Faculty of Informatics
Masaryk University

Visual Coordination Networks

David Šafránek

Ph.D. Thesis

Supervisor: Luboš Brim
Brno, September 4, 2006

Abstract

Concerning software systems, there has been developed a huge scale of architectural formalisms, so-called Architectural Description Languages (ADL), which support formal specification and analysis of software architectures and architectural styles, e.g., Wright, UniCon, or Darwin. However, all these architectural languages lack features suitable for abstract description of other kinds of systems such as complex synchronous or asynchronous hardware circuits.

During our five years long experience of working in a team specialised on a software/hardware codesign and development of a high-speed hardware accelerated network monitoring hardware, we found it very encouraging to rise the notion of software architectural description to architectural description of a computer-based system of any kind. To this end, we decided to develop a framework of Visual Coordination Networks (VCN) which would make a step towards satisfaction of the above mentioned needs.

In this thesis, a visual formalism Visual Coordination Networks (VCN) for description and analysis of system architectures is developed. This formalism puts together ideas of exogenous coordination models and principles of architectural description and incorporates them in order to achieve an architectural description framework suitable for description and analysis of such a scale of systems for which the family of traditional architectural description languages is insufficient. Moreover, VCN is aimed to serve as a generic coordination model, which allows modelling of a variety of coordination primitives in a single language (from asynchronous Linda-like coordination to synchronous channel-based communication). The most significant properties we are taking into account are compositionality and hierarchy, which are important factors in component-based design.

Acknowledgements

At first, many thanks go to my supervisor Luboš Brim, especially for a lot of encouragement and a lot of suggestions during the work on this thesis.

A lot of thanks go also to Jean-Marie Jacquet, especially for giving me important suggestions in development of the behavioural model (Chapter 7) of the language introduced in the thesis, and also, for initiating of the research on the universal language for description of coordination models — the result of Chapter 6.

My thanks go also to Zdeněk Řehák, to members of Parallel and Distributed Systems Laboratory (ParaDiSe), and to members of Institute for Theoretical Computer Science (ITI).

Finally, I would like to thank Erika and my family for a lot of patience and support during writing of this thesis.

David Šafránek

Contents

Chapter 1

Introduction

Nowadays, the complexity of real systems, including the safety-critical systems such as aeroplane control software, medical devices, and various embedded systems (e.g., control components in cars), and on the other hand, very large web applications, the importance of having powerful tools for comfortable design and development of such complex systems is inevitable. The necessary part of any modern software engineering design methodology is modelling of systems at high level of abstraction. However, formal methods are still not satisfactorily used. The work undertaken in the thesis deals with improving this situation. The power of formal methods together with results of concurrency theory unified with the ideas of coordination are brought closer to a common system designer, who can use them practically.

There exist several techniques for analysis, modelling, and proving correctness of various aspects of systems. These techniques, so-called formal methods, have an exact mathematical base. In last decades, the main research topic on formal methods has been aimed to capture concurrent systems, which are more complex and require different and more sophisticated approaches (CSP [Hoa85], CCS [Mil89], Pi-calculus [Mil99]). It has appeared very important to deal with concurrency, because it is a natural property of the most of real systems. Real systems are composed of components that run in concurrent and interact with one another. With the increasing power of hardware systems, the notion of interaction plays more and more crucial role. The primary concern in the design of a concurrent application becomes its model of cooperation: how the various active entities comprising the application are to cooperate with each other.

1.1 Motivation

1.1.1 Coordination Models

The models of cooperation used in the most of present concurrent applications are essentially a set of ad hoc templates that have been found to be useful in practise. There is no paradigm wherein we can systematically talk about cooperation of active entities, and wherein we can compose cooperation scenarios such as (and as alternatives to) models like client-server, workers pool, etc., out of a set of primitives and structuring constructs. Consequently, developers must directly deal with the lower-level communication primitives that comprise the realisation of the cooperation model of a concurrent application.

To capture these problems, the notion of coordination models has appeared in the last decade, and it is a state-of-the-art approach to design and development of complex concurrent systems. Coordination can be defined as the study of topologies of interactions among components, and the construction of protocols to realise such topologies that ensure correctness. Moreover, coordination can be treated exogenously [Arb98], which means that interaction aspects are managed separately from component behaviour. Such property fits the principles of component-based design and allows compositional analysis of the system semantics.

During the last decade, a variety of languages for modelling various kinds of coordination have appeared. These are mostly Linda [CG89] (based on shared tuple space), Gamma [BM93] (based on multi-set rewriting), and the most recent exogenous coordination language REO [Arb04] developed at CWI. The state-of-the-art research question around this group of languages is currently to adapt both the formal methods for analysis and verification of concurrent systems to handle coordination.

1.1.2 Architectural Description

An important problem which any system developer encounters is the size and complexity of the system he or she develops. As the complexity of computer systems increases, correctness of the overall system structure — so-called *architecture* — comes to consideration as a crucial problem of system design.

A typical architecture of a system appears usually as an informal or semi-formal description realised in some kind of diagrammatic notation such as a flow-chart or a diagram of UML. Hence it seems very natural for system designers to use some visual notation in order to describe the critical architecture of a system. In other words, simple visual notations fit the requirements of a system designer to describe an architecture of the system he or she develops. However, the problem lies just in the informality or semi-formality of the used notation.

Although the recent version of UML, UML 2.0 [OMG03], offers a large scale of notations which comprise descriptions of systems of arbitrary kind, they still suffer from existence of a formal semantics which would be analysable in terms of formal methods.

Concerning software systems, there has been developed a huge scale of architectural formalisms, so-called Architectural Description Languages (ADL), which support formal specification and analysis of software architectures and architectural styles, e.g., Wright [AG97], UniCon [SDK$^+$95], or Darwin [MK96]. However, all these architectural languages lack features suitable for abstract description of other kinds of systems such as complex synchronous or asynchronous hardware circuits or rail interlocking mechanisms.

During our five-year long experience of working in a team specialised on a software/hardware codesign and development of a high-speed hardware accelerated network monitoring hardware [SRV$^+$06, HKR$^+$04], and our collaboration with members of a team specialized on formalization of rail control systems [BCJ$^+$04], we found it very encouraging to lift the framework of software architectural description to capture different kinds of computer-based systems than the traditional component-based software. Especially, we focused on embedded systems which employ principles of concurrency and reactivity. To this end, we decided to develop a framework of Visual Coordination Networks (VCN) which would make a step towards satisfaction of the above-mentioned intention.

1.2 Thesis Objectives

In this thesis, a visual formalism Visual Coordination Networks (VCN) for description and analysis of system architectures is developed. This formalism puts together ideas of exogenous coordination models and principles of architectural description and incorporates them in order to achieve an architectural description framework suitable for description and analysis of such a scale of systems for which the family of traditional architectural description languages is insufficient. Moreover, VCN is aimed to serve as a generic coordination model, which allows modelling a variety of coordination primitives in a single language (from asynchronous Linda-like coordination to synchronous channel-based communication). The most significant properties we are taking into account are compositionality and hierarchy, which are important factors in component-based design.

1.3 Thesis Contribution

Research related to architectural description and coordination languages has been vital during the last decade. On the one hand, various architectural description languages (ADLs, [AB05, RC03, AG97, Cle96]) have appeared in order to enable formal high-level description of component-based topologies of software systems. Moreover, also the incessantly semi-formal Unified Modelling Language (UML,[OMG03]) has turned to its second version incorporating a lot of features of ADLs. On the other hand, the research on coordination has aimed to studying of implementation and formal-based modelling of correct component interaction.

The most recently introduced coordination language Reo [Arb04] employs some features of ADLs, i.e., connectors and connector types, in order to make the development of a coordination glue more flexible. In consequence, joining of the results of both communities appears significant to turn towards achieving of correctness-by-construction property in design of computer-based systems of any kinds. Moreover, what appears to be very promising with respect to the success of UML, is supporting of such a process of design by easily comprehensible visual notations which can rapidly simplify the process of high-level system design.

However, visual notations which appear useful for high-level description of component-based system topologies are still too far from being satisfactorily used for formal system design and its consequent analysis. To our best knowledge, there is no precisely defined formal-based visual notation (visual formalism) which would satisfy the needs of formal architectural description while sufficiently employing the correctness-by-construction property. All the previous approaches to develop such an architecture-based visual formalism [RC03, CDS00, Saf02] lack some of the required features.

The contribution of this thesis is development of a visual formalism for architectural description which satisfies the following properties:

- a simple but yet sufficiently expressive graphical notation
- precise formal mapping of graphical notation to well-defined mathematical structures
- embodying of the correctness-by-construction property in terms of automatised compatibility checking of architecture parts and interoperability checking of entire architectures
- enhancement of the correctness-by-construction property by allowing abstract specification and reuse of exogenous coordination models

1.4 Thesis Structure

The thesis is organised as follows.

Chapter 2 gives a background overview of coordination and architectural description research topics emphasising the properties relevant for establishing of a visual formalism for architectural description. In consequence, the approach developed in this thesis is briefly introduced and compared with the most significant related work.

Chapter 3 gives a brief step-by-step introduction to Visual Coordination Networks on an example of a system architecture description. The chapter also contains a roadmap to simplify reading of the thesis.

Chapter 4 introduces informally all the features of the VCN graphical notation and explains semantics of VCN structures.

Chapter 5 gives formal mathematical representation of the VCN graphical notation and discusses correctness of the included definitions. The entire static part of the VCN language is exhaustively described in this chapter.

Chapter 6 defines four families of bus specification languages for specification and reuse of connector types — so-called bus classes. Consequently, an algorithm for construction of particular connectors from such specifications is given and model-theoretic and expressiveness issues of bus classes are studied.

Chapter 7 defines behavioural model of VCN architectures in terms of structural operational semantics. The issues of compositionality of the semantics are subsequently analysed and expressiveness of the behavioural model is discussed.

Chapter 8 develops a framework for architectural interoperability checking of VCN architectures based on weak bisimulation equivalence of architecture parts.

Chapter 9 gives a report about current state of the implementation support for VCN.

