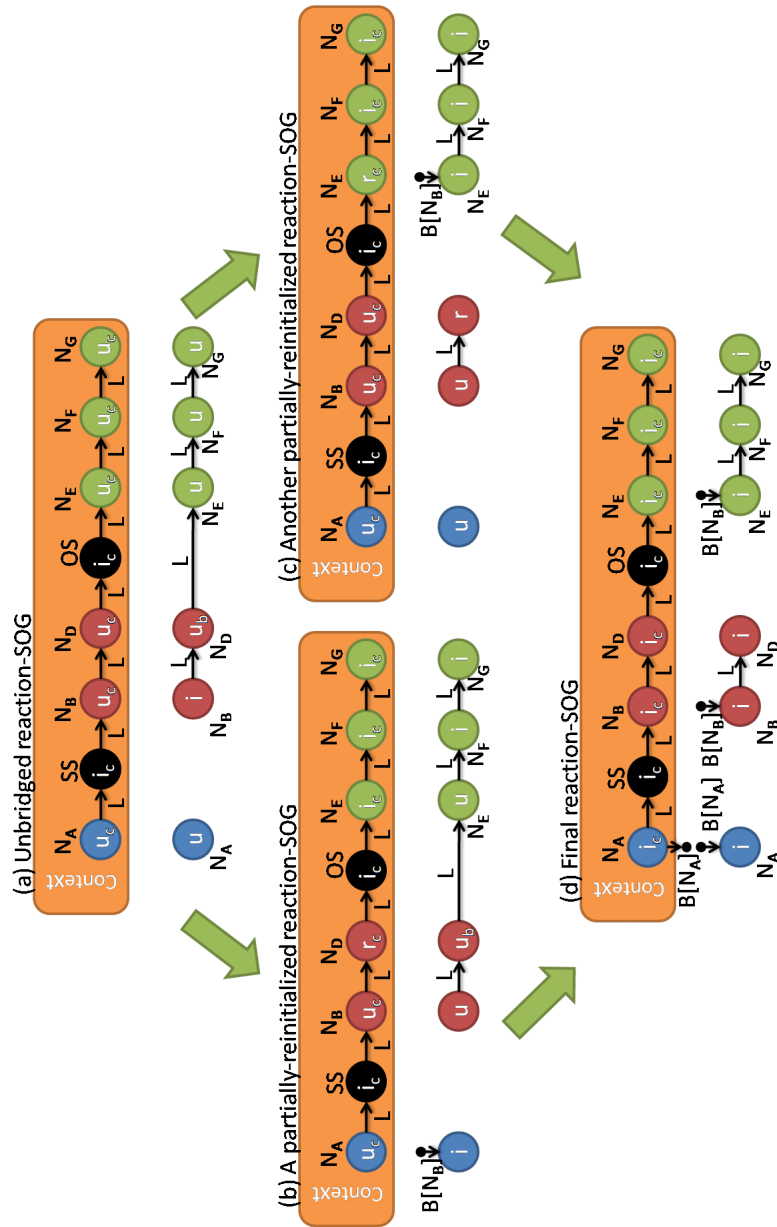
4.4.8 Confluence during the reinitialization stage

In this section, we show the confluence that holds in the reinitialization stage.

Definition 42 (Partially-reinitialized reaction-SOG). *For any reaction $\rho \in CReacts \cup DReacts$, a partially-reinitialized reaction-SOG of ρ is a reaction-SOG $G \in ISOGF(\rho)$ satisfying that one strand in G at least is a reinitializing strand.*

For any reaction $\rho \in CReacts \cup DReacts$, all the intermediate reaction-SOGs that we obtain during the reinitialization stage, i.e., during the rewriting

of the reaction-SOG *unbridged*(ρ) into the reaction-SOG *final*(ρ), are partially-reinitialized reaction-SOG. Since the reinitialization stage consists of three independent processes (i.e., the reinitialization of the context, the reinitialization of the stimulus, and the reinitialization of the material strands), it is possible to get different partially-reinitialized reaction-SOGs depending on the speed of these three processes. For example, the unbridged reaction-SOG shown in Fig. 4.10a may be rewritten into the partially-reinitialized reaction-SOG shown in Fig. 4.10b, or into the partially-reinitialized reaction-SOG shown in Fig. 4.10c, depending on the speed of the reinitialization of the context, of the stimulus strand, and of the material strands. These are only two examples out of many other possibilities. Any partially-reinitialized reaction-SOG of ρ consists of intermediate results of the reinitialization of the context of ρ , of the stimulus strand of ρ , and of the material strands of ρ . Since these processes are independent, it is obvious that any partially-reinitialized reaction-SOG of ρ is always rewritten into the final reaction-SOG of ρ . For example, for both partially-reinitialized reaction-SOGs shown in Figs. 4.10b and c, the reinitialization of the context, of the stimulus strand and of the material strands will continue to make them rewritten into the final reaction-SOG shown in Fig. 4.10d. In other words, any reinitialization process may merge to a confluence.



Assuming that a strand can participate in other reactions only after all the strands of its current reaction are reinitialized, the unbridged reaction-SOG (a) of a reaction is always rewritten into the final reaction-SOG of the reaction (d). However, since the reinitialization stage consists of three independent processes, it is possible to get intermediate different results that we call partially-reinitialized reaction-SOGs. For example in this figure, we show two out of many possible partially-reinitialized reaction-SOGs for the reaction ρ' . However, since any partially-reinitialized reaction-SOG will always be rewritten into the final reaction-SOG, there is confluence.

Figure 4.10: An illustration of the confluence found in the reinitialization stage of a reaction ρ' (Fig. 3.5).

4.5 Correctness of the rewriting rules for implementing a Reaction

Theorem 4.5.1. *Suppose that each connection is stable once it is established, and that the three Assumptions 4.2.1, 4.4.1, and 4.2.2 holds. Then, for any reaction $\rho \in CReact \cup DReact$, the initial reaction-SOG $initial(\rho)$ is always uniquely rewritten into the final reaction-SOG $final(\rho)$.*

Proof. From Theorem 4.2.6 and Theorem 4.4.11, it is obvious that Theorem 4.5.1 holds. □

4.6 Correctness of Error Management Rules

There are two different cases in which the reaction process may fail. The first case may occur due to the loss of a B connection because of the instability of proximity-based connections, and the second case may occur due to the incompatibility between a pair of strands. We assume that the L connections are all stable. We deal with this error by rewriting the current reaction-SOG of the reaction into the initial reaction-SOG of the reaction. We call this rewriting the resetting process.

As well as for the reinitialization stage, during the resetting process of a reaction some strands are reinitialized before the resetting process is fully finished. This means that some strands may participate in another reaction before the resetting process of the reaction is finished, and thus will not be initialized anymore if this happens. However, once the strands are reinitialized, they are not needed for completing the resetting of the remaining strands. Thus, when looking at only the resetting process, the two following scenarios can be considered equivalent.

- (1) A strand can participate in another reaction as soon as it is reinitialized.
- (2) A strand can participate in another reaction only when all the strands of the current reaction are reinitialized.

In practice, the first scenario will happen. However, we assume for simplicity that strands can participate in another reaction only when all the strands of the current reaction are reinitialized.

4.6.1 Broken Connection Management

From the set of rules given in Fig. 3.13 and Fig. 3.14, it is obvious that a B connection can be spanned between two NSOs only when they are in specific states. All the possible cases are shown in Fig. 3.16a. It is also obvious that at any moment during a reaction process, there exist only one or two B connections.

Definition 43 (Broken context-strands). *For any reaction $\rho \in CReacts \cup DReacts$, the broken context-strand of degree n ($1 \leq n \leq p$) of ρ is the instance $C : c_1 \rightarrow \dots \rightarrow c_p$ of the context strand $C(\rho)$ satisfying the following conditions.*

- (1) $1 \leq n \leq p$
- (2) Each NSO $c_1, \dots, c_{n-1}$ is in the state u_c or i_c .
- (3) The NSO c_n is in the state d_c .
- (4) Each NSO $c_{n+1}, \dots, c_p$ is in the state i_c .
- (5) All the NSOs of the strand C have their B port hidden.

Definition 44 (Broken material-strands). *Let $U = u_1 \rightarrow u_2 \rightarrow \dots \rightarrow u_p$ be a template strand for any $p \geq 1$. The broken material-strand of degree n of U ($n = 1, \dots, p$) is the instance of $V = v_1 \rightarrow v_2 \rightarrow \dots \rightarrow v_p$ satisfying the following conditions.*

- (1) $1 \leq n \leq p$
- (2) Each NSO $v_1, \dots, v_{n-1}$ is in the state u .
- (3) The NSO v_n is in the state e .
- (4) Each NSO $v_{n+1}, \dots, v_p$ is in the state i .
- (5) All the NSOs of the strand V have their B port hidden.

Definition 45 (Broken output-SOG). *For any reaction $\rho \in CReacts$, for any $h = 1, \dots, \omega(\rho)$, and for any $n = 1, \dots, \text{size}(M_h)$, $\mathcal{B}c_n^h(\rho)$ denotes the instance-SOG called a broken output-strand of ρ consisting of $\omega(\rho)$ instance-strands $M'_1, \dots, M'_h$ satisfying the following conditions.*

- (1) Each strand M'_i is the established-unbridged strand of $M_i(\rho)$ for $i = 1, \dots, h-1$.
- (2) The strand M'_h is the broken material-strand of degree n of $M_h(\rho)$.

- (3) The strands $M'_1, \dots, M'_h$ form a terminated strand $M'_1 \rightarrow \dots \rightarrow M'_h$.
- (4) Each strand M'_j is the initialized visible-strand of $M_j(\rho)$ for $j = h+1, \dots, \omega(\rho)$.

Definition 46 (Broken output-strand). For any reaction $\rho \in D\text{Reacts}$, for any $h = 1, \dots, \omega(\rho)$, and for any $n = 1, \dots, \text{size}(M_h)$, $\mathcal{B}d_n^h(\rho)$ denotes the instance-strand $M'_1 \rightarrow \dots \rightarrow M'_h$ called a broken output-strand of ρ satisfying the following conditions.

- (1) Each strand M'_i is the breaking-unbridged strand of $M_i(\rho)$ for $i = 1, \dots, h-1$.
- (2) The strand M'_h is the broken material-strand of degree n of $M_h(\rho)$.
- (3) Each strand M'_j is the initialized hidden-strand of $M_j(\rho)$ for $j = h+1, \dots, \omega(\rho)$.
- (4) The strand $M_1 \rightarrow \dots \rightarrow M_{\omega(\rho)}$ is a terminated strand.

4.6.2 Loss of a B connection for the context

Lemma 4.6.1. Suppose that each L connection is stable once it is established. For any reaction $\rho \in C\text{Reacts} \cup D\text{React}$, for any $\text{SOG } G \in \text{ISOGF}(\rho)$, for any $1 \leq n \leq \text{size}(C(G))$, if $C(G) : c_1 \rightarrow \dots \rightarrow c_p$ is a broken context-strand of degree n , then $C(G)$ is always uniquely rewritten into $C(\text{initial}(\rho))$.

Proof. If $\text{type}(c_{n+1}) = IS$, only the rule e2 can be applied. Otherwise, if $n = p$, only the rule e4 can be applied. Otherwise, only the rule e3 can be applied. In these three cases, by applying the corresponding rule e2, $C(G)$ is rewritten into the reinitializing context-strands C' of degree n of ρ . From Lemma 4.4.2, C' is rewritten into the reinitializing context-strands C'' of degree 1 of ρ . Then, it is obvious that only the rule 13 can be applied to C'' . By applying the rule 13 to C'' , C'' is rewritten into the $C(\text{initial}(\rho))$, and therefore Lemma 4.6.1 holds. $\square$

Lemma 4.6.2. Suppose that each L connection is stable once it is established. For any reaction $\rho \in C\text{Reacts} \cup D\text{React}$, for any $\text{SOG } G \in \text{ISOGF}(\rho)$, for any $1 \leq n \leq \text{size}(C(G))$, if a B connection is lost during the output-strand formation, then the context is either reinitialized to continue the reaction, or is reset to start a new reaction.

Proof. During the inter-double-strand propagation (i.e., application of the sequence of rules 5-6-7), a B connection may be lost from an NSO in the state n_c , which

means we lost this B connection just after the application of the rule 6. It is obvious that only the rule e1 can be applied. By applying the rule e1, the SOG is reset to the situation that we had after the application of the rule 5, and has another chance to apply the rule 6. If during the inter-double-strand propagation we lost both the B connection from the NSO in the state n_c and the B connection from the NSO in the state d_c , the rule e2 will be applied and we will obtain a broken context-strand. Thus, from Lemma 4.6.1 the context will be reset to start a new reaction. If we lose a B connection from an NSO in the state i_c , which is possible only after having applied the sequence of rule 1-2, the loss of this B connection automatically reinitialize the context. All the other cases of the loss of a B connection will lead to a broken context-strand. Thus, from Lemma 4.6.1, the context will be reset to start a new reaction. Therefore, Lemma 4.6.2 holds. $\square$

4.6.3 Loss of a B connection for the material strands in a composition reaction

Lemma 4.6.3. *Suppose that each L connection is stable once it is established. For any reaction $\rho \in CReacts$, for any $h = 1, \dots, \omega(\rho)$, and for any $n = 1, \dots, size(M_h)$, the broken output-SOG $\mathcal{B}c_n^h(\rho)$ is always uniquely rewritten into the broken output-SOG $\mathcal{B}c_1^h(\rho)$ by the set of rules in Figs. 3.13, 3.14, 3.15, 3.16, and 3.17.*

Proof. We prove Lemma 4.6.3 by induction.

When $n = 1$, it is obvious that Lemma 4.6.3 is true.

Suppose that Lemma 4.6.3 holds for $n = k$, where $1 \leq k \leq size(M_h(\rho)) - 1$. Then for $n = k + 1$, since in $\mathcal{B}c_{k+1}^h(\rho)$ the only activated node is the $(k + 1)$ -th NSO of $M_h(\mathcal{B}c_{k+1}^h(\rho))$, it is obvious that only the rule e7 can be applied to $\mathcal{B}c_{k+1}^h(\rho)$. By applying the rule e7, $\mathcal{B}c_{k+1}^h(\rho)$ is uniquely rewritten into $\mathcal{B}c_k^h(\rho)$. And since $\mathcal{B}c_k^h(\rho)$ is always uniquely rewritten into $\mathcal{B}c_1^h(\rho)$, $\mathcal{B}c_{k+1}^h(\rho)$ is always uniquely rewritten into $\mathcal{B}c_1^h(\rho)$. Therefore, Lemma 4.6.3 holds for $n = k + 1$.

By the principle of induction, Lemma 4.6.3 holds. $\square$

Lemma 4.6.4. *Suppose that each L connection is stable once it is established. For any reaction $\rho \in CReacts$, for any $h = 1, \dots, \omega(\rho)$, the broken output-SOG $\mathcal{B}c_1^h(\rho)$ is always uniquely rewritten into the broken output-SOG $\mathcal{B}c_1^1(\rho)$ by the set of rules in Figs. 3.13, 3.14, 3.15, 3.16, and 3.17.*

Proof. We prove Lemma 4.6.4 by induction.

When $h = 1$, it is obvious that Lemma 4.6.4 is true.

Suppose that Lemma 4.6.4 holds for $h = k$, where $1 \leq k \leq \omega(\rho) - 1$. Then for $h = k + 1$, since there is only one activated node in a reinitializing output-strand, it is obvious that only the rule e9 can be applied to $\mathcal{B}c_1^{k+1}(\rho)$. By applying the rule e9 to $\mathcal{B}c_1^{k+1}(\rho)$, $\mathcal{B}c_1^{k+1}(\rho)$ is rewritten into $\mathcal{B}c_{size(M_k(\rho))}^k(\rho)$. From Lemma 4.6.3, $\mathcal{B}c_{size(M_k(\rho))}^k(\rho)$ is always uniquely rewritten into $\mathcal{B}c_1^k(\rho)$. And since $\mathcal{B}c_1^k(\rho)$ is always uniquely rewritten into $\mathcal{B}c_1^1(\rho)$, $\mathcal{B}c_1^{k+1}(\rho)$ is always uniquely rewritten into $\mathcal{B}c_1^1(\rho)$. Therefore, Lemma 4.6.4 holds for $h = k + 1$.

By the principle of induction, Lemma 4.6.4 holds. □

Lemma 4.6.5. *Suppose that each connection L is stable once it is established, and that no strand participates in another reaction while the resetting process is not terminated. For any reaction $\rho \in CReacts$, let $G_i = initial(\rho)$. If a B connection is lost during the output-strand formation and if we obtain an SOG $G \in ISOGF(\rho)$, then the $\omega(\rho)$ strands $M_1(G_f), \dots, M_{\omega(\rho)}(G_f)$ are always uniquely rewritten to obtain the strands $M_1(G_i), \dots, M_{\omega(\rho)}(G_i)$.*

Proof. If we lost a B connection during the beginning stage, i.e., the application of the sequence of rule 1-2-3 (or 8-2-3), we can obtain the following cases.

- If we lost the B connection just after having applied the rule 2, then no material NSO has the chance to change its state. Thus, it is only required to apply the rule e5 (that is the only possible rule to apply) to reinitialize the port that lost the connection and to have another chance to apply the rule 2.
- If we lost the B connection just after having apply the rule 3, then it is obvious that only the rule e6 can be applied. If the rule e6 is applied, we obtain a broken output-SOG.

If we lost a B connection during the inter-double-strand propagation stage, i.e., the application of the sequence of rule 5-6-7, we can obtain the following cases.

- If we lost the B connection just after having applied the rule 5, then it is obvious that only the rule e6 can be applied. If the rule e6 is applied, we obtain a broken output-SOG. The rules managing a loss of a B connection for the context will change the state of the port from visible to hidden (see rule e2). Thus, we will not have the material strand that becomes connected to this port.

- Similarly, if we lost both connections just after having applied the rule 6, it is obvious that only the rule e6 can be applied. If the rule e6 is applied, we obtain a broken output-SOG. The rules managing a loss of a B connection for the context will change the state of the port from visible to hidden (see rule e2). Thus, we will not have the material strand that becomes connected to this port.
- If we lost the B connection from the NSO in the state n_c just after having applied the rule 6, then the NSO to which we lost the connection did not have the chance to change its state. Thus, it is only possible and required to apply the rule e5 to reinitialize the port that lost the connection and to have another chance to apply the rule6.
- If we lost the B connection just after having applied the rule 7, then it is obvious that only the rule e6 can be applied. If the rule e6 is applied, we obtain a broken output-SOG. The rules managing a loss of a B connection will change the state of the port from visible to hidden (see rule e2). Thus, we will not have the material strand that becomes connected to this port.

All the other cases lead to a broken output-SOG by applying the rule e6. Thus, from Lemma 4.6.4, we will obtain a broken output-SOG of degree 1. It is obvious that the only possible rule to apply to a broken output-SOG of degree 1 is the rule e10. By doing so, the broken output-SOG of degree 1 is rewritten into the strands $M_1(G_i), \dots, M_{\omega(\rho)}(G_i)$. Therefore, Lemma 4.6.5 holds. $\square$

4.6.4 Loss of a B connection for the material strands in a decomposition reaction

Lemma 4.6.6. *Suppose that each connection L is stable once it is established, and that no strand participates in another reaction while the resetting process is not terminated. For any reaction $\rho \in D\text{Reacts}$, for any $h = 1, \dots, \omega(\rho)$, and for any $n = 1, \dots, \text{size}(M_h)$, the broken output-strand $\mathcal{B}d_n^h(\rho)$ is always uniquely rewritten into the broken output-strand $\mathcal{B}d_1^h(\rho)$ by the set of rules in Figs. 3.13, 3.14, 3.15, 3.16, and 3.17.*

Proof. We prove Lemma 4.6.6 by induction.

When $n = 1$, it is obvious that Lemma 4.6.6 is true.

Suppose that Lemma 4.6.6 holds for $n = k$, where $1 \leq k \leq \text{size}(M_h(\rho)) - 1$. Then for $n = k + 1$, since in $\mathcal{B}d_{k+1}^h(\rho)$ the only activated node is the $(k + 1)$ -th NSO

of $M_h(\mathcal{B}d_{k+1}^h(\rho))$, it is obvious that only the rule e7 can be applied to $\mathcal{B}d_{k+1}^h(\rho)$. By applying the rule e7, $\mathcal{B}d_{k+1}^h(\rho)$ is uniquely rewritten into $\mathcal{B}d_k^h(\rho)$. And since $\mathcal{B}d_k^h(\rho)$ is always uniquely rewritten into $\mathcal{B}d_1^h(\rho)$, $\mathcal{B}d_{k+1}^h(\rho)$ is always uniquely rewritten into $\mathcal{B}d_1^h(\rho)$. Therefore, Lemma 4.6.6 holds for $n = k + 1$.