Chapter 10 gives a case-study of a railway automatic signalling system.

Chapter 11 concludes the achieved results and states directions of future research.

Chapter 2

Background and Preliminaries

In this chapter, we firstly summarise important properties of computer-driven systems which are relevant for high-level system modelling and, in consequence, we give a brief overview of fundamental methods for formal high-level description of systems (Section 2.1). In Section 2.2 we emphasise some key properties of architectural description languages, and in Section 2.3, we touch on coordination languages in order to single out some characteristics of coordination which are useful for high-level modelling of system architectures. In Section 2.4 we focus on existing methods which can be related with high-level (semi)formal visual description of component-based systems.

Section 2.5 introduces our approach for architectural description. Subsequently, the comparison of our approach with related approaches is given in Section 2.6.

Finally, in Section 2.7, the formal notions employed throughout the thesis are declared.

2.1 Characteristics of System Design

Present modern computer-driven systems can be divided into the following two distinct classes:

interactive systems - The basic property of systems from this class is constant interaction with the environment, in which they run. Additionally, this systems can be viewed as the leaders of the interaction. Whenever it can, the system listens to its environment, which calls for its services. The system delivers the services as soon as they are available. The typical interactive systems are operating systems, distributed databases, distributed algorithms, networking, etc. The main critical properties are deadlock avoidance, fairness, and coherence of distributed information.

reactive systems - This systems continuously react to stimuli coming from their environment by emitting back other stimuli. In contrary to interactive systems, reactive systems are purely input-driven and they must react in time dictated by the environment. In this systems, the environment is the leader of the interaction. Common members of this class are industrial process control systems, air-plane or automobile control systems, embedded systems, audio and video protocols, hardware circuits, man-machine interfaces drivers, etc. The key critical properties are safety and timeliness.

Of course, every complex computer-driven system never falls entirely into any of these two classes. Nevertheless, it is always useful to identify which parts of the system are interactive and reactive, and to handle them with appropriate formal methods. The key property shared by both interactive and reactive systems is *concurrency*. At first, these systems act concurrently with their environment. At second, their internal parts are usually also concurrent and communicate with each other. For example the client and the server in the distributed database system or the time-keeper, stopwatch, and alarm in a digital watch. Such a distinction of internal parts of a system leads to the idea of *component-based design*, where the individual parts are called *components* and the entire system is constructed modularly from such components.

The way of communication (a particular communication protocol, also called a *coordination model*) between components inside the system can be of both kinds — endogenous or exogenous. Endogenous communication is fully realised inside the components as a part of their internal computation. E.g., clients of a database server have to implement a protocol which ensures mutual exclusive access to a particular piece of data. In contrary, exogenous communication stands apart of the internal component computation providing that there are some special components in the system which control mutual communication of all the other components. E.g., clients of a database server have only to call some communication infrastructure service which ensures mutual exclusiveness of their access to the data on the server. With respect to these two kinds of communication protocols, two kinds of coordination models are distinguished — *endogenous* and *exogenous coordination model* [Cia96]. In modern component-based design, the exogenous model of coordination plays a very significant role. To emphasise the specificity of components which realise communication protocols, these special components are called *connectors* [AG97].

2.1.1 Modelling Techniques

There exist several techniques for modelling and analysis of all the kinds of systems mentioned above. These techniques, so-called *formal methods*, have a precisely formal mathematical base. The most important theory, which underlies formal methods, is the theory of formal semantics [Win93]. For proving the correctness of a system, its mathematical model must be given. Such a mathematical model is defined by the formal semantics of the system. In seventies, C.A.R. Hoare presented the approach called axiomatic semantics, for proving correctness of classical "transformational" systems (systems which compute the output values for given initial input values and stop). In the last two decades, the main research topic on formal methods has been aimed to capture reactive systems, which are more complex and require different and more sophisticated approach. Formal analysis of such systems is based on their operational, denotational, and axiomatic semantics. In this thesis we deal with the operational approach of formal semantics, as this approach is close to the intuitive comprehension of interactive and reactive system computation and especially enables, in case of finite models, fully automatised analysis of system correctness (in terms of model checking [EMCP99] and equivalence checking [CS01a]).

In last decades, several languages for both reactive and interactive systems were founded. In the field of interactive systems, languages CSP (Communicating Sequential Processes [Hoa85]) and CCS (Calculus of Communicating Systems [Mil89]), based on the process algebraic approach, have appeared. Their formal semantics and the theory behind them relate them naturally with formal verification methods. On one hand, these languages lead to more expressive extensions in terms of synchrony (SCCS [Mil83], Meije [BRS93]), broadcasting (CBS [Pra91]),BSP [Geh84]) , and mobility (π-calculus, [Mil99]). On the other hand, the concurrent programming languages for interactive systems Ada and Occam were build on principles of CCS and CSP. This languages have also appeared to be useful for specification of communication protocols, to this end, the language LOTOS [vEVD89] was developed. In LOTOS, the specification features of CCS are combined with some features of CSP. For reactive systems, SCCS and Meije calculi became the base for Esterel [Ber98] language, around which a large group of so-called *synchronous languages* appeared.

2.1.2 Compositional Hierarchy

CCS and CSP implement a very useful aspect of system design, in particular, the feature of compositional *hierarchy*. By applying this principle, processes can be defined as compositions of other processes. As this principle can be applied recursively, a deep hierarchy of processes can be constructed. Moreover, the compositionality property ensures that the semantics of each process is inferred modularly from semantics of its subprocesses. This feature allows to determine behavioural equivalence of two processes with the same compositional structure from equivalence of the respective subprocesses, taken component-wisely. Additional feature of hierarchical process definition provides abstraction of subprocess actions, so-called hiding. This way, observation of behaviour of a particular process behaviour can be abstracted from internal actions of its subprocesses.

The feature of hierarchy conforms to the methodology of component-based design [HC01], and hence allows this methodology to be applied in design using this kind of specification languages.

2.1.3 Atomicity

Two different notions of atomicity are concerned in formal modelling of behaviour of interactive and reactive systems — *interaction atomicity* and *execution atomicity*. Interaction is considered atomic if its effect on participating components cannot be altered through interference with another interaction [Sif05]. An example of atomic interaction is synchronous handshake communication in CCS as no two communications can occur simultaneously. In contrary, a model of broadcasting communication expressed in CCS by a sequence of actions is by its nature an instance of non-atomic interaction.

The execution atomicity is characterised by the requirement that no two component computation actions can overlap in time of execution. An example of atomic execution can be found in CCS, as well. In particular, all individual non-communication actions of a CCS process occur atomically. In contrary, the execution models of synchronous languages or SCCS are non-atomic [Gra99], as a group of more than one events can be processed at a single computation step.

Atomicity is an important property which must be taken into account when searching for a specification language suitable for modelling of the system behaviour at the required level of abstraction. I.e., it appears useful in high-level architectural modelling to employ high abstraction of the real system behaviour in order to avoid over-complication of the model and rather focus on architectural aspects only.

2.1.4 Synchrony vs. Asynchrony

Similarly as in the case of atomicity, two kinds of synchrony characterise particular kinds of reactive and interactive systems — *interaction synchrony* and *execution synchrony*. Interaction synchrony deals with the way of how the system interacts with its environment while the execution synchrony describes how the system components execute with each other.

Synchronous interaction, also called *strict synchronisation* [Sif05], assumes the system to synchronise with the environment in order to perform an action. An example of synchronous interaction is the operational model of CCS. An action in CCS executes just if the environment executes the complementary action. Note that this model of interaction can introduce a deadlock in systems of interacting deadlock free components. This happens just if a component is forced to perform a particular action but the environment can never execute the respective complementary action. In contrary, in asynchronous interaction, also called *non-strict synchronisation*, execution of outputs does not require synchronisation with inputs. This is a typical property of synchronous languages.

Synchronous execution considers that a system executes in synchrony with its environment. It assumes the so-called *synchrony hypothesis* [BG92], which requires the system to be infinitely faster than its environment. This model of executions fits the reactive systems such as hardware circuits in which signal levels change non-atomically and synchronously with the clock. This kind of execution implies existence of a global execution step of a system.

In contrary, *asynchronous execution* does not adopt any notion of global execution step. Each component of a system executes independently (in concurrency) of other components. The typical example of such models are interactive systems. Note that the interleaving semantics of CCS and CSP implements just this kind of execution.

2.2 Architectural Description

Architectural description is in general a formal high-level representation of a system topology, in particular, a static configuration of component and connectors of the system. For encoding of architectural descriptions there exists a family of Architectural descriptions languages (ADLs) which contains various formalisms suitable for formal architectural description of a large scale of software systems. As there is still a little consensus in the research community on what an ADL exactly is and what aspects should be modelled by an ADL, these ADLs are divers in various aspects (see [Cle96] for a survey).

In general, each ADL provides some feature of consistency checking of architectures. Typically, it can be checked whether all the pairs of components and connectors in the architecture are *architecturally compatible* in the sense that their mutual interaction never introduces a deadlock under the assumption that all the participating components and connectors are deadlock-free. Such a property is crucial especially when a designer composes an architecture by reusing already predefined components and connectors. As he treats such components as *black boxes*, the compatibility checking is a useful method which can ensure such a composition to be correct without designer's knowledge of the component behaviour.

Note that there can appear cyclic relationships among components and connectors. Such relationships are natural. In particular, consider for instance a kind of a client-server architecture sketched in Figure 2.1. There are two different servers, one client, and three connectors composed in the system. The client is connected to both servers and also both servers are mutually connected (by the connector CON_3). $SERVER_1$ has a role of a client with respect to the $SERVER_2$. If the particular behaviour of all the connectors and components satisfied the mutual compatibility check, then, considering the entire system, a deadlock situation could still arise [BCD02]. Thus another checking mechanism is required to ensure deadlock freedom of architectural description. A mechanism which considers not only the pairwise compatibility, but also the cyclic interoperability, is called *architectural interoperability checking*.