By the principle of induction, Lemma 4.6.6 holds. □

Lemma 4.6.7. *Suppose that each connection L is stable once it is established, and that no strand participates in another reaction while the resetting process is not terminated. For any reaction $\rho \in DReacts$, for any $h = 1, \dots, \omega(\rho)$, the broken output-strand $\mathcal{B}d_1^h(\rho)$ is always uniquely rewritten into the broken output-strand $\mathcal{B}d_1^1(\rho)$ by the set of rules in Figs. 3.13, 3.14, 3.15, 3.16, and 3.17.*

Proof. We prove Lemma 4.6.7 by induction.

When $h = 1$, it is obvious that Lemma 4.6.7 is true.

Suppose that Lemma 4.6.7 holds for $h = k$, where $1 \leq k \leq \omega(\rho) - 1$. Then for $h = k + 1$, since there is only one activated node in a reinitializing output-strand, it is obvious that only the rule e8 can be applied to $\mathcal{B}d_1^{k+1}(\rho)$. By applying the rule e8 to $\mathcal{B}d_1^{k+1}(\rho)$, $\mathcal{B}d_1^{k+1}(\rho)$ is rewritten into $\mathcal{B}d_{size(M_k(\rho))}^k(\rho)$. From Lemma 4.6.6, $\mathcal{B}d_{size(M_k(\rho))}^k(\rho)$ is always uniquely rewritten into $\mathcal{B}d_1^k(\rho)$. And since $\mathcal{B}d_1^k(\rho)$ is always uniquely rewritten into $\mathcal{B}d_1^1(\rho)$, that means $\mathcal{B}d_1^{k+1}(\rho)$ is always uniquely rewritten into $\mathcal{B}d_1^1(\rho)$. Therefore, Lemma 4.6.7 holds for $h = k + 1$.

By the principle of induction, Lemma 4.6.7 holds. □

Lemma 4.6.8. *Suppose that each connection L is stable once it is established, and that no strand participates to another reaction while the resetting process is not terminated. For any reaction $\rho \in DReacts$, let $G_i = initial(\rho)$, if a B connection is lost during the output-strand formation, and if we obtain an SOG $G \in ISOGF(\rho)$, then the $\omega(\rho)$ strands $M_1(G_f), \dots, M_{\omega(\rho)}(G_f)$ are always uniquely rewritten to obtain the strand $M_1(G_i) \rightarrow \dots \rightarrow M_{\omega(\rho)}(G_i)$.*

Proof. If we lost a B connection during the beginning stage, i.e., the application of the sequence of rule 1-2-3 (or 8-2-3), we can obtain the following cases.

- If we lost the B connection just after having applied the rule 2, then no material NSO has the chance to change its state. Thus, it is only required to apply the rule e5 (that is the only possible rule to apply) to reinitialize the port that lost the connection and to have another chance to applied the rule2.

Chapter 4. Validity Proof of Rules for Reactions

- If we lost the B connection just after having applied the rule 3, then it is obvious that only the rule e6 can be applied. If the rule e6 is applied, we obtain a broken output-strand-SOG.

All the other cases lead to a broken output-strand-SOG by applying the rule e6. Thus, from Lemma 4.6.7, we will obtain a broken output-strand-SOG of degree 1. It is obvious that the only possible rule to apply to a broken output-SOG of degree 1 is the rule e10. By doing so, the broken output-SOG of degree 1 is rewritten into the strands $M_1(G_i), \dots, M_{\omega(\rho)}(G_i)$.

Therefore, Lemma 4.6.8 holds. $\square$

4.6.5 Improved Correctness proof of the rewriting rules for implementing a Reaction

Theorem 4.6.9. *Suppose that each connection L is stable once it is established, and that no strand participates in another reaction while the resetting process is not terminated. For any reaction $\rho \in CReacts \cup DReact$, let $G_i = initial(\rho)$ and $G_f = final(\rho)$. If a B connection is lost during the output-strand formation such that we obtain an SOG $G \in ISOGF(\rho)$, G is either always uniquely rewritten into G_i , or always uniquely rewritten into G_f , depending on which connection is lost.*

Proof. From Theorem 4.2.6, Theorem 4.4.11, Lemma 4.6.2, Lemma 4.6.5, and Lemma 4.6.8, it is obvious that Theorem 4.6.9 holds. $\square$

Moreover, from the discussions in Section 3.4.4, we will obtain the following theorem.

Theorem 4.6.10. *Suppose that each connection L is stable once it is established, and that no strand participates in another reaction while the resetting process is not terminated. During the output-strand formation of a reaction, the sets of rules shown in Fig. 3.16 and 3.17 deal with the incompatible material-strands by resetting the context and all the material strands of this reaction when an NSO of the context becomes connected to the last NSO of a incompatible material-strands.*

4.7 Discussion about the assumptions

In this section, we discuss what happens if the different assumptions defined during the correctness proof of the of the rewriting rules for implementing a reaction do not hold.

4.7.1 Assumption “No rule 14”

In the proof of the validity of the output-strand formation stage, we supposed that Assumption 4.2.1 holds with the aim of simplifying our theory. The only purpose of this assumption, is to stop the stimulus strand from starting its reinitialization before the output-strand formation stage is completed. For any reaction $\rho \in CReacts \cup DReacts$ satisfying that $\alpha(\rho) = 0$, this could happen once the bridged reaction-SOG $\rho_{size(M_0(\rho))}^0$ is rewritten into the reaction-SOG G by the application of the rule 8, as shown in Fig. 4.11.

However, once we obtained the reaction-SOG G , the stimulus strand $M_0(G)$, and the other strands in $G - M_0(G)$ can be considered individually (Figs. 4.11c and e). In other words, we can separately consider two independent processes. The first one reinitializes the stimulus strand (Figs. 4.11c and d). The second one continues the output-strand formation with the strands in $G - M_0(G)$ and reinitializes these strands (Figs. 4.11e, f and g).

The first process consists of the rewriting of the strand $M_0(G)(= M_0(unbridged(\rho)))$ into the strand $M_0(final(\rho))$. However, from Lemma 4.4.10, if each connection is stable once it is established, and if Assumption 4.2.2 holds, $M_0(unbridged(\rho))$ is always uniquely rewritten into $M_0(final(\rho))$.

The second process consists of the rewriting of the SOG $G - M_0(G)$ into the SOG $unbridged(\rho) - M_0(unbridged(\rho))$, and the rewriting of the reinitialization of the strands in $unbridged(\rho) - M_0(unbridged(\rho))$. It is obvious that only the rule 2 can be applied to G . By applying the rule 2 to G , G is rewritten into $\rho_1^1 - M_0(\rho_1^1)$. If we look at the mechanism that uniquely rewrites the bridged reaction-SOG ρ_1^1 into the unbridged reaction-SOG $unbridged(\rho)$ (Lemmas 4.2.3 and 4.2.4), we observe that the stimulus strand is not necessary for this rewriting. Thus, proving that the SOG $\rho_1^1 - M_0(\rho_1^1)$ is always uniquely rewritten into the SOG $unbridged(\rho) - M_0(unbridged(\rho))$, is the same as proving that the bridged reaction-SOG ρ_1^1 is always uniquely rewritten into the unbridged reaction-SOG $unbridged(\rho)$. And the proof that the bridged reaction-SOG ρ_1^1 is always uniquely rewritten into the unbridged reaction-SOG $unbridged(\rho)$ is included in Theorem 4.2.6. Once the SOG $\rho_1^1 - M_0(\rho_1^1)$ is rewritten into the SOG $unbridged(\rho) - M_0(unbridged(\rho))$, we obtain the strand $C(unbridged(\rho))$, and the strand $M_1(unbridged(\rho)) \rightarrow \dots \rightarrow M_\omega(\rho)(unbridged(\rho))$. From Lemmas 4.4.3, 4.4.6, and 4.4.9, the strand $C(unbridged(\rho))$ is always uniquely rewritten into the strand $C(final(\rho))$, and the strand $M_1(unbridged(\rho)) \rightarrow \dots \rightarrow M_\omega(\rho)(unbridged(\rho))$ is always uniquely rewritten into the strand $M_1(final(\rho)) \rightarrow \dots \rightarrow M_\omega(\rho)(final(\rho))$ if $\rho \in CReacts$, or into

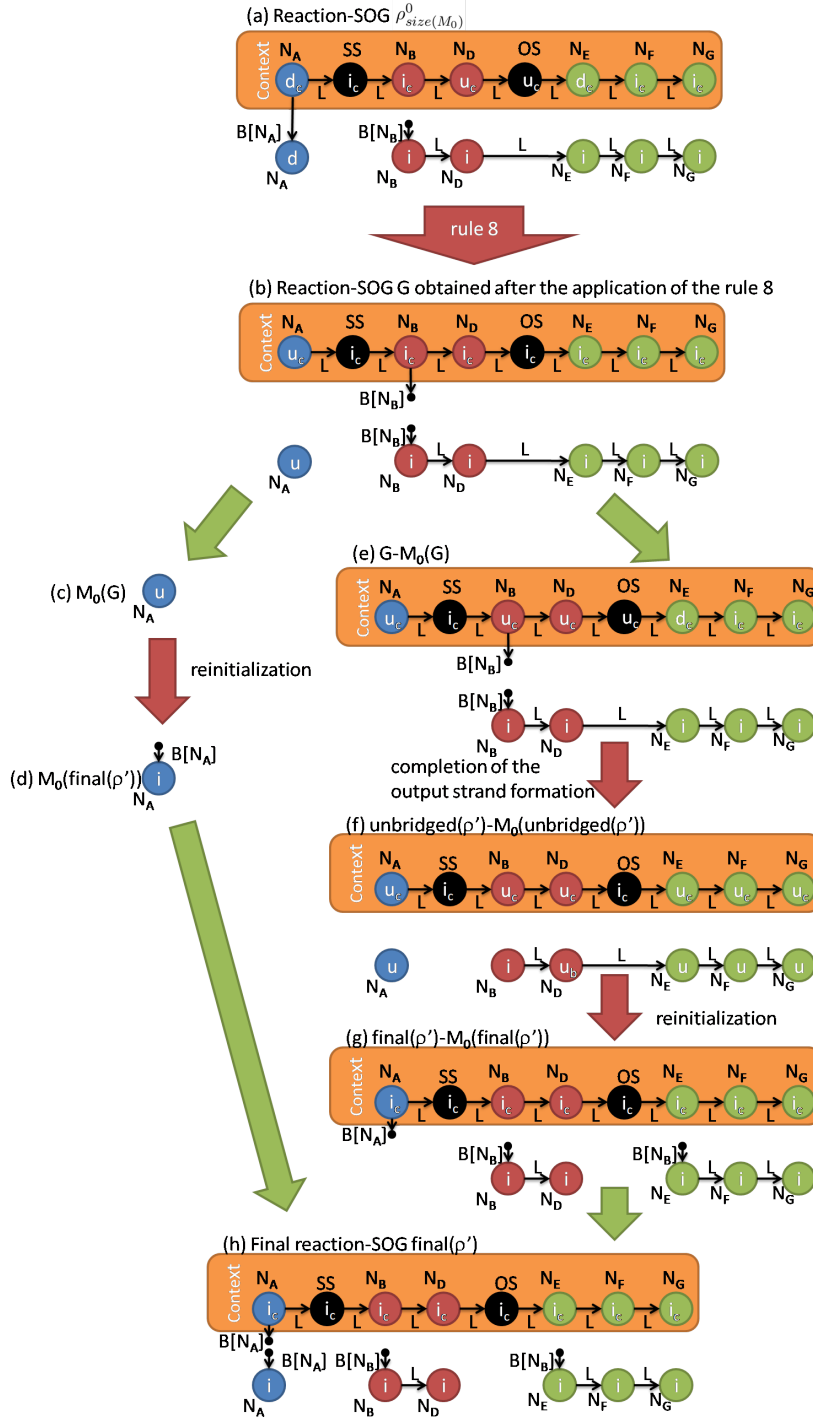


Figure 4.11: An illustration of the confluence found in the output-strand formation stage of a reaction ρ' (Fig. 3.5), if Assumption 4.2.1 does not hold. For readability, the hidden ports are not represented.