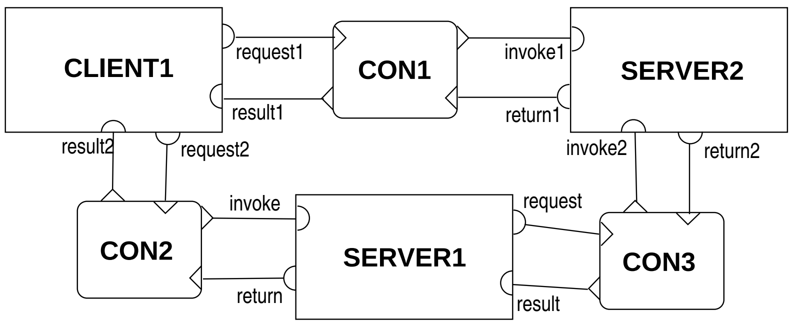


Figure 2.1: An architecture with cyclic relationships

From the topological perspective of architectures we distinguish two kinds of architecture checking methods — *horizontal* and *vertical checking* [Pla05].

Vertical checking concerns hierarchical embedding of one architecture into another architecture. The main purpose of this checking method is to ensure that the hierarchical embedding of a deadlock free sub-architecture into the component of the superior architecture introduces no deadlock. In other words, by this checking an inter-level interoperability property is considered — it is analysed whether the sub-architecture interoperates correctly with the interface of the superior component.

Horizontal checking deals with mutual compatibility of connectors and components at a particular level of architecture hierarchy. This checking method ensures that mutual interaction of any deadlock free component with any deadlock free connector introduces no dead lock. In other words, this checking method considers an inner-level interoperability property — it analyses whether each component in the architecture interoperates correctly with each connector that is connected to it. Moreover, as we have mentioned above, cyclic relationships should be also considered by the horizontal checking method.

In this thesis, we use the term *architectural interoperability checking* as a general notion which comprises all the architectural interoperability aspects mentioned above.

In the following subsections, we take two significant representatives of software ADLs and analyse their particular features and properties which are important for construction of a visual formalism for architectural description of a wider class of systems than is considered by the original aim of ADLs.

2.2.1 Wright

Owing to its closeness to the notion of exogenous coordination and the provided static kind of architectural description, we consider the language Wright [AG97], which is defined in terms of CSP, as a characteristic representative of an ADL established on a formal base. Moreover, Wright strictly distinguishes between components and connectors, and thus is a typical representative of ADLs.

In Wright, an architecture is represented by a so-called configuration. Each configuration is determined by a number of *components* and *connectors*. Component is defined by a *computation* encoded in CSP, and a set of *ports*. Each port has attached an interaction protocol expressed in CSP. Connector is defined by a *glue* and a set of *roles*. Glue represents the coordination model of a connector and is expressed in CSP. Similarly to ports, each role has attached an interaction protocol. After the component and connector declaration, the configuration contains instance declaration which instantiates the component and connectors declared above. Thus each connector and component can be replicated in the configuration. The last structure contained in a configuration is the set of *attachments*. Each attachment is a one-to-one link connecting a particular port of some component instance to a role of some connector instance. An example of a simple Wright specification is demonstrated in Figure 2.2.

Additionally, it is also worth noting that a configuration can be considered as a component computation. In this way, hierarchical design is allowed. Binding of the sub configuration events to the ports of the superior component is realised in terms of CSP relabelling.

In consequence of encoding an architecture description as a Wright configuration, its analysis can be performed. For this purpose, Wright introduces a checking methodology which has to be performed in order to decide that the architecture (hierarchical Wright configuration) is correct. In general, a configuration is said to be correct, if checking of all the following properties is successful (the respective tests are realised by the notion of CSP process refinement):

Configuration ABC
  Component A-type
    Port Out = $\overline{\text{a}} \rightarrow \text{Out} \sqcap \S$
    Computation = $\overline{\text{Out.a}} \rightarrow \textbf{Computation} \sqcap \S$
  Component B-type
    Port In = $\text{c} \rightarrow \text{In} \; \Box \; \S$
    Computation = $\text{In.c} \rightarrow \overline{\text{b}} \rightarrow \textbf{Computation} \; \Box \; \S$

  Connector C-type
    Role Origin = $\overline{\text{a}} \rightarrow \text{Origin} \sqcap \S$
    Role Target = $\text{c} \rightarrow \text{Target} \; \Box \; \S$
    Glue = $\text{Origin.a} \rightarrow \overline{\text{Target.c}} \rightarrow \textbf{Glue} \; \Box \; \S$

 Instances
  A:A-type
  B:B-type
  C:C-type

 Attachments
  A.Out **as** C.Origin
  B.In **as** C.Target
End ABC.

Figure 2.2: An architecture specified in Wright

1. Each port is consistent with computation of the respective component.
2. Each port is compatible with a role attached to it (if any).
3. Each connector glue is deadlock-free.
4. Each component computation is deadlock-free.

As it can be deduced from the list above, Wright offers both vertical and horizontal interoperability checking methods. However, there is no support for checking of cyclic relationships in Wright configurations. In spite of this fact, there are some other tests which can be performed in Wright. In particular, the initiator tests and the tests which concern the conformance of an architecture with respect to its *architectural style*. Architectural styles are defined as parametrised configurations which allow generalisation of architectures. For instance, the number of components, connectors, roles, and ports can be parameterised. As not all parameter values might be allowable, to disallow the unexpected architectures to be generated from the style, so-called *parameter constraints* can be provided as a part of the style description.

2.2.2 PADL

A representative of another kind of ADL is Process Algebraic Architectural Description Language (PADL) [AB05]. In PADL, there is no distinction between components and connectors. An architectural description in PADL consists only of components interconnected by links. Another difference from Wright is in ports. In PADL, a port represents a single event (so-called interaction, as in terms of PADL events are called interactions, but here we avoid this term to not confuse the vocabulary). Hence, there is no counterpart for the notion of port (resp. role) protocol. PADL specifications are formalised in terms of a CCS-like process algebra. The previous example of a Wright specification can be encoded as a PADL architecture by the following specification:

```
ARCHI_TYPE
 ARCHI_ELEM_TYPES

  ELEM_TYPE A-type

   BEHAVIOR
    A = out.A + stop

  ELEM_TYPE B-type

   BEHAVIOR
    B = in.b.B + stop

  ELEM_TYPE C-type

   BEHAVIOR
    C = a.c.C + stop

 ARCHI_TOPOLOGY

  ARCHI_ELEM_INSTANCES
   A : A-type
   B : B-type
   C : C-type

 ARCHI_ATTACHMENTS
  FROM A.out TO C.a
  FROM B.in TO C.c

END
```

The feature of hierarchical embedding is more expressive in PADL than in Wright. In particular, subcomponents can be attached to a port of the superior component not only in one-to-one fashion, but also in many-to-one fashion. There are two kinds of semantics of many-to-one connection. At first, subcomponent events can be marked `OR`. In situation when two or more subcomponents are capable of performing an event marked as `OR` which is attached to a superior component port, one of them is chosen to perform that event provided that the choice is nondeterministic. At second, subcomponent events can be marked `AND`. If there arises a situation when two or more subcomponents are set to perform `AND`-marked event attached to the superior component, then all these events will participate in synchronisation with the superior component event. Thus, the meaning of `AND` connections is synchronous broadcasting.

Concerning the analysis of architectures, PADL allows compatibility checking of interactions among all the components in the architecture. Moreover, also interoperability of cyclic architectures is supported, which is a significant property of this language. Concerning the issues of parameterisation, similarly to Wright also in PADL the architectures can be parametrised. However, owing to the structure of PADL, there is no vertical checking method considered.

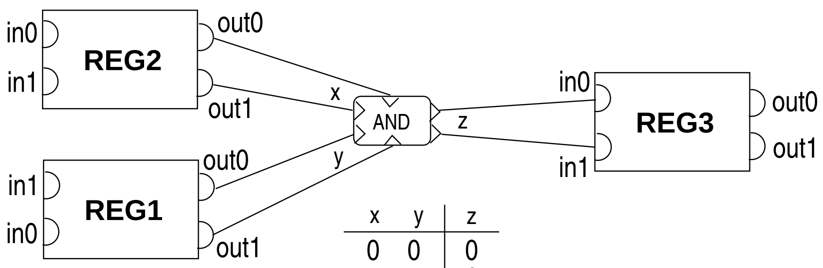
2.2.3 Properties of ADLs

To conclude the aspects of architectural languages presented above and discuss their usability for specific domains of interactive and reactive systems, we have to consider an example of an architecture of a simple hardware system depicted in Figure 2.3. In the scheme, there is a simple architecture of a circuit which in every tick of the clock takes the information stored in registers REG_1 and REG_2, performs the logical AND operation, and stores the result into the register REG_3, provided that the entire operation is realised atomically. Even if we abstracted from the synchronous aspects of the circuit and model the system in terms of asynchronous execution, we would found no primitive for atomic realisation of the AND connector in any of the ADLs described above.

Finally, it is worth noting that neither of the ADLs described above provides a visual notation, although visualisation of architectures seems to be very natural as the architectures are static and does not allow any kind of evolution (i.e., components cannot create other components during system computation). However, the schemes in the examples above shifts us towards the idea to define a visual notation for static architectures precisely.

2.3 Coordination Languages

Coordination languages constitute a family of programming and modelling languages which abstract away the details of computation, and focus on the invariant properties of systems. As such, coordination focuses



x	y	z
0	0	0
1	0	0
0	1	0
1	1	1

Figure 2.3: A specific system architecture inexpressible in a typical ADL

on system patterns that specifically deal with interaction. Coordination is relevant in design, development, debugging, maintenance, and reuse of all concurrent systems [Arb98]. Coordination models and languages are meant to close the conceptual gap between the cooperation model of an application and the lower-level communication model used in its implementation. The inability to deal with the cooperation model of a concurrent application in an explicit form contributes to the difficulty of developing working concurrent applications that contain large numbers of active entities with non-trivial protocols of cooperation. Thus it is required to treat cooperation explicitly — this model of coordination is called *exogenous*. The idea of exogenous coordination fits the principle of architectural description where connectors are treated independently of components.