the $\omega(\rho)$ strands $M_1(\text{final}(\rho)), \dots, M_{\omega}(\rho)(\text{final}(\rho))$ if $\rho \in D\text{Reacts}$.

If each connection is stable once it is established, and if Assumption 4.2.2 holds once the both processes are completed, we obtain the reaction-SOG $\text{final}(\rho)$. There is a confluence. Thus, we obtain the same final result whether Assumption 4.2.1 holds or not. In practice Assumption 4.2.1 does not hold. However, for simplicity, in our theory we can assume that Assumption 4.2.1 holds.

4.7.2 Assumption “No rule 1”

The only purpose of Assumption 4.4.1 is to stop the context strand from starting another reaction before all the strands are reinitialized. This is to guarantee that for any reaction $\rho \in C\text{Reacts} \cup D\text{Reacts}$, the unbridged reaction-SOG $\text{unbridged}(\rho)$ is always rewritten into the final reaction-SOG $\text{final}(\rho)$. For example, for a reaction ρ' (Fig. 3.5), if the context strand and the stimulus strand are already reinitialized, but the other strands are still reinitializing, we can obtain the partially-reinitialized SOG shown in Fig. 4.12a. If Assumption 4.4.1 does not hold, we can apply the sequence of rules 1-2. By applying this sequence, the context strand and the stimulus strand become bridged and we obtain a SOG G as shown in Fig. 4.12b. If this happens, the SOG G cannot be rewritten anymore into the final reaction-SOG $\text{final}(\rho)$ shown in Fig. 4.12c. However, since the reinitialization of the other strands is independent by definition, the reinitialization of the material strands in the reaction SOG G shown in Fig. 4.12b will continue, and thus, we may obtain the SOG H shown in Fig. 4.12d. If Assumption 4.4.1 holds, the partially-reinitialized reaction-SOG shown in Fig. 4.12a is always rewritten into the final reaction-SOG shown in Fig. 4.12b (see Section 4.4.8). However, by applying the sequence of rules 1-2 to the final reaction SOG shown in Fig. 4.12c, this SOG is rewritten into the SOG H shown in Fig. 4.12d. In other words, all the SOGs that can be obtained if Assumption 4.4.1 does not hold can also be obtained if Assumption 4.4.1 holds. However, for simplicity, in our theory we can assume that Assumption 4.4.1 holds.

4.7.3 Assumption “Simple Proximity”

For any reaction $\rho \in C\text{Reacts} \cup D\text{Reacts}$, the purpose of Assumption 4.2.2 is to guarantee that the initialized reaction-SOG $\text{initialized}(\rho)$ is always uniquely rewritten into the unbridged reaction-SOG $\text{unbridged}(\rho)$, and that $\text{unbridged}(\rho)$ is always rewritten into the final reaction-SOG $\text{final}(\rho)$. If Assumption 4.2.2 does not hold, we may have the two following cases.

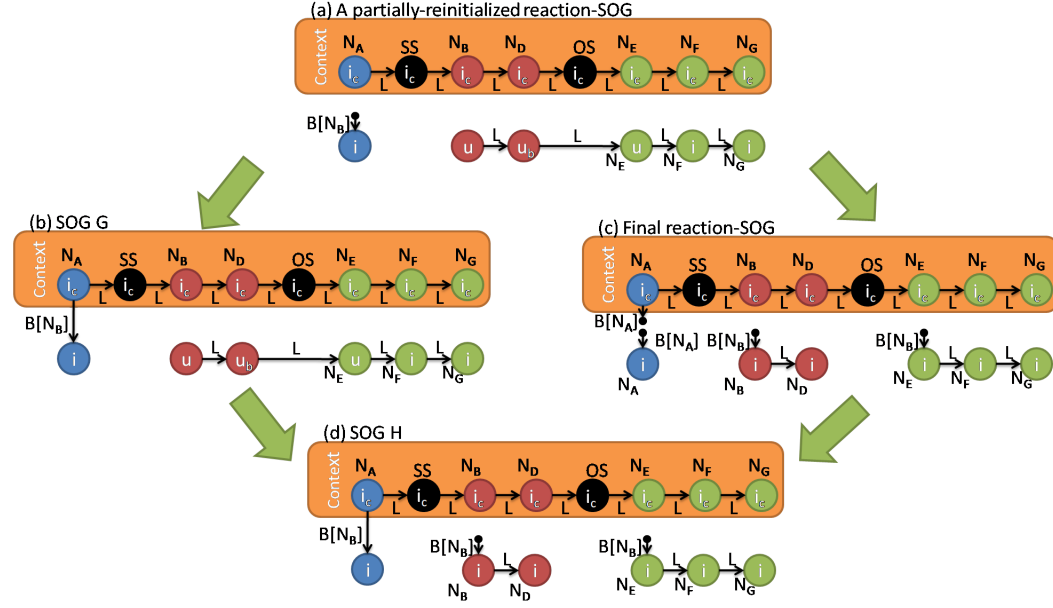


Figure 4.12: An illustration of the equivalence if Assumption 4.4.1 holds or not, for a reaction ρ' (Fig. 3.5). For readability, the hidden ports are not represented.

- Case I: For any $i \in \{\alpha(\rho), \dots, \omega(\rho)\}$, the NSOs of $C_i(\rho)$ may have in their proximity NSOs of other strands than $M_i(\rho)$.
- Case II: For any $i \in \{\alpha(\rho), \dots, \omega(\rho)\}$, the NSOs of $M_i(\rho)$ may be in the proximity of the NSOs of other strands than $C_i(\rho)$.

Let us first assume Case I. This implies that every time a connection B is spanned from the context to the first NSO of a material strand, the context may create a connection to the wrong strand. For example, the first NSO of the strand C_i of the context strand of ρ may span a B connection to the first NSO of a strand M'_i instead of the first NSO of the strand M_i . However, either M'_i is type compatible with M_i , or the strand M'_i is incompatible. If the strand M'_i is type compatible with M_i , the output-strand formation stage and the reinitialization stage will continue with the strand M'_i instead of the strand M_i . Once the reinitialization is completed, we obtain a result similar to the one that we would have obtained with M'_i . If the strand M'_i is incompatible, then from Theorem 4.6.10, we know that the strand C_i , the strand M_i , and each strand $M_\alpha(\rho), \dots, M_{i-1}$ will be reset. Once these strands are reset, the output-strand formation stage restart, and we have another chance to span a B connection to the right strand, i.e., the strand M_i .

Before considering the Case II, we give the following definitions.

Definition 47 (Partially-reinitialized-incomplete reaction-SOG). *For any reaction $\rho \in CReacts \cup DReacts$, a partially-reinitialized-incomplete reaction-SOG of ρ is an SOG G that consists only of a partially-reinitialized reaction-SOG of ρ from which we removed at least one initialized strand.*

Definition 48 (Final-incomplete reaction-SOG). *For any reaction $\rho \in CReacts \cup DReacts$, a final-incomplete reaction-SOG of ρ is an SOG G that consists only of the final reaction-SOG of ρ from which we removed at least one strand.*

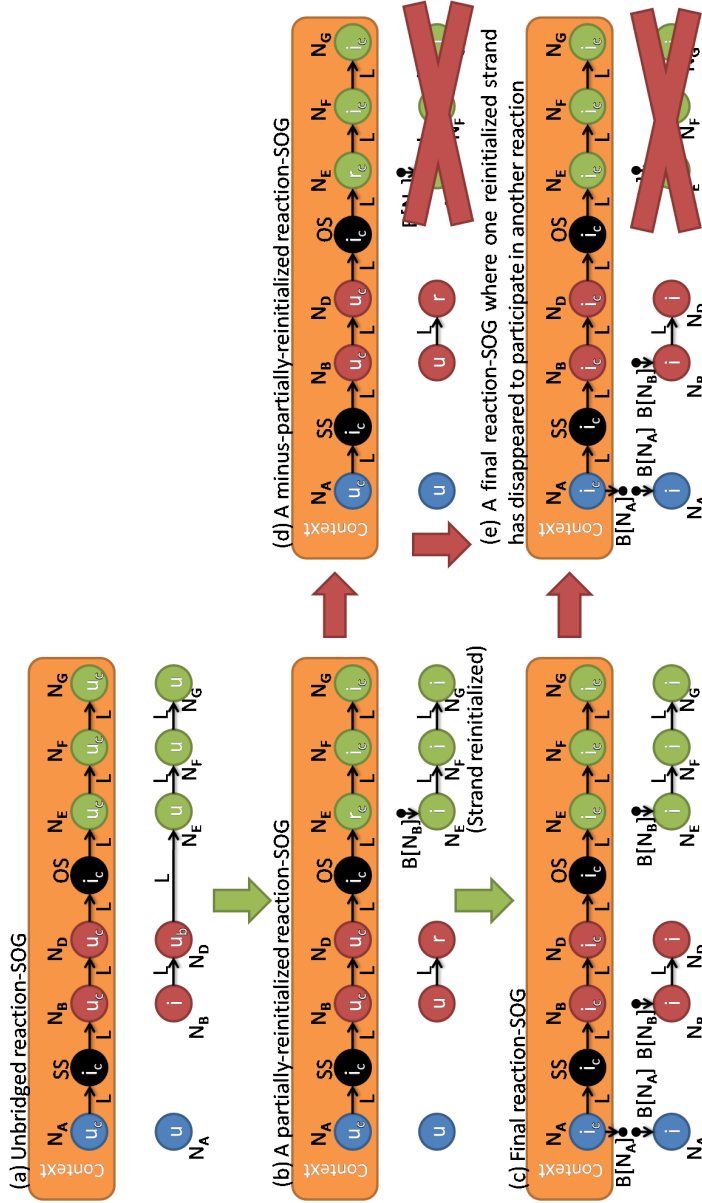
Let us now assume Case II. This implies that as soon as a strand is reinitialized, this strand may participate in another reaction. If Assumption 4.2.2 holds, we know from Theorem 4.4.11 that, for any $\rho \in CReacts \cup DReacts$, the reaction-SOG $unbridged(\rho)$ is always rewritten into the reaction-SOG $final(\rho)$. For example, in Fig. 4.13, the reaction-SOG shown in Fig. 4.13a is always rewritten into the reaction-SOG shown in Fig. 4.13c.

During this rewriting, we obtain partially-reinitialized reaction-SOG, that are always rewritten into the reaction-SOG $final(\rho)$ (see Section 4.4.8). For example, the reaction-SOG $unbridged(\rho)$ in Fig. 4.13a, may be first rewritten into the partially-reinitialized reaction-SOG shown in Fig. 4.13b, and then will always be rewritten into the reaction-SOG $final(\rho)$ shown in Fig. 4.13c. However, if Assumption 4.2.2 does not hold, a partially-reinitialized-reaction SOG of ρ is not always rewritten into the final reaction-SOG of ρ , because its strand may participate in another reaction.

If this happens, we consider that the strands participating in another reaction are removed from the partially-reinitialized-reaction SOG. For example, the SOG shown in Fig. 4.13b, may be rewritten into the reaction-SOG $final(\rho)$, or it may be rewritten into the partially-reinitialized-incomplete reaction-SOG shown in Fig. 4.13d. If the unbridged reaction-SOG $unbridged(\rho)$ is rewritten into a partially-reinitialized-incomplete reaction-SOG, it is not guaranteed that this SOG will be rewritten into the reaction-SOG $final(\rho)$. However, for any reaction $\rho' \in DReacts$, for any $h = 1, \dots, \omega(\rho')$, and for any $n = 1, \dots, size(M_h(\rho'))$, if we look at the mechanism reinitializing the strands $M_1, \dots, M_h$ (Lemmas 4.4.7 and 4.4.8), we can observe that the reinitialization of the strands $M_1, \dots, M_h$ is independent on each strand M_j , for any $j \in \{h+1, \dots, \omega(\rho')\}$. This implies that the strands $M_1, \dots, M_h$ are always reinitialized, independently of the participation in another reaction or not of each strand M_j .

Moreover, for any $\rho \in CReacts \cup DReacts$, the reinitialization of the context strand, of the stimulus strand, and of the output strand(s) are independent. All this implies that for any partially-reinitialized-incomplete reaction-SOG G of a reaction

ρ , the reinitializing strands in G always complete their reinitialization, independently of the participation in other reactions or not of some strands of G . By doing so, G is rewritten into a final-incomplete reaction-SOG H of ρ . For example, the partially-reinitialized-incomplete reaction-SOG shown in Fig. 4.13d may be rewritten into the final-incomplete reaction-SOG shown in Fig. 4.13e. By definition, if we remove strand from the final-reaction-SOG of ρ , we can also obtain the same final-incomplete reaction-SOG. For example, the final reaction-SOG shown in Fig. 4.13c may also be rewritten into the final-incomplete reaction-SOG shown in Fig. 4.13e by removing a strand. In other words, for any reaction $\rho \in CReacts \cup DReacts$, all the SOGs obtained by reinitializing the strands of a partially-reinitialized-incomplete reaction-SOGs of ρ , can also be obtained by removing some strands from the final reaction-SOG $final(\rho)$, and vice versa.



In theory, the unbridged reaction-SOG (a) of a reaction is always rewritten into a partially-reinitialized reaction-SOG (b). And this partially-reinitialized reaction-SOG will always be rewritten into the final reaction-SOG (d) of the reaction. However, in practice, a strand is allowed to participate in other reactions as soon as it is reinitialized. Thus the SOG shown in (b) may be rewritten into the SOG shown in (d). Since the reinitialized strands are not needed for the completion of the reinitialization of the remaining strands, the SOG shown in (d) will still be rewritten into the SOG shown in (e). However, the final reaction-SOG (c) can also be rewritten into the SOG shown in (e). So the SOGs reachable from (a) are the same independent of if strands can participate in others reaction as soon as they are reinitialized or not.

Figure 4.13: An illustration comparing to different types of system behaviour for a reaction ρ' (Fig. 3.5) depending on whether Assumption 4.2.2 or not. For readability, the hidden ports are not represented.

However, having or not Assumption 4.2.2 that holds is not exactly the same. Indeed, it is possible to have a deadlock situation if this assumption does not hold. For example, let $\rho_1 = \langle nil, B_{\rho_1} \rangle_C$ and $\rho_2 = \langle nil, B_{\rho_2} \rangle_C$ be two composition reactions where $B_{\rho_1} = (A, B)$ and $B_{\rho_2} = (B, A)$. Let $C_{\rho_1} : C_1^{\rho_1} \rightarrow s_1^{\rho_1} \rightarrow C_2^{\rho_1}$ be the context strand representation of ρ_1 , where $type(s_1^{\rho_1}) = IS$. Let $C_{\rho_2} : C_1^{\rho_2} \rightarrow s_1^{\rho_2} \rightarrow C_2^{\rho_2}$ be the context strand representation of ρ_2 , where $type(s_1^{\rho_2}) = IS$. Let M_A and M_B be the material-strand representation of A and B , respectively. Let consider the instance SOG G that consists of only the initialized context-strands of C_{ρ_1} and C_{ρ_2} , the initialized visible-strands of M_A and M_B , and nothing else. Finally, let us assume that M_A is in the proximity of $C_1^{\rho_1}$ and of $C_2^{\rho_2}$, and that M_B is in the proximity of $C_2^{\rho_1}$ and of $C_1^{\rho_2}$. An illustration of G is given in Fig. 4.14. Then, by applying the sequence of rules 1-2-3, it is possible, to span a B connection from the first NSO of the instance of C_{ρ_1} to the first NSO of the instance of the M_A . After this, by applying the same sequence, it is possible to span a B connection from the first NSO of the instance of C_{ρ_2} to the first NSO of the instance of the M_B . Thus, G can be rewritten into the SOG H shown in Fig. 4.15. If we obtain H , it is obvious that we are in a deadlock situation. The reaction ρ_1 needs the strand M_B initialized to pursue the output-strand formation, and the reaction ρ_2 needs the strand M_A initialized to pursue the output-strand formation. For a system implementing our theory, to have equivalence between having or not Assumption 4.2.2 that holds, it is necessary to add a feature that releases the strands after a certain timeout. If such feature exists, once the strands are released and reinitialized, we have another chance to have one of the reactions that happens.

4.8 Implementation

As a proof of concept, we implemented our framework using the Sun SPOT units¹. This implementation was presented in [17]. A Sun SPOT unit is a small device that uses the IEEE 802.15.4 standard for wireless communication. This standard is the basis of the ZigBee protocol. A Sun SPOT unit also possesses some sensors (i.e., light sensor, accelerometer, and temperature sensor), and LEDs to display information. They integrate a JVM and thus are easily programmable in Java.

We show the architecture of our implementation in Fig. 4.18. We divided this architecture into three layers. The first layer, e.i., the base smart object layer, provides all the features common to all the SOs. This layer includes four main modules. The

¹<http://www.sunspotworld.com/>

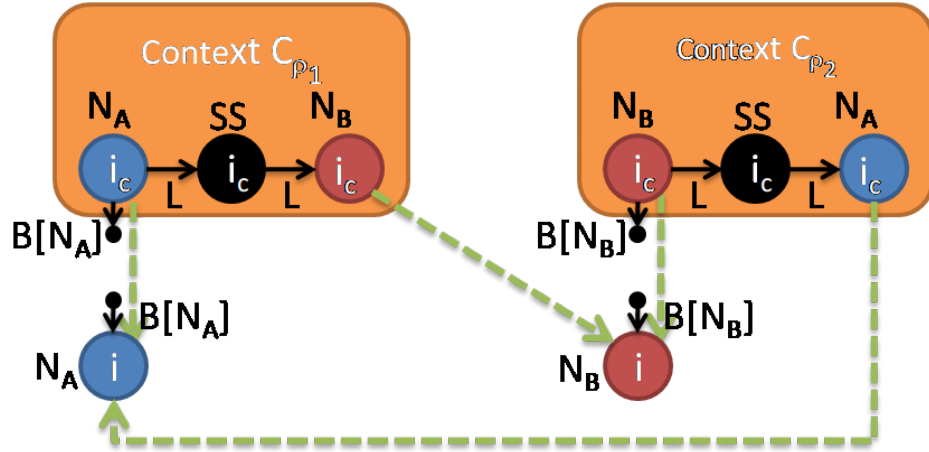


Figure 4.14: An illustration of an SOG G that may lead to a deadlock situation. For readability, the hidden ports are not represented.