2.3.1 Linda-like Languages

There is number of software platforms and libraries, so-called middleware, aimed to the development of concurrent applications, e.g., MPI or CORBA. Coordination languages can be comprehended as the linguistic counterpart of these platforms. One of the best known coordination languages is Linda [CG89], which is based on the notion of a shared tuple space. The tuple space of Linda is a centrally managed space which contains all pieces of information that processes want to communicate. Linda processes can be written in any language augmented with Linda primitives. There are only four primitives provided by Linda, each of which associatively operates on (e.g., reads or writes) a single tuple in the tuple space. There is a number of other models of coordination. Examples include various forms of parallel multi-set rewriting (Gamma, [BM93]) or a platform of a software bus (ToolBus, [BK98]). An exhaustive expressiveness comparison of Linda-like languages is given in [BJ03].

2.3.2 Reo

Concerning the above mentioned family of coordination languages in the context of high-level design and architectural modelling of a system, we find their underlying models too low-level for such a purpose. This drawback is rectified by another class of coordination models which are based on explicit support for coordination mechanisms. The most recent representative of such languages is Reo [Arb04] which is founded on the principle of channel-based coordination.

In Reo the exogenous coordination model is realised in terms of complex connectors, which are compositionally built out of simpler ones. The simplest connectors in Reo are a set of channels with well-defined behaviour supplied by users. A connector has a graphical representation, called a *Reo circuit*, which can be produced by applying certain composition operators to channels. In the static version of the Reo language, in which no dynamic evolution of component and channels is allowed, a Reo-circuit is just a finite graph where the nodes are labelled with pair-wise disjoint, non-empty sets of channel ends, and where the edges represent the respective connecting channels. An example of a Reo channel is depicted in the left-side of Figure 2.4. This channel is composed from two kinds of basic channels — a one-bounded FIFO channel (represented by a link with a box), and two synchronous handshake channels (represented by the common arrow). The entire connector composition behaves like a one-place buffer cell which synchronously outputs the stored data to its two output ends in terms of synchronous broadcast. Thus, this simple circuit combines the (bounded) asynchronous interaction with atomic synchronous interaction.

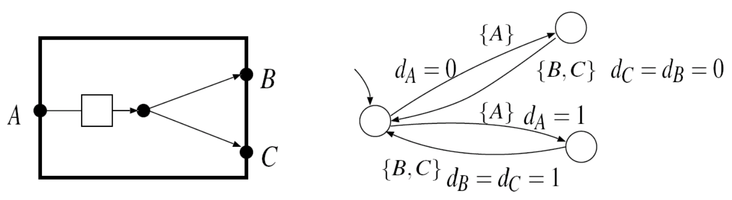


Figure 2.4: An example of a Reo connector and its semantics

Semantics of a Reo connector is defined as a *constraint automaton* [AR02] which is a traditional state-transition system extended in such a way that each transition is labelled with a set of atomically cooperating ports and a respective data constraint. The semantics of the above mentioned connector is depicted in the right-side of the Figure 2.4.

Reo offers a large scale of basic connectors which can be used for connector composition. As Reo is primary intended as a programming language, with the compositionality feature of channels Reo can be used also for description of high-level models of coordination. However, Reo does not directly support the feature of architectural compatibility and interoperability checking.

To emphasise the properties of Reo that are relevant for our purposes, note that the connector of Figure 2.3, which cannot be encoded in typical ADLs, can be be satisfactorily expressed as a Reo circuit. Moreover, all typical aspects of connectors described in traditional ADL approach can be also implemented in Reo channels due to the expressive power of constraint automata, i.e., synchronous and asynchronous ways of interaction.

2.4 Visual Notations

Commonly used visual methods for specification of various system aspects, nowadays unified into the Universal Modelling Language (UML [OMG03]), also include notations useful for specification of reactive and interactive system architectures. As the primary aspect of a system architecture is a very abstract specification of the system, visualisation of architectures rapidly simplifies understanding of the static system structure. Moreover, visual notations have the advantage of being simple to use by system designers. Thus, additional equipping of a particular visual notation with a formal semantics (this way a so-called *visual formalism* is established) brings formal methods closer to system designers. Therefore we believe that it is very useful to develop a visual formalism for architectural description.

Concerning development of a visual formalism, a difficult problem is to find a compromise between the richness of syntactic constructs and the comprehensible formal semantics of the chosen visual notation. I.e., it is worth noting that there is no formal semantics of full versions of UML notations.

In our thesis proposals we pioneered a survey of visual formalisms which can be applied to design of reactive and interactive systems. In general, we separated the visual formalisms into two groups — *state-based languages* (based on ideas of Harel's Statecharts [Har87] and their synchronous variants Argos [Mar91], SyncCharts [APF00], or asynchronous GCSR [DS97],...), and *data-flow-based languages* (build on ideas of Message Sequence Charts (e.g. UML sequence and communication diagrams). Both approaches emphasise different aspects of designed systems. For purposes of architectural design, the latter group is relevant. However, formalisms of the former group can be used at a lower-level of architectural design, in particular, for visual description of components and connectors. This fully agrees with Harel [Har87]:

> "...one has to assume some physical and functional description of the system, providing, say, a hierarchical decomposition into subsystems and the functions and activities they support..."

Additionally, It is worth noting that we did not include Petri nets [KB99] to that survey, owing to the fact that the principles of exogenous coordination and hierarchical design cannot be satisfactorily realised in Petri nets. The reason for that is true-concurrent semantics model of Petri nets based on the inherent notion of a token. However, concerning architectural description, Petri nets can be employed in the sense of the above mentioned application of state-based formalisms. This way, a very high expressiveness of components or connectors can be achieved, which might be useful especially for architectures of reactive systems.

As in this thesis we deal with a visual formalism for architectural description, here we only mention some data-flow-based formalisms of that survey. Data-flow-based visual formalisms concern description of communication relations among system components, in particular, they abstract from internal process behaviour. The semantics of a data-flow-based language is based on a particular coordination model. Coordination models are build upon a communication media and coordination laws. Examples of coordination media are channels, tuple spaces, buses, etc. Coordination model is given by a number of coordination laws which can describe both asynchronous or synchronous behaviours for communication of processes via coordination media.

Message flow graphs

Message flow graphs (MFGs) are visual notation for describing partial message-passing interaction between communicating concurrent processes. MFGs may represent different descriptions of communicating processes, e.g. concurrent programming language code, abstract specifications of communication services or protocols, or high level message flow diagrams like the well-known *Message Sequence Charts* (MSCc).

The basic idea of the MFG is that it is represented by a graph structure which is based on the concept of *send* and *receive* events represented as nodes. MFGs have two types of edges, *next-event* and *signal* edges, both represent explicit relations between nodes. Solid arrows represent next-edges and dashed arrows correspond to signal relation. All nodes in MFG, with the exception of the *start* and *finish* nodes, must be connected to just one other node. An example of an MFG is in figure 2.5. The formalisation of MFG is given in [Leu94]. To gain more power, MFGs can be equipped with conditions. MFGs, especially MSCs, are used in the description of telecommunication systems, in the analysis of parallel code, and in object-oriented system analysis and design methodologies. An example of the

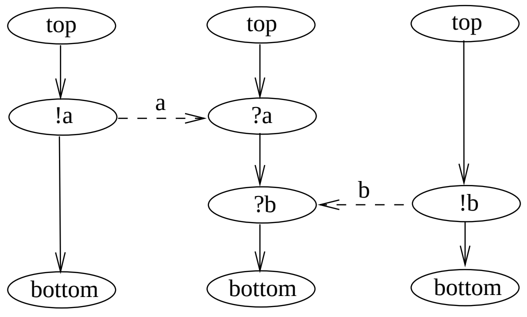


Figure 2.5: Example of an MFG diagram

latter are UML sequence diagrams.

The semantics of MFGs are traces of interleaved atomic communication events. The coordination model can be synchronous or asynchronous. Finite state operational semantics is required for analysis of MFG using formal methods. Finiteness of the global state space can be achieved with the assumption requiring the number of messages being sent at an instant of time to be finite.

The key property of MFGs is that they describe the behaviour of the system partially. In particular, each MFG models a particular view of a system interactive behaviour. In order to describe an entire system architecture we need a number of MFGs. Therefore this formalism is not suitable for architectural description.

Graphical calculus of communicating systems

Another approach, *Graphical calculus of communicating systems* (GCCS), aimed to visual specification of coordination aspects of interactive systems with abstraction from internal behaviour of components, has been proposed in [CDS00]. It preceded the formalism Visual Coordination Diagrams (VCN) which we develop in this thesis. As we originally elaborated on GCCS, in particular, we formalised and implemented the semantics of GCCS in [Saf01]. As VCN refines the principles of GCCS we mentioned its basic properties. adapts the robust process algebraic approach of CCS for the purpose of architectural description. GCCS is equipped with the binary synchronous handshake-style coordination model.

The basic principle of GCCS is hierarchical and compositional specification of coordination of components, which have their behaviour defined in terms of the process-algebraic operational semantics which is a parallel-composition-free sub-language of CCS. Comparing with Statecharts and other state-based languages which support concurrency and hierarchy, the coordination is in GCCS strictly semantically separated from the behaviour of components.

In GCCS, a component is represented as a *box* with *ports*, which have the meaning of the input/output *interface*. Coordination relations among components are represented as so-called *buses*. The set of boxes interconnected with buses makes a *network*. Networks can be embedded into other networks as components which makes the hierarchical structure. An example of a network is in figure 2.6.

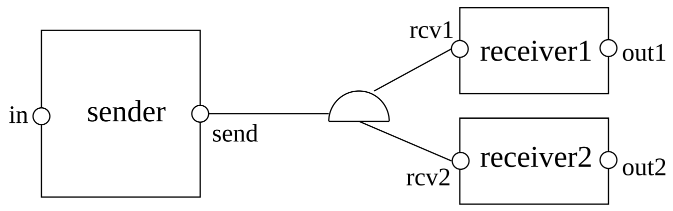


Figure 2.6: Example of a GCCS network

2.5 VCN Approach of Architectural Description

We have found out that the inability of GCCS to incorporate (at the level of connectors) any other coordination primitive than the binary handshake is very restrictive. In this sense, GCCS is less expressive than PADL. In PADL, there also a synchronous broadcast can be expressed together with a binary handshake. But it is still impossible to encode specification like the one from Figure 2.3, where the connector represents a more complicated coordination primitive. Therefore, we elaborated on GCCS and extended it to its synchronous variant SGCCS [Saf02] which enables such kind of connectors.