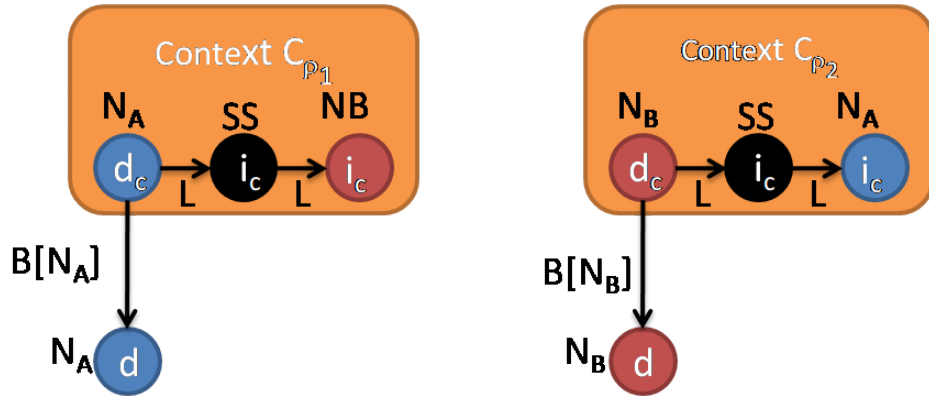


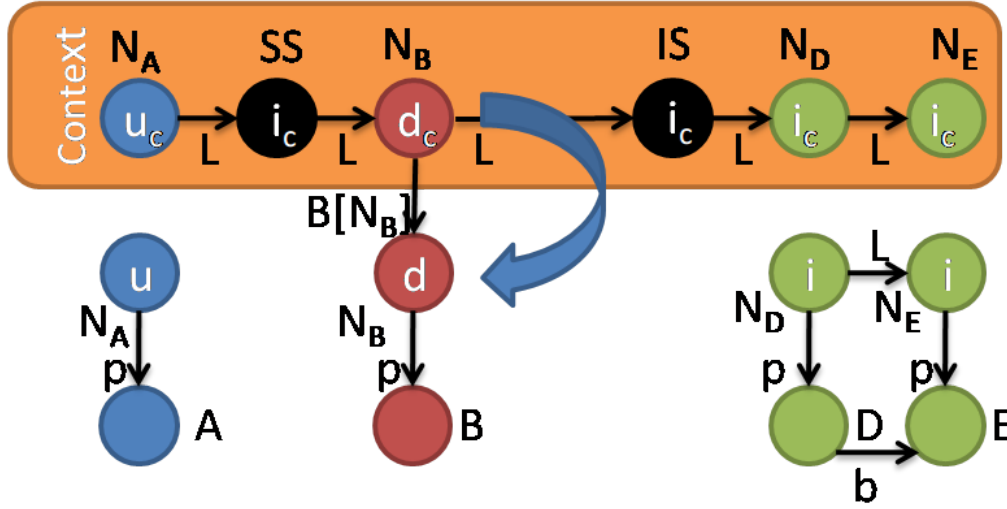
Figure 4.15: An illustration of an SOG I that is in a deadlock situation. For readability, the hidden ports are not represented.

first module, the core module, provides all the properties of the SOs, i.e., the types, the states, the ports, etc. The second module, the rule engine module, executes the rules of the SOs. It implements all the operations given in Section 3.2.5. The third module, the abstract communication module, manages the message exchange communication among the SOs. This layer is independent from the type of hardware used. That is why the communication module is said to be abstract. It is the role of the second layer, i.e., the hardware layer, to implement the JAVA interfaces of the abstract communication module to allow communication. This layer is dependent on the hardware, and provides encapsulated methods used by the first layer to make wireless communication. It also provides the module user interface to display appropriate information. To do that, the core module implements an observer pattern. Through this pattern, the user interface module can receive events from the core module. Such an event is for example the state change of a port. The third layer, i.e., the instance layer, corresponds to the instantiation and the integrated wrapping of the core module, the rule engine module, the communication module, and the user interface module. It is in this layer that the developer will define the properties of the SOs (states, types, etc. . .) by instantiating the core module, and the rules of the SO by instantiating the rule engine module. It is also in this layer that the developer refers to the second layer for the material dependent communication and user interface. By referring to another implementation of the second layer, we can easily change the type of hardware we use (assuming that this material supports a JVM). For example we can have two implementations of the second layer, one for the Sun SPOT units, and the other for the Android devices. The first implementation would provide a communication module that uses the IEEE 802.15.4 protocol, and the user interface module that uses the LEDs of the Sun SPOTs to display information as shown in Fig. 4.19. The second implementation would provide a communication module that uses the WiFi protocol, and the user interface module that uses the touch screen of an android device to display information.

4.9 Establishing the Connections among the Application Smart Objects

During the reaction, some context NSOs may send a program code to the materials NSOs to which there are docked. This program code will be executed once the material strands are reinitialized to establish or to break the connections among the application smart objects.

For any composition reaction $\rho \in CReacts$, only one program code is sent during the reaction process. This happens when we obtain the SOG ρ_1^1 . In this SOG ρ_1^1 , it is the first NSO of the strand $C_1(\rho_1^1)$ that sends the program code to the first NSO of the strand $M_1(\rho_1^1)$ through the connection B as show in the example in Fig. 4.16a. Once the reaction is completed, we obtained the SOG $G = final(\rho)$. In G , the first NSO of the strand $M_1(G)$ has a directed-path to all the application smart objects as shown in the example in Fig. 4.16c. That means the program code in this NSO can access all the application smart objects, and thus can established or break the necessary connections among these application smart objects as shown in the example in Fig. 4.16d.



(a) An example of a program code being sent from the context to the future output strand in a SOG ρ_1^1 .

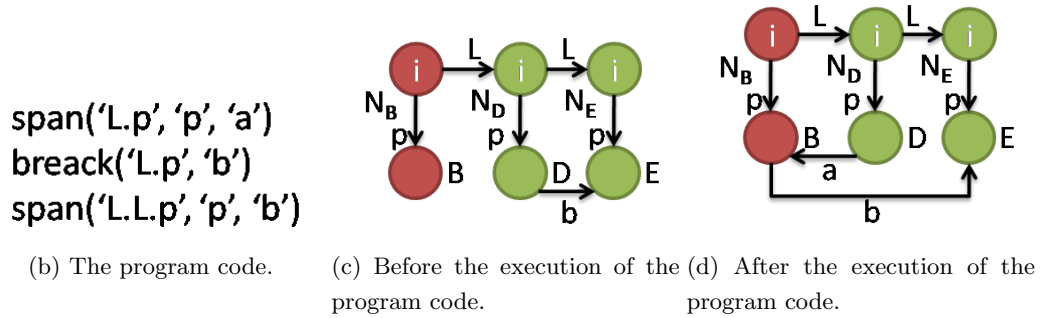


Figure 4.16: Example of program code for a composition reaction ρ . For readability, the hidden port are not shown.

Chapter 4. Validity Proof of Rules for Reactions

For any decomposition reaction $\rho' \in DReacts$, $\omega(\rho')$ program codes are sent during the reaction process. Every time we obtain an SOG ρ_1^h , a program code is sent from the first NSO of the strand $C_h(\rho_1^h)$ to the first NSO of the strand $M_h(\rho_1^h)$ through the connection B, for any $h = 1, \dots, \omega(\rho')$, as shown in the example in Figs. 4.17a and 4.17b. Once the reaction is completed, we obtained the SOG $G' = final(\rho')$. In G' , the first NSO of each strand $M_1(G'), \dots, M_\omega(G')$ has a directed-path to all the application smart objects of this strand as shown in the example in Fig. 4.17e. That means the program code in the first NSO of each of these strands can access all the application smart objects of each of these strands, and thus can established or break the necessary connections among these application smart objects as shown in the example in Fig. 4.17f.

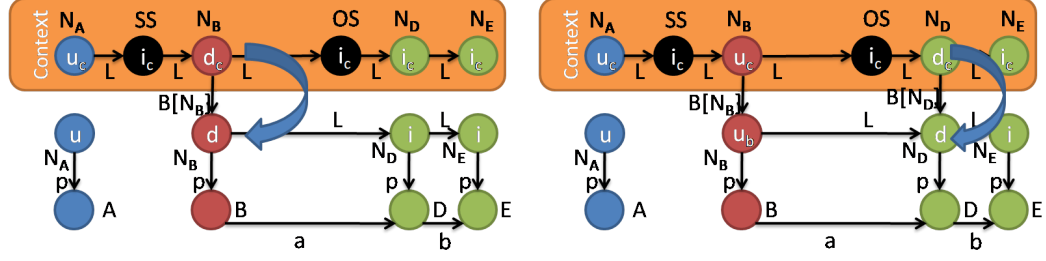
A program code is deleted immediately after being executed.

We have developed an implementation of the second layer for the Android devices supporting the Wi-Fi Direct² protocol. The Wi-Fi Direct protocol allows more than one connection simultaneously. However, the Wi-Fi Direct protocol is not a pure P2P system and assumes that one peer is the owner of the network. Thus, the implementation using the Wi-Fi Direct protocol does not completely match with our implementation compared to implementation using the SunSPOT units. However, as long as all the peers stay connected to the owner of the network, this implementation can support and demonstrate the applications using our framework. It also demonstrates that the base smart object layer can be easily reused.

More generally, there are two major ways to implement our framework. The first one is to use a hardware supporting more than one wireless connection simultaneously to implement our framework. A hardware supporting only one connection at the same time will still be compatible with this implementation of our framework, but can be only used to implement an SO with only one port. For example, a material supporting Wi-Fi protocol, but not Wi-Fi Direct protocol, can still communicate with a material supporting the Wi-Fi Direct Protocol. The second way to implement our framework is to use a hardware device that has broadcasting capability. It is possible to simulate more than one connection at the same time by using this broadcasting capability. However, the performance will certainly become very poor, and this type of implementation is incompatible with the previous one.

In our implementations, the scope of each smart object corresponds to the maximum range of the wireless communication capacity of their hardware device. Research as [24, 22, 46, 45, 1, 8, 56, 34] can help us to determine the distance among the

²<http://www.wi-fi.org/discover-wi-fi/wi-fi-direct>

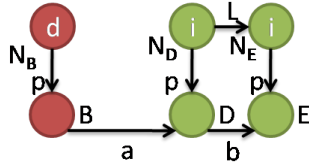


(a) An example of a program code being sent from the context to a future output strand in a SOG ρ_1^1 . (b) An example of a program code being sent from the context to a future output strand in a SOG ρ_1^2 .

break(", 'a')

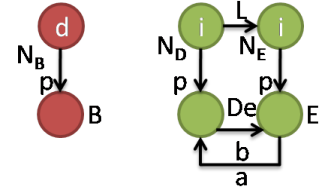
span('L.p', 'p', 'a')

(c) The program code sent to the NSO of type N_B .



(e) Before the execution of the program code.

(d) The program code sent to the NSO of type N_D .



(f) After the execution of the program code.

Figure 4.17: Example of program code for a decomposition reaction ρ' . For readability, the hidden ports are not shown.