However, both SGCCS and GCCS in comparison with the ADLs suffer from the inability to parameterise the architectural description. This is mainly because of the visual notation — a number of ports appearing on a box representing the component, and also the number of links in an architecture, has to be fixed. To this end, we extend the language with a mechanism of so-called *bus classes*. A bus class can be treated as an abstract parameterisable specification of a particular coordination model, i.e., a family of connectors. We define a language for description of such bus classes and, subsequently, an algorithm for construction of particular buses, so-called *bus instances*, which satisfy such specifications. In this language, abstract connector types representing combinations of asynchronous and synchronous coordination can be specified. To realise semantics of bus instances, we introduce a notion of *cooperation machines*, which are state-transition systems in which each transition is labelled by a *cooperation*. Each cooperation represents a particular atomic synchronous interaction that can be any kind of multi-synchronisation of actions of participating components. The state space of the cooperation machine then represents asynchronous kinds of interaction, i.e., states represent a memory of a bus. The capacity of the memory is implicitly considered to be finite which implies that asynchronous coordination is implemented in terms of bounded buffers. This boundedness requirement is employed in order to allow automatised analysis of the architectures.

We also improve hierarchical aspects in such a way that a notion of a multi-purpose gate is introduced. Gates allow flexible encapsulation of architectures into components of higher-level architectures provided that both one-to-one and many-to-one kinds of relations are allowed. In terms of PADL, both `OR` and `AND` kinds of embedding can be employed in our extension. Moreover, a so-called *synchronous gate* is introduced. Meaning of a gate can be comprehended as a special kind of the AND of embedding. When n ports of some sub-architecture components is attached to a port of the superior component in terms of synchronous gate, then only n-to-one synchronisation might be performed. More precisely, each subcomponent must be enabled to perform the event attached by a synchronous gate in order to realise the expected synchronisation with the respective superior component's port.

Concerning the analysis features, we extend the architectural interoperability framework of PADL to checking of architectures extended with the above mentioned features. As some of those features violate modularity of architecture semantics, in establishing of this analysis feature the framework of PADL cannot be directly applied. Especially, the feature of synchronous gate requires a vertical checking methodology to be introduced in VCN.

Finally, we unify all the features mentioned above into a single visual formalism, which we call *Visual Coordination Networks*.

2.6 Related Work

In previous sections, various formalisms for description of architectures and coordination models have been briefly summarised. In this section we emphasise the crucial aspects of these formalisms in comparison with our approach. In particular, considering all of the compared languages, we discuss the following properties (in the respective order):

1. generality (parameterisation of specifications)
2. hierarchical embedding of architectures
3. expressiveness of components and connectors
4. architectural interoperability checking

2.6.1 Wright

1. As a textual specification formalism, Wright allows parameterisation of entire architectures in terms of architectural styles. In general, this is not possible in VCN, as VCN is exclusively visual formalism. However, VCN introduces a parameterised connector types — bus classes. Hence in VCN the developer has to instantiate such connector types in order to get particular connectors to the architecture.

2. Hierarchical design is allowed in both languages. However, VCN has more expressive primitive of gates available.

3. Wright has semantics model of both components and connectors defined in terms of CSP, whereas VCN component model is based on a CCS-like process algebra. Additionally, VCN connector model is based on a special kind of transition systems — cooperation machines. If we consider the bottom-most components in the architecture hierarchy then both languages have the same expressiveness at this level. However, when taking connectors into account, Wright is less expressive as is demonstrated by the architectural specification in Figure 2.3. Such a specification is realisable in VCN, but not in Wright.

   Conversely, CSP expressions specifying computation of Wright components are sufficiently represented by CCS-like expressions in VCN. However, there is no counterpart to port and role protocols. This fact is the main difference of the two languages. In particular, we leave for future work the extension of VCN to an interface-based view of specifications in terms of [dAH01b]. That way, such a disadvantage of VCN can be removed. The version of VCN defined in this thesis is already syntactically prepared for such an interface-based extension.

4. In both Wright and VCN, architectural compatibility checking of pairwise mutual interaction of components and connectors is implemented. However, Wright has no implicitly defined mechanism for interoperability checking of cyclic architectures.

2.6.2 PADL

1. Architectural descriptions in PADL may be parametrised in the same way like in Wright. Thus the same differences apply here.

2. Hierarchical embedding of architectures is realised in PADL by one-to-one or many-to-one attachments. The letter can be distinguished with respect to the behavioural model to AND and OR attachments.

All these kinds of embedding can be also realised in VCN by the notion of gate. Moreover, gates are even more expressive, as it has been mentioned in previous section.

3. Components in VCN have the same formal base as in PADL, therefore the languages are equivalent at this level. In contrary, PADL has no direct counterpart to the notion of connectors. Although the AND and the OR embedding can be sensed as kinds of inter-level connectors, their expressiveness is limited only to binary handshake or synchronous broadcast.

4. Concerning the architectural interoperability checking, it is supported by both languages. However, as in VCN the notion of hierarchy has more expressive features than in PADL, VCN comprises also a mechanism of vertical interoperability checking which is not defined in PADL.

2.6.3 AID

Here we discuss relations of VCN to another architectural language AID (Architectural Interaction Diagrams, [RC03]). We mention this language here because it is another extension of the GCCS language introduced above, and it has close relations with our work. In principle, an AID is a GCCS network extended with multi-purpose connectors.

1. Concerning the parameterisation of architectures, none of the two formalisms allows description of architectural styles like in PADL or Wright. However, both AID and VCN allow abstract definitions of connectors. In contrary to VCN, AID has no counterpart to the flexible bus class specification language. In AID, one has to use a second-ordered predicate logic to describe conditions that the respective bus transition systems have to satisfy. Moreover, the feature of link ranking makes the bus class specification language of VCN more expressive (for details see Chapter 6).

2. The feature of hierarchical embedding of architectures is also present in AID, but comprises only one-to-one attachments. Hence in this aspect is VCN more expressive.

3. In contrary to VCN, the formal semantics of AID components is defined directly in terms of labelled transition systems. Semantics of connectors is also defined as a state-transition system. Such a transition system differs from the VCN cooperation machine in the format of transition labels (for details see Section 7.3 in Chapter 7). In general, this difference leads to a greater expressive power of AID in this

aspect, as a synchronous lossy coordination model can be expressed in AID, but not in VCN.

4. There is no methodology of architectural interoperability checking employed in AID.

2.6.4 Other Component-based Models

ADLs are primary aimed to high-level description of component-based systems, and together with formal verification methods they support, ADLs seem to be ideal tools for analysis of software architectures and for employing the design-by-correctness property in software design. However, just because of their high-level granularity and the distance between an abstract model and the executable code, they are not being widely commercially used in software design. The low-level granularity component models such as COM/DCOM [Mic] or CORBA [OMG] are being much more frequently used for direct software design, although they do not support such large scale of formal methods as ADLs. We believe that the high-level granularity architecture description formalisms necessarily require a direct support of some visual notation which would bring the abstract power of the high-level formalisms closer to system designers. In our approach to high-level specification of component-based systems we try to tackle just this goal.

In the gap between the two different levels of component models stated above there lie concepts which can be used, to a specific extent, for both design and implementation of component-based systems. Very promising is the exogenous coordination language Reo [Arb04] developed at CWI and already described in the previous section, and also the experimental component model SOFA/DCUP [PBJ98] developed at Charles University Prague. In this subsection we compare our approach with both of these significant representatives of complex models suitable for component-based design of concurrent systems.

Reo

In spite of its different (lower-level) purpose, we have mentioned Reo [Arb04] especially because of the kind of abstract connectors it introduces. In comparison to VCN, we just focus on discussion of the notion of connectors in both languages. The main difference between VCN and Reo connectors is in the way of their specification. In Reo, one has to compose complicated connectors from basic connectors. Generality of such specifications is then achieved by abstracting from particular sets of input and output ends. In contrary, in VCN one has to give an overall specification of the connector type — the bus class. There is no notion of bus class composition in VCN. Therefore, these two approaches of connector specification can be comprehended as dual. However, Reo allows specification of synchronous lossy channels which is not possible in VCN.

SOFA

SOFA/DCUP [PBJ98, Pla05] is a component model for design and deployment of concurrent software systems which offers abstraction at the level very similar to ADLs. Such abstraction allows the top-down design in SOFA to start from abstract architectures with component interfaces interconnected with connector roles leading to executable concurrent programs deployed on different platforms where the complex multi-platform coordination glue is generated from connector specifications [Bal02]. Connectors in SOFA can be implicit — remote procedure call, event delivery mechanism, and data stream communication, or explicit — connectors created by architectural composition of other connectors and components. The abstract level of SOFA concerns specification of component and connector frames (black-box view). Component frames are composed from interfaces, whereas connector frames contain roles. Roles and interfaces have attached behavioural protocols specified as regular expressions. Each component or connector can be hierarchically refined with a sub-architecture.

Architectural interoperability checking in SOFA consider both horizontal and vertical checking and is focused on component composition correctness. This checking methods include searching for specific kind of software component communication errors which cannot be caught by traditional CSP or CCS approach [AP05].

In spite of the different purpose, it is worth comparing SOFA with our approach, especially as possible requests for future extensions of both models can arise from their mutual inspiration. By its nature, VCN is strictly binded to its visual notation and therefore is a high-level design language. At this level the abstract way of connector specification in terms of bus classes allows high-level modelling and reuse of coordination models. This can be comprehended as a more abstract level with respect to the SOFA level of abstraction. A VCN connector can be taken as an abstract model of a complex SOFA connector. In other words, a SOFA connector can capture implementation aspects of a given coordination model specified by a VCN bus. However, as VCN is primarily aimed to abstract description of reactive and interactive systems with high-level atomicity of interactions, in contrary to SOFA, the VCN interoperability checking framework has no support for checking of specific errors which arise from asynchronous component composition.

2.7 Formal Preliminaries

In this section, we recall some basic notions of concurrency theory which are used throughout the thesis.

First of all we present a definition of the fundamental notion of a general labelled transition system which provides the semantic domain for VCN architecture parts. In consequence, we recall the notion of bisimulation equivalence that is used in Chapter 8 to establish the framework for architectural interoperability checking of VCN architectures.

Definition 2.1 *A* labelled transition system (LTS) *is a quadruple* $\langle Q, \mathrm{Act}, T, q_0 \rangle$ *where*

- *Q is the set of states,*
- *Act is the alphabet of the system, satisfying $\tau \notin Act$;*
- *$T \subseteq Q \times (Act \cup \{\tau\}) \times Q$ is the transition relation where $\langle q, q' \rangle \in T$ is denoted $q \xrightarrow{a}_T q'$*
- *$q_0 \in Q$ is the initial state.*

A labelled transition system is called finite *if both its set of states and its transition relation are finite.*

Subsequently, we present the definition of strong bisimulation equivalence of two labelled transition systems.

Definition 2.2 *Let $S_1 = \langle Q_1, Act, T_1, q_{0_1} \rangle$ and $S_2 = \langle Q_2, Act, T_2, q_{0_2} \rangle$ labelled transition systems.*

- *A relation $R \subseteq Q_1 \times Q_2$ is a* strong bisimulation *if whenever $\langle q_1, q_2 \rangle \in R$ then for each $a \in Act$ both of the following holds:*
  1. *If $q_1 \xrightarrow{a}_{T_1} q_1'$ then $\exists q_2' \in Q_2.\, q_2 \xrightarrow{a}_{T_2} q_2'$ and $\langle q_1', q_2' \rangle \in R$.*
  2. *If $q_2 \xrightarrow{a}_{T_2} q_2'$ then $\exists q_1' \in Q_1.\, q_1 \xrightarrow{a}_{T_1} q_1'$ and $\langle q_1', q_2' \rangle \in R$.*
- *We say the states q_1 and q_2 are* strongly bisimulation equivalent *and write $q_1 \sim q_2$ iff there exists a strong bisimulation R such that $\langle q_1, q_2 \rangle \in R$.*
- *We say that the* labelled transition systems *S_1 and S_2 are* strongly bisimulation equivalent *and write $S_1 \sim S_2$ if and only if $q_{0_1} \sim q_{0_2}$.*

Another kind of behavioural equivalence which is employed in this thesis is weak bisimulation. Before we define the weak bisimulation equivalence itself, we fix some notation.

Notation 2.3 *For each $a \in Act$ denote $\hat{a}$ the following event:*

- $\hat{a} \overset{\text{df}}{=} \epsilon$, *if* $a = \tau$;
- $\hat{a} \overset{\text{df}}{=} a$, *otherwise.*

Denote $\overset{a}{\Rightarrow}$ *the following sequence of succeeding transitions:*

$$\overset{a}{\Rightarrow} \overset{\text{df}}{=} (\overset{\tau}{\to})^* \overset{a}{\to} (\overset{\tau}{\to})^*$$

Definition 2.4 *Let $S_1 = \langle Q_1, Act, T_1, q_{0_1} \rangle$ and $S_2 = \langle Q_2, Act, T_2, q_{0_2} \rangle$ labelled transition systems.*

- *A relation $R \subseteq Q_1 \times Q_2$ is a* (weak) bisimulation *if whenever $\langle q_1, q_2 \rangle \in R$ then for each $a \in Act$ both of the following holds:*

  1. *If $q_1 \overset{a}{\to}_{T_1} q_1'$ then $\exists q_2' \in Q_2 .\, q_2 \overset{\hat{a}}{\Rightarrow}_{T_2} q_2'$ and $\langle q_1', q_2' \rangle \in R$.*
  2. *If $q_2 \overset{a}{\to}_{T_2} q_2'$ then $\exists q_1' \in Q_1 .\, q_1 \overset{\hat{a}}{\Rightarrow}_{T_1} q_1'$ and $\langle q_1', q_2' \rangle \in R$.*

- *We say the states q_1 and q_2 are* (weakly) bisimulation equivalent *and write $q_1 \sim q_2$ iff there exists a (weak) bisimulation R such that $\langle q_1, q_2 \rangle \in R$.*

- *We say that the* labelled transition systems *S_1 and S_2 are* (weakly) bisimulation equivalent *and write $S_1 \approx S_2$ if and only if $q_{0_1} \approx q_{0_2}$.*

Chapter 3

Introduction to VCN

In this chapter, we introduce basic principles of VCN on a real example of an architecture of a part of a hardware design developed in the Liberouter project [AFN03]. In particular, we consider the design of a DRAM access scheduler which makes an important part of Liberouter four-port hardware-accelerated high-speed router [CES06].

3.1 Modelling a Shared Memory Access System

The overall scheme of the memory scheduler is depicted in Figure 3.1. Main components of the entire router design are a header-field extractor (HFE), which parses headers and data information from incoming packets, an edit engine (EE), which is responsible for modifying packet headers according to the current routing table, and a packet replicator (RP), which allows multicasting of packets. As the router is designed to incorporate four network interfaces, there are four header field extractors and four edit engines to capture the high-speed routing process for all interfaces. The memory scheduler has the purpose of storing parsed packet data and allowing quick packet replication and editing. The scheduler is implemented together with header-field extractors, edit engines, and the replicator in a Xilinx FPGA chip, hence the design has the character of a firmware loaded into the chip during the booting process.

The design is encoded in VHDL hardware description language [vEVD90]. In spite of the fact that VHDL has high-level features and employees component-based hierarchical design methodologies, our experiences show that VHDL programs are very difficult to read and understand, especially concerning wire interconnections. These experiences motivate us to adapt our architectural visual notation VCN for hardware design architecture description.

To demonstrate basic features and principles of VCN we show how an architecture of a particular part of the above-mentioned design can be vi-

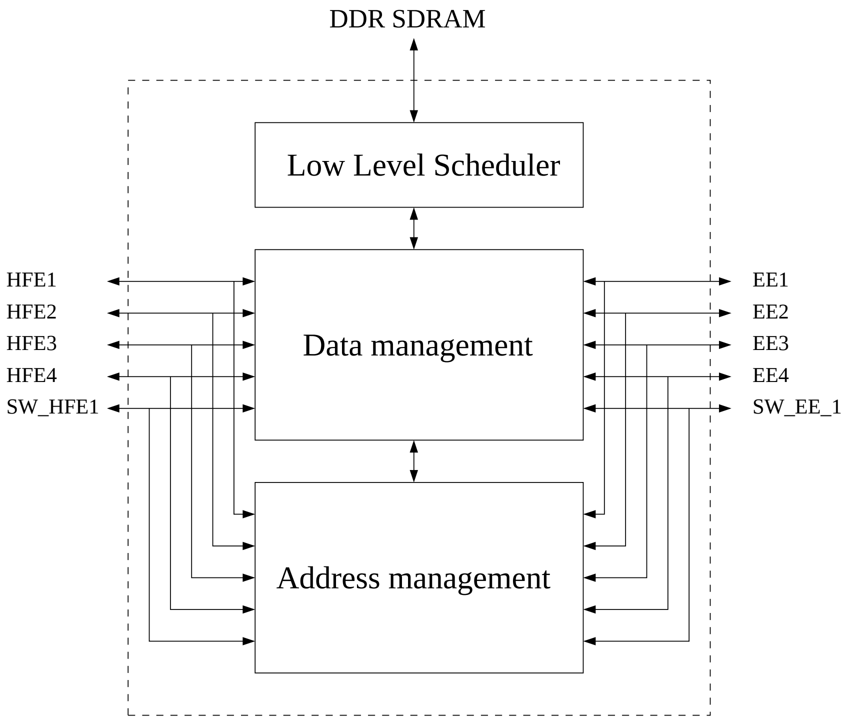


Figure 3.1: Scheme of Liberouter DRAM scheduler

sually formalised using the VCN notation. In particular, we consider a version of the memory scheduler design simplified to a single packet editor and we focus on the address management unit. Moreover, we abstract from the header-field extractor part of the design.

3.1.1 Top-Most Network of Components and Buses

The component of the memory scheduler serves address reference manipulation requests obtained from concurrently running edit engine and packet replicator components, and controls correct interaction of both components with the shared memory ensuring mutually exclusive access. The memory is organized in frames of a constant size provided that with each packet data there is stored a number of references. The replicator component increases this number in order to set the replication factor. On the contrary, the edit engine unit decreases the number of references in order to unset the packet replication factor. The entire address management unit is responsible for ensuring mutual exclusion of both operations.

To create the VCN architecture of the memory scheduler component, we will follow guidelines of the top-down design methodology. First of all, we identify the top-most components of the system:

- PR . . . a component which controls multicasting of packets to different virtual network flows captured by network interfaces. In our model we abstract from the internal replicator behaviour and consider only the behaviour involved in the address management process, in particular, the address reference number increase feature.
- EE . . . a component which performs packet header editing in order to control the routing task. Similarly as in the case of the replicator, we abstract from the edit engine behaviour that is not relevant for the address management. More precisely, we focus on the address reference number decrease feature.
- $Addr\_Mng$ . . . a component which models DRAM address administration and realizes mutual access of reference number decrease and increase operations.

Firstly we focus on each component separately and determine its input and output ports. In other words, we establish component interfaces. In

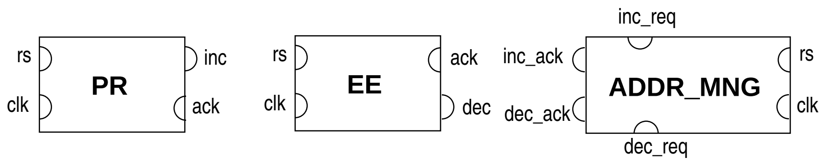


Figure 3.2: Interfaces of top-most components

Figure 3.2 there are interfaces of all the three components visualised using the VCN notation. Each interface is represented as a box with the name of the component written inside. Input and output ports are depicted as semicircles. The meaning of each port is the following:

- PR
  - inputs
    - rs . . . has the purpose of receiving events of a global asynchronous reset signal. The component reacts to the reception of the reset signal by returning to the initial state.
    - clk . . . receives events of clock signal rising edges. As a typical synchronous hardware component, PR behaviour is synchronous with the clock signal.
    - ack . . . accepts the acknowledge signal denoting successful completion of the last reference number increase operation.