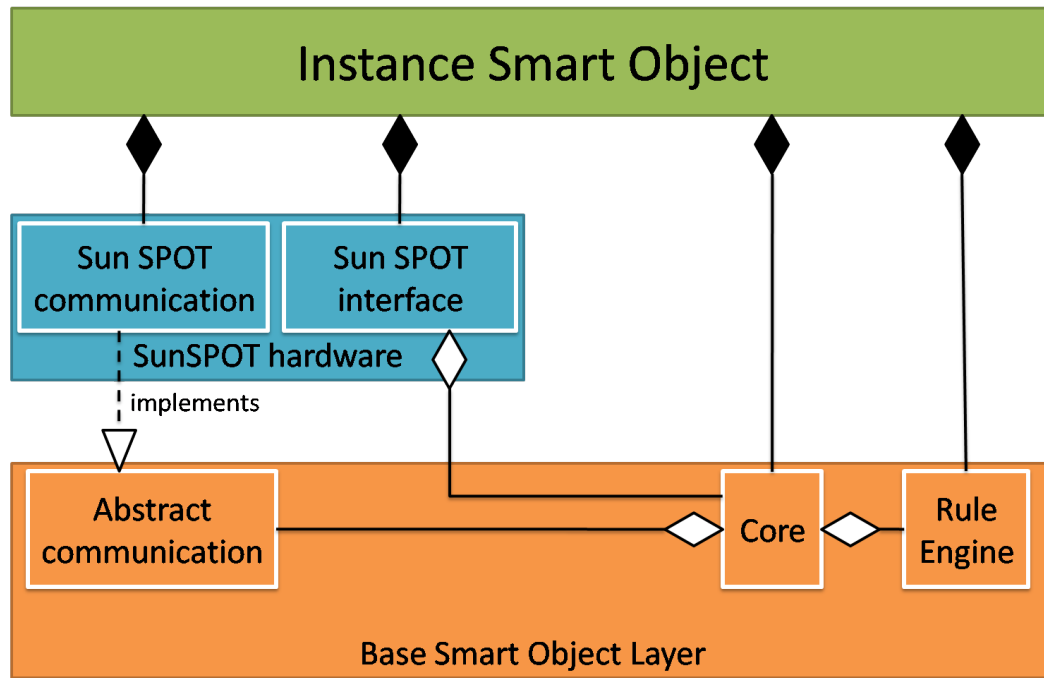


Figure 4.18: The architecture of our implementation.

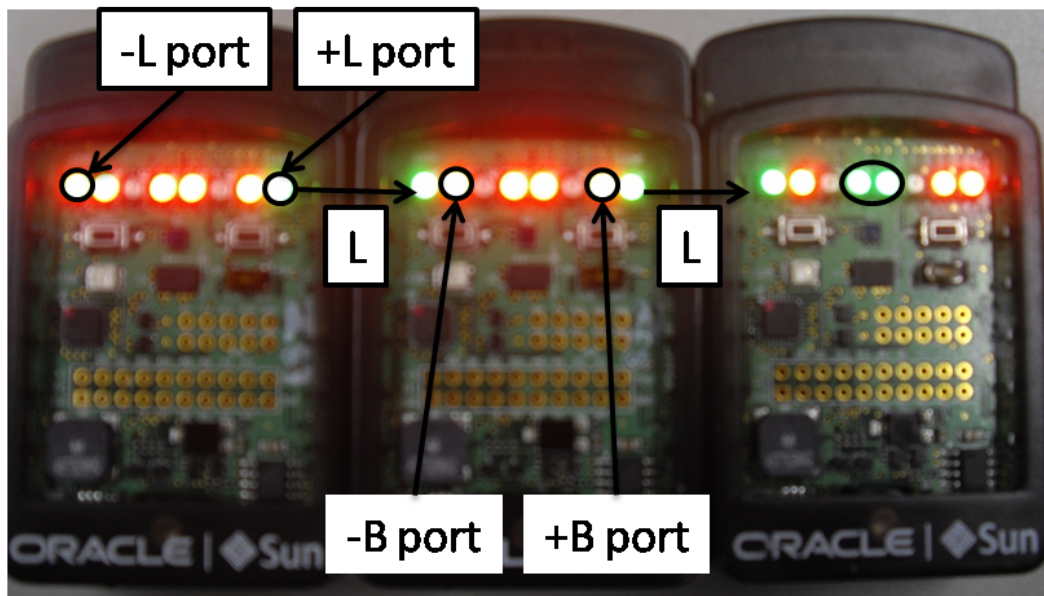


Figure 4.19: Example of a context build with some Sun SPOTs.

smart object with accuracy. We are confident that soon an accurate distance sensing method that can be executed by cheap smart objects and that are fast enough for our scenarios will be released. Then, we will be able to fix the size of the scope of each smart object with accuracy, that may be useful for specific applications.

Chapter 4. Validity Proof of Rules for Reactions

Chapter 5

Conclusion

In the present thesis, we presented and prove the validity of a new formal model for smart object. This formal model deals with the following three different levels, where each level is built on top of its previous level:

- At the first level, or at the bottom level, the port-matching model describes the federation and inter-operation mechanism between two SOs.
- At the second level, the graph-rewriting rules describe the dynamic change of federation structures among SOs.
- At the third level, the catalytic-reaction-network model describes complex application scenarios involving mutually related multiple federations.

This formal model allows the smart objects to dynamically and autonomously reconstruct their federation configurations depending on their location and context change. To achieve this, no centralized control is used.

The third model, i.e., the catalytic reaction network system, is used to model scenarios involving complex federations of SOs where each material denotes an SO or a federation of SOs. The first major contribution of this thesis is to provide this model as well as a framework implementing our third level with the second level. This implementation is inspired by a biological RNA replication mechanism.

The second major contribution of this thesis is the proof of the validity of the implementation of catalytic-reaction network by graph-rewriting rules. It was shown that proving the properties such as confluence or termination of graph rewriting systems is not possible in general [37, 38]. However, our system only deals with a certain type of graphs. To prove the validity of our graph rewriting system, we

Chapter 5. Conclusion

needed to formally represent the class of graphs to which our graph rewriting rules are applied. These graphs all share the same type of structures called the double-strand structures, as shown in Fig. 1.2. Proving the validity of our system for this type of graphs is possible, unlike proving it in general.

A system based on our theory has several advantages. It is robust and scalable. Each scenario implemented by such a system is easy to modify. Even if there are several copies of the same context, each federation reaction uses only one of them. Multiple copies of the same context, however, may increase the robustness of the federation mechanism. Even if one of them may malfunction and get terminated, the others can continue to make federation. For the scalability, whenever a certain context is overloaded, you may add more copies of this context to immediately increase the concurrent operation capacity. Each scenario is easy to modify because different contexts deal with different federation reactions, and each of them can be added or removed independently from the others.

In ubiquitous computing, we need formal computation models and middlewares to describe and to develop complex applications. Process calculi can model the behaviour of mobile objects in more detail than our model. However one big difference between these models and our model is that they are not focused on the interconnection structure of mobile objects and the dynamical change of this structure.

In the middleware world, researchers try to provide middleware systems to simplify the development of pervasive environments. Conventional middleware systems provide many useful tools and services to develop ubiquitous systems. However, they lack a high-level-abstract model necessary to describe complex scenarios. On the other hand, process calculi are too abstract to describe the dynamic interconnection changes among mobiles objects.

To summarize, existing solutions for modelling and implementing ubiquitous computing system either lack high-level-abstract model required to describe complex application scenarios, or do not focus on the interconnection structure of mobile objects and the dynamical change of this structure. Both are required to consider, to describe, and to implement innovative scenarios beyond the current scope of stereotyped applications of ubiquitous, pervasive, and/or mobile computing. Moreover, most of the existing solutions assume centralized control, that limit the flexibility of a system. To answer these problems, this thesis propose a formal model that allows the smart objects to dynamically and autonomously reconstruct their federation configurations depending on their location and context change. To achieve this, no centralized control is used. This formal model deals with three different

levels, where each level is built on top of the underlying one. The first level, or the bottom level, of our formal model, called the port-matching model, describes the federation and inter-operation mechanism between two SOs. The second level, the graph-rewriting model, describes the dynamic change of federation structures. This level is used to describe the interconnection structure of mobile objects and the dynamical change of this structure. The third level, the catalytic-reaction-network model corresponds to a high-level-abstract model that describes complex application scenarios involving multiple mutually related federations. To implement the third level, we use the second level. This implementation is inspired by a biological RNA replication mechanism. In our framework, this replication mechanism is represented by graph rewriting rules. The validity of the implementation of this third level was formally proved. We also provide solution to build contexts quickly. Since our system is decentralized, each scenario implemented by such a system is easy to modify because different contexts deal with different composition or decomposition reactions, and each of them can be added or removed independently from the others. This property make such a system easy to scale by adding copies of contexts, and robust.

With this new approach based on the RNA replication mechanism, and with the representation of the interactions among smart objects based on catalytic-reaction-networks, we can consider, describe, and implement innovative scenarios beyond the current scope of stereotyped applications of ubiquitous, pervasive, and/or mobile computing.

Related Publications

- [1] Jérémie Julia and Yuzuru Tanaka. Proximity-Based Federation of Smart Objects: Its Graph-rewriting Framework and Correctness. *Journal of Intelligent Information Systems*, pages 132, 2015.
- [2] Jérémie Julia, Yuzuru Tanaka and Nicolas Spyrtos. Formalization of an RNA-inspired middleware for complex smart object federation scenarios. In *PECCS 2012 - Proceedings of the 2nd International Conference on Pervasive Embedded Computing and Communication Systems*, pages 86-105, Rome, Italy, February 2012.
- [3] Jérémie Julia and Yuzuru Tanaka. Improvisational construction of a context for dynamic implementation of arbitrary smart object federation scenarios. In *In proceedings of 2012 IEEE International Conference on Multimedia and Expo Workshops (ICME-2012)*, pages 223-229, Melbourne, Australia, July 2012.
- [4] Jérémie Julia and Yuzuru Tanaka. Specifying the federation structure among application smart objects by example through direct manipulations. In *In Post-workshop Proceedings of Information Search, Integration and Personalization (ISIP 2012), volume 146 of Communications in Computer and Information Science*, pages 31-40. Sapporo, Japan, October 2012. Springer Berlin Heidelberg.

Bibliography

- [1] I. Amundson, J. Sallai, X. Koutsoukos, A. Ledeczi, and M. Maroti. Rf angle of arrival-based node localisation. *Int. J. Sen. Netw.*, 9(3/4):209–224, May 2011.
- [2] D. Blostein, H. Fahmy, and A. Grbavec. Issues in the practical use of graph rewriting. In *Selected papers from the 5th International Workshop on Graph Grammars and Their Application to Computer Science*, pages 38–55, London, UK, UK, 1996. Springer-Verlag.
- [3] L. Cardelli and A. D. Gordon. Mobile ambients. In *FoSSaCS*, pages 140–155, 1998.
- [4] T. R. Cech. The rna worlds in context. *Cold Spring Harbor Perspectives in Biology*, 2011.
- [5] S. Chetan, J. Al-Muhtadi, R. Campbell, and M.D. Mickunas. Mobile gaia: a middleware for ad-hoc pervasive computing. In *Consumer Communications and Networking Conference, 2005. CCNC. 2005 Second IEEE*, pages 223–228, Jan 2005.
- [6] A. Corradini. Concurrent graph and term graph rewriting. In Ugo Montanari and Vladimiro Sassone, editors, *CONCUR '96: Concurrency Theory*, volume 1119 of *Lecture Notes in Computer Science*, pages 438–464. Springer Berlin Heidelberg, 1996.
- [7] J. Crowcroft. Engineering global ubiquitous systems. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 366:3833–3834, 2008.
- [8] B. J. Dil and P. J. M. Havinga. Stochastic radio interferometric positioning in the 2.4 ghz range. In *Proceedings of the 9th ACM Conference on Embedded Networked Sensor Systems, SenSys '11*, pages 108–120, New York, NY, USA, 2011. ACM.