  - outputs
    - inc ... emits requests for increase of a reference number of a particular DRAM frame. As we consider the non-value passing version of VCN, in the behaviour of the PR component it is abstracted from transmission of a particular address value.

- EE
  - inputs
    - rs, clk ... have the same meaning as in the case of the PR component.
    - ack ... receives the acknowledge signal denoting successful completion of the last reference decrease operation.
  - outputs
    - dec ... emits requests for decrease of a reference number of a particular DRAM frame.

- $Addr\_Mng$
  - inputs
    - rs, clk ... have the same purpose as in previous components.
    - in_req ... receives requests for increase of reference numbers.
    - dec_req ... accepts reference number decrease requests.
  - outputs
    - inc_ack ... emits the signal notifying successful completion of the last reference increase operation.
    - dec_ack ... emits the signal notifying successful completion of the last reference decrease operation.

At this point, we have identified all interfaces of top-most components. As all three top-most components cooperate with each other, what remains to be done is to specify their mutual interconnection. In particular, we have to include the three components in a VCN *network* describing the respective top-most architecture. Thus, we add a so-called *bus* with *links* that realize interconnection of components. The resulting VCN network is depicted in Figure 3.3. As the interaction of top-most components is assumed to be synchronous, the bus included in the network represents a binary handshake coordination model with cooperations listed in Table 3.1. Each line of the table shows a particular atomic interaction of components, which is given by a particular set of input ports and a set of output ports involved in the interaction. As in our case all interactions are binary, the respective sets are singletons in all the listed cooperations. The dot notation in the third and in the fourth line is used for unambiguous identification of the inc port which is included in both the PR and the EE component interface. Note that input and output ends of the bus $1to1\_HSK$ are marked by numbers. These numbers represent a so-called ranking which is related with general specification of buses (see Section 3.3).

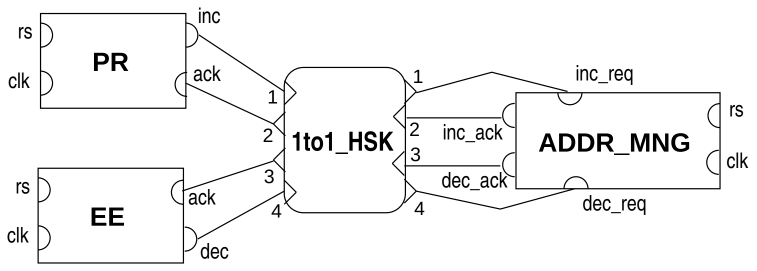


Figure 3.3: A VCN network of the entire address management system

Cooperations of the $1to1\_HSK$ bus
{inc}/{inc_req}
{dec}/{dec_req}
{inc_ack}/{PR.ack}
{dec_ack}/{EE.ack}

Table 3.1: Interaction of the top-most network components

3.1.2 Inter-level vs. Inner-level Cooperation Specification

A key property of VCN is the possibility of creating hierarchy of networks. To demonstrate this feature in our example, we refine the specification of the $Addr\_Mng$ component by a sub-network describing architecture of the address management unit. The refinement has to satisfy the requirement that all components which appear in the lower-level network must form a proper decomposition of the superior component. Especially, the decomposition must employ all ports that appear in the superior component interface. Moreover, these ports are not permitted to be involved in any bus-link connection (ports which satisfy this requirement are called *free ports*). This requirement conveys the higher-level character of these ports.

A Lower-level Network of the Address Management Unit Architecture

Before we formalize the address management unit architecture, we informally explain behaviour of the unit. The purpose of the address management unit is to manipulate incoming memory address reference increase and decrease requirements, to implement respective operations, and to acknowledge completion of the last performed operation. Moreover, as the packet replicator and the edit engine components execute concurrently, another important task of the address management unit is to ensure mutual exclusion of both operations.

To achieve the above-mentioned tasks modularly, the $Addr\_Mng$ component is decomposed into following four lower-level components (their respective interfaces are depicted in Figure 3.4):

- Req_Regs is a component modelling request registers. This component stores an incoming request (received on inc_req and dec_req ports) for increase or decrease operation provided that both an increase request and a decrease request can be captured simultaneously in one tick of the clock. In some future tick, a request currently kept in the respective register can be read by other components (on output ports inc and dec). Both registers can be cleared by the reset signal.

- $Addr\_Admin$ is a component which realizes memory reference increase and decrease operations.

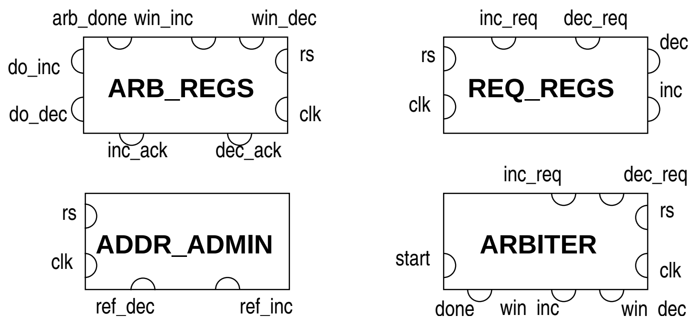


Figure 3.4: Interfaces of lower-level components

- $Arbiter$ is a component responsible for controlling the mutual exclusive shared memory access of both decrease and increase operations. The component checks for a request stored in request registers (by inc_req and dec_req ports). If a request is encountered, the arbitration process is started. The necessary precondition of arbitration is the requirement that there is no previous arbitration winning request waiting for realization. This requirement is checked by the $start$ input port. The process of arbitration itself searches for the highest priority request. The winning request is then emitted on the win_inc or win_dec port.

- Arb_Regs is a component modelling arbitration registers. It includes a register for storing a winning request for both operations and also a register notifying the last successfully completed request. Winning requests are received on win_inc and win_dec ports, notification of successful arbitration is received on the arb_done port. The particular winning request is sent for realization by the do_inc or do_dec port. The component includes a control logic which ensures that each request is sent for realization if and only if the respective arbitration process has been successfully completed. Finally, notification of successful submission of a request for realization is signaled on inc_ack or dec_ack port.

Interconnection of the four components is depicted in Figure 3.5. The $1to1\_HSK1$ and $1to1\_HSK2$ buses employ a binary handshake coordination model similarly as the top-most network bus. Cooperations defined by both buses are listed in Table 3.2.

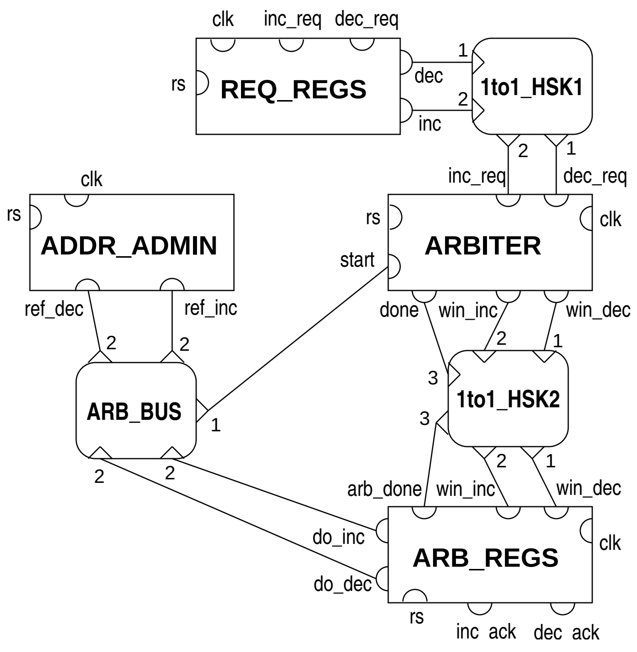


Figure 3.5: A VCN network representing the address management unit

The Arb_Bus, which interconnects $Arbiter$, $Addr\_Admin$, and Arb_Regs components, employs a more complicated coordination model, i.e., interactions of these three components are asynchronous. In particular, the bus is

Cooperations of $1to1\_HSK1$ bus	**Cooperations of $1to1\_HSK2$ bus**
{inc}/{inc_req}	{win_inc}/{win_inc}
{dec}/{dec_req}	{win_dec}/{win_dec}
	{done}/{arb_done}

Table 3.2: Interaction of the lower-level network components

equipped with a memory in that it stores a request submitted for realization by the arbitration process (by a signal received from the *do_dec* or *do_inc* port of the *Arb_Regs* component). When the address administration unit (the component *Addr_Admin*) processes a request, the request is removed from the bus internal memory and signaled to the *ref_dec* or *ref_inc* port of the *Addr_Admin* component. Finally, the bus is also responsible for notification of the fact that no request is currently waiting for realization in order to coordinate initiation of a next arbitration process. The notification is signaled to the *start* port of the *Arbiter* component. Entire behaviour of the bus is specified by the state transition diagram depicted in Figure 3.6. Each transition of the diagram denotes a particular cooperation. To simplify the notation of cooperations in the diagram, we typically exclude braces denoting sets of input and output ports, as in the case of this diagram. The empty set is denoted by a dash.

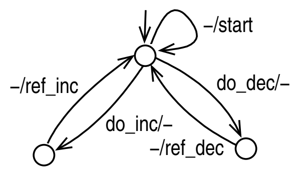


Figure 3.6: Specification of *Arb_Bus*

Embedding of the Lower-level Network

At this point, we have decomposed the *Addr_Mng* component to a lower-level architecture. What remains to be done is to bind the respective lower-level network to the interface of the higher-level *Addr_Mng* component. In particular, inter-level cooperation relationships have to be specified. This is realized by connecting each of higher-level interface ports to a particular free port of the lower-level network. There are several kinds of such an inter-level port connection, i.e., one-to-one and many-to-one connections are allowed. For a particular lower-level network, the group of all respective connections is called a *gate*.