Bibliography

- [9] H. Ehrig, M. Pfender, and H.J. Schneider. Graph-grammars: An algebraic approach. In *Switching and Automata Theory, 1973. SWAT '08. IEEE Conference Record of 14th Annual Symposium on*, pages 167–180, Oct 1973.
- [10] H. Fahmy and R.C. Holt. Using graph rewriting to specify software architectural transformations. In *Automated Software Engineering, 2000. Proceedings ASE 2000. The Fifteenth IEEE International Conference on*, pages 187–196, 2000.
- [11] J. Feder. Plex languages. *Inf. Sci.*, 3(3):225–241, 1971.
- [12] M. Fernndez, H. Kirchner, and O. Namet. A strategy language for graph rewriting. In Germn Vidal, editor, *Logic-Based Program Synthesis and Transformation*, volume 7225 of *Lecture Notes in Computer Science*, pages 173–188. Springer Berlin Heidelberg, 2012.
- [13] D. Gelernter. Generative communication in linda. *ACM Transactions on Programming Languages and Systems*, 7:80–112, 1985.
- [14] R. Grimm, J. Davis, E. Lemar, A. Macbeth, S. Swanson, T. Anderson, B. Bershad, G. Borriello, S. Gribble, and D. Wetherall. System support for pervasive applications. *ACM Trans. Comput. Syst.*, 22(4):421–486, November 2004.
- [15] K. Henriksen, J. Indulska, and A. Rakotonirainy. Modeling context information in pervasive computing systems. in *F. Mattern and M. Naghshineh (Eds.): Pervasive 2002, LNCS*, 2414:167–180, 2002.
- [16] J. Julia and Y. Tanaka. Improvisational construction of a context for dynamic implementation of arbitrary smart object federation scenarios. In *2012 IEEE International Conference on Multimedia and Expo Workshops*, pages 223–229, Melbourne, Australia, July 2012.
- [17] J. Julia and Y. Tanaka. Specifying the federation structure among application smart objects by example through direct manipulations. In *Information Search, Integration and Personalization - International Workshop (ISIP 2012)*, volume 146 of *Communications in Computer and Information Science*, pages 31–40, Sapporo, Japan, October 2012. Springer Berlin Heidelberg.
- [18] J. Julia and Y. Tanaka. Proximity-based federation of smart objects. *Journal of Intelligent Information Systems*, pages 1–32, 2015.

- [19] J. Julia, Y. Tanaka, and N. Spyratos. Formalization of an rna-inspired middleware for complex smart object federation scenarios. In *PECCS 2012 - Proceedings of the 2nd International Conference on Pervasive Embedded Computing and Communication Systems*, pages 96–105, Rome, Italy, February 2012.
- [20] S. A. Kauffman. *Investigations*. Oxford University Press, 2000.
- [21] M. Kumar, B. Shirazi, S.K. Das, B.Y. Sung, D. Levine, and M. Singhal. Pico: a middleware framework for pervasive computing. *Pervasive Computing, IEEE*, 2(3):72–79, July 2003.
- [22] B. Kusy, G. Balogh, J. Sallai, Á. Lédeczi, and M. Maróti. Intrack: High precision tracking of mobile sensor nodes. In *Proceedings of the 4th European Conference on Wireless Sensor Networks, EWSN’07*, pages 51–66, Berlin, Heidelberg, 2007. Springer-Verlag.
- [23] M. Lwe. Algebraic approach to single-pushout graph transformation. *Theoretical Computer Science*, 109(12):181 – 224, 1993.
- [24] M. Maróti, P. Völgyesi, S. Dóra, B. Kusý, A. Nádas, Á. Lédeczi, G. Balogh, and K. Molnár. Radio interferometric geolocation. In *Proceedings of the 3rd International Conference on Embedded Networked Sensor Systems, SenSys ’05*, pages 1–12, New York, NY, USA, 2005. ACM.
- [25] R. Milner. *Communicating and Mobile Systems: The π -calculus*. Cambridge University Press, New York, NY, USA, 1999.
- [26] R. Milner. Bigraphical reactive systems: Basic theory. In *Proceedings of the International Congress of Mathematicians*, pages 155–169, 2001.
- [27] R. Milner. Theories for the global ubiquitous computer. *Foundations of Software Science and Computation Structures, LNCS*, 2987:5–11, 2004.
- [28] R. Milner. Scientific foundation for global computing. *Transactions on Computational Systems Biology IV*, pages 1–13, 2006.
- [29] R. Milner. Ubiquitous computing: Shall we understand it? *Comput. J.*, 49:383–389, 2006.
- [30] U. Montanari and M. Sammartino. Network conscious π -calculus: A concurrent semantics. *Electr. Notes Theor. Comput. Sci.*, pages 291–306, 2012.

Bibliography

- [31] M. Nagl. Formal languages of labelled graphs. *Computing*, 16(1-2):113–137, 1976.
- [32] M. Nagl. Graph rewriting systems and their application in biology. In Jrgen Berger, WolfgangJ. Bhler, Rudolf Repges, and Petre Tautu, editors, *Mathematical Models in Medicine*, volume 11 of *Lecture Notes in Biomathematics*, pages 135–156. Springer Berlin Heidelberg, 1976.
- [33] T. Pavlidis. Linear and context-free graph grammars. *J. ACM*, 19(1):11–22, January 1972.
- [34] C. Peng, G. Shen, and Y. Zhang. Beepbeep: A high-accuracy acoustic-based system for ranging and localization using cots devices. *ACM Trans. Embed. Comput. Syst.*, 11(1):4:1–4:29, April 2012.
- [35] J. L. Pfaltz and A. Rosenfeld. Web grammars. In *Proceedings of the 1st International Joint Conference on Artificial Intelligence, Washington, DC, May 1969*, pages 609–620, 1969.
- [36] G. P. Picco, A. L. Murphy, and G.-C. Roman. Lime: Linda meets mobility. In *ICSE’99*, pages 368–377, 1999.
- [37] D. Plump. Termination of graph rewriting is undecidable. *Fundam. Inform.*, 33(2):201–209, 1998.
- [38] D. Plump. Checking graph-transformation systems for confluence. *ECEASST*, 26, 2010.
- [39] V. Raychoudhury, J. Cao, M. Kumar, and D. Zhang. Middleware for pervasive computing: A survey. *Pervasive Mob. Comput.*, 9(2):177–200, April 2013.
- [40] T. Rodden. Living in a ubiquitous world. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 366:3837–3838, 2008.
- [41] P.J. Rodgers and P.J.H. King. A graph-rewriting visual language for database programming. *Journal of Visual Languages and Computing*, 8(56):641 – 674, 1997.
- [42] F. Rossell and G. Valiente. Graph transformation in molecular biology. In Hans-Jrg Kreowski, Ugo Montanari, Fernando Orejas, Grzegorz Rozenberg, and Gabriele Taentzer, editors, *Formal Methods in Software and Systems Modeling*,

- volume 3393 of *Lecture Notes in Computer Science*, pages 116–133. Springer Berlin Heidelberg, 2005.
- [43] G. Rozenberg, editor. *Handbook of Graph Grammars and Computing by Graph Transformation: Volume I. Foundations*. World Scientific Publishing Co., Inc., River Edge, NJ, USA, 1997.
 - [44] D. Saha and A. Mukherjee. Pervasive computing: A paradigm for the 21st century. *Computer*, 36(3):25–31, March 2003.
 - [45] J. Sallai, Á. Lédeczi, I. Amundson, X. Koutsoukos, and M. Maróti. Using rf received phase for indoor tracking. In *Proceedings of the 6th Workshop on Hot Topics in Embedded Networked Sensors*, HotEmNets ’10, pages 13:1–13:5, New York, NY, USA, 2010. ACM.
 - [46] J. Sallai, P. Volgyesi, and A. Ledeczi. Radio interferometric quasi doppler bearing estimation. In *Proceedings of the 2009 International Conference on Information Processing in Sensor Networks*, IPSN ’09, pages 325–336, Washington, DC, USA, 2009. IEEE Computer Society.
 - [47] M. Satyanarayanan. Pervasive computing: Vision and challenges. *IEEE Personal Communications*, 8:10–17, 2001.
 - [48] J. Sousa and D. Garlan. Aura: An architectural framework for user mobility in ubiquitous computing environments. In *Software Architecture: System Design, Development, and Maintenance (Proceedings of the 3rd Working IEEE/IFIP Conference on Software Architecture)*, pages 29–43, Montreal, Canada, Aug 2002.
 - [49] J. P. Sousa, V. Poladian, D. Garlan, B. R. Schmerl, and P. Steenkiste. Steps toward activity-oriented computing. In *IPDPS*, pages 1–5, 2008.
 - [50] Y. Tanaka. Proximity-based ad hoc federation among smart objects and its applications. *Proc. of CSTST’08 int’l*, 2008.
 - [51] Y. Tanaka. Proximity-based federation of smart objects: Liberating ubiquitous computing from stereotyped application scenarios. *Proc. of KES’10 Int’l Conference, LNCS*, 6276/2010:14–30, 2010.
 - [52] K. Ueda. Lmntal as a hierarchical logic programming language. *Theor. Comput. Sci.*, 410(46):4784–4800, 2009.

Bibliography

- [53] P. Verssimo, U. Lisboa, V. Cahill, A. Casimiro, U. Lisboa, K. Cheverst, A. Friday, J. Kaiser, and U. Ulm. Cortex: Towards supporting autonomous and cooperating sentient entities. In *In Proceedings of European Wireless 2002*, pages 595–601, 2002.
- [54] R. Want. Remembering mark weiser: Chief technologist, xerox parc. *Personal Communications, IEEE*, 7(1):8–10, Feb 2000.
- [55] M. Weiser. The computer for the 21st century. *Scientific American*, pages 94–104, september 1991.
- [56] Z. Zhong and T. He. Wireless sensor node localization by multisequence processing. *ACM Trans. Embed. Comput. Syst.*, 11(1):3:1–3:33, April 2012.