In our particular example, two kinds of inter-level connections, so-called *gate mappings*, are employed. The inc_req and the dec_req ports are connected to homonymous ports of the Req_Regs component by one-to-one mappings. Similarly, the inc_ack and the dec_ack ports are connected to homonymous ports of the Arb_Regs component. More complicated is the mapping of the rs and clk ports. In particular, the rs port is connected to all rs ports appearing in lower-level component interfaces by a many-to-one mapping. Interconnection of the clk ports is treated similarly by another many-to-one gate mapping. The many-to-one kind of mapping requires a more precise specification concerning its semantics. Especially, a many-to-one mapping can have basically two different interpretations — non-deterministic choice of just one of connection ways or synchronous processing of just all connection ways (for details see Chapter 4). In our example, both many-to-one mappings have the latter interpretation. The reason for that lies in the fact that both the reset and the clock signals are required to be atomically transmitted in an indivisible time instant to all components of the decomposition.

In the VCN visual notation, inter-level connections are specified by dashed lines. Gates employed in our example are depicted in Figure 3.7. Synchronous interpretation of a many-to-one gate mapping is specified by the '$\times$' symbol inside the respective higher-level port semicircle.

3.1.3 Computation Layer

Up to this point, we have tackled modelling of interconnection aspects of the system architecture. The respective part of the VCN language is called a *coordination layer*. As it is demonstrated on our example, the coordination layer is hierarchical. The entire coordination layer hierarchy is closed by component computation specification — the so-called *computation layer*. More specifically, instead of further refinement of a component by another lower-level network, an undecomposable *leaf* representing computation aspects of the respective component can be embedded in the component interface. VCN accepts the model of component computation defined in terms of a state transition system. Such a model can be considered either as an abstract (interface-level) specification of component computation or as an implementation of a component compiled into a state transition system.

Considering our example, we demonstrate aspects of the computation layer by making each of the lower-level network components a leaf of the entire model. To show how a particular leaf can be defined, a state transition system of the Req_Regs component is depicted in Figure 3.8. Each transition label represents an event of reception (denoted '?') or transmission (denoted '!') of a signal on the respective port.

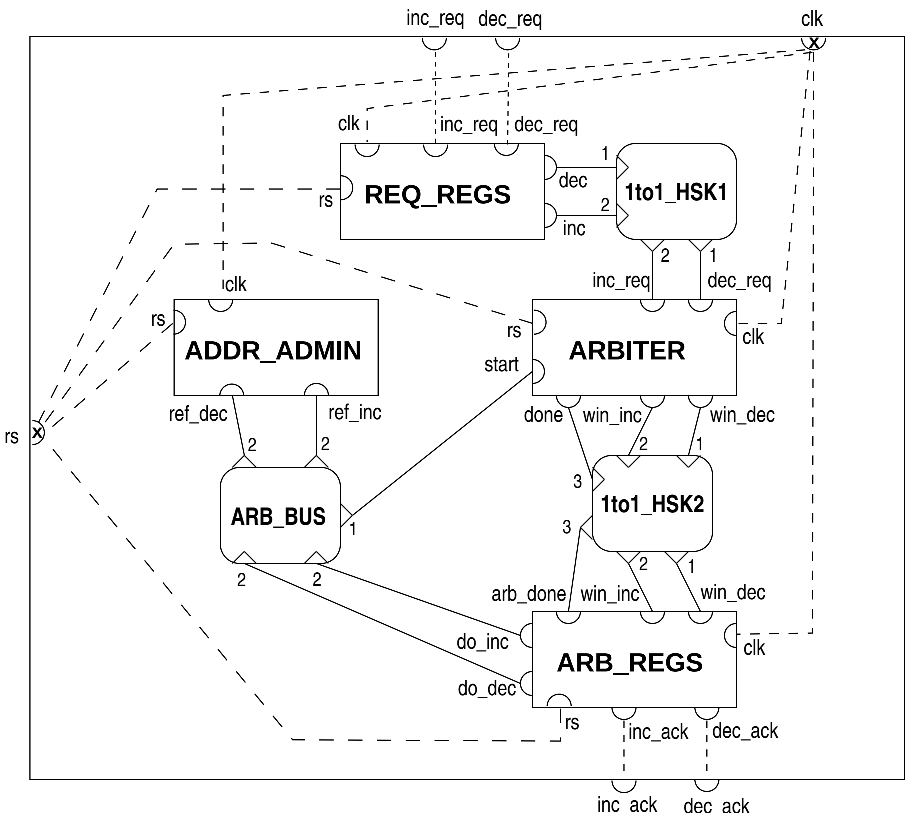


Figure 3.7: Gate mappings of higher-level ports to lower-level free ports

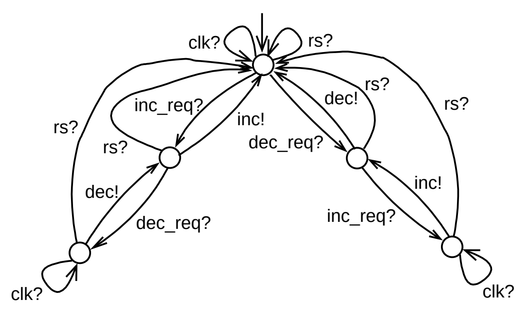


Figure 3.8: Computation layer of the bottom-most component *Req_Regs*

3.2 Architectural Interoperability Checking

A completely defined VCN architectural specification (a hierarchy of networks closed by leaves) that satisfies all syntactic requirements imposed by the VCN language can be further analyzed by the so-called architectural interoperability checking framework. In particular, it can be automatically checked if all horizontal (bus-component) and vertical (subnetwork embedding) connections are correct with respect to behavioural aspects of the entire architecture.

A typical property which is checked in system architectures is deadlock freedom. An architecture is said to be *interoperably correct* with respect to the deadlock freedom property if and only if all the following conditions are satisfied:

1. All components are deadlock free.
2. All buses are deadlock free.
3. Interaction of all components and buses does not introduce any deadlock at any level of hierarchy.
4. Every hierarchical embedding of a sub-architecture does not introduce any deadlock.

The general interoperability checking framework incorporated in the VCN language allows not only the deadlock freedom property to be checked, i.e., the interoperability check can be set for a large variety of behavioural properties (see Chapter 8 for details).

3.3 Generic Specification of Coordination Models

As it can be seen in the example of the memory access system architecture, particular coordination models employed in architectural modelling are typically repeatedly reused. In our example, three buses employ the binary handshake coordination model. To allow generic specification of such frequently reused coordination models, VCN includes a framework for parametrized description of buses – the framework of so-called *bus classes*. Bus classes represent templates for construction of particular buses. For specification of bus classes, VCN introduces the *bus class specification language*.

In the example of the memory access system presented above, there are two kinds of coordination models employed – the binary handshake (all $1to1\_HSK$ buses) and a one-place buffer with emptiness signalization (Arb_Bus). At first we show how a basic bus class for the former model can be specified in the bus class specification language. In consequence, we demonstrate more complex features of bus classes on the latter model.

A bus class specification consists of a list of so-called *cooperation patterns*. A cooperation pattern is a generic template for particular kind of cooperations making the respective coordination model. Each cooperation pattern can be optionally equipped with a so-called rank constraint. The purpose of rank constraints is mutual relation of individual input and output ports connected to buses (instances of a particular bus class). Assuming that each link of a particular bus is identified with a natural number – a so-called *rank*, rank constraints realize intended relations of bus input and output ends.

The binary handshake coordination model can be generally defined by the following bus class:

$$\begin{array}{l} \mathcal{B}(\texttt{In}, \texttt{Out}, \texttt{rank}) := \{ \\ \quad \texttt{In}, \texttt{Out}, \texttt{rank} \neq \emptyset \\ \quad \texttt{i/o} \wedge \texttt{rank(i)} = \texttt{rank(o)} \\ \} \end{array}$$

The $\texttt{In}, \texttt{Out}$ parameters of the bus class denote sets of input and output ports that can be connected to particular buses, respectively. The `rank` parameter represents ranking of particular bus links. The first line of the specification is a parameter constraint that requires all these parameters to have non-empty values. The second line of the specification is a cooperation pattern that generates a binary synchronization cooperation for each pair of a bus input and a bus output port which satisfies the rank constraint, i.e., links which connect both ports of the pair to the bus must have the same rank.

If we look into our example, we observe that all handshaking buses are equipped with link ranking. Thus, with respect to the particular ranking numbers, the bus class specified above generates exactly cooperations listed in Table 3.1 and Table 3.2 for the respective buses.

The Arb_Bus models a more intricated coordination model. Especially, this bus is equipped with a bounded memory (i.e., a one-place buffer). Although there is only one bus of this kind in the considered system architecture, we demonstrate how it can be generalized for arbitrary number of registered signals. The resulting bus class specification has the following form:

$$\begin{array}{l} \mathcal{B}(\texttt{In}, \texttt{Out}, \texttt{rank}, \texttt{capacity}) := \{ \\ \quad \texttt{In}, \texttt{Out}, \texttt{rank} \neq \emptyset \wedge \texttt{capacity} = 1 \\ \quad \texttt{i}\!\uparrow\!\texttt{reg/}- \wedge \texttt{rank(i)} = 2 \\ \quad -\texttt{/o}\!\downarrow\!\texttt{reg} \wedge \texttt{rank(o)} = 2 \\ \quad -\texttt{/?reg,o} \wedge \texttt{rank(o)} = 1 \\ \} \end{array}$$

Comparing this bus class with the previous specification, on the first view it can be observed that there is one more parameter and that cooperation patterns have a more complicated structure. The presence of the `capacity` parameter notifies the fact that the bus class is equipped with a buffer. A particular value of this parameter then sets the number of buffer cells. Cooperation patterns of this bus class contain so-called *memory handles* that are responsible for manipulating the referred buffer (i.e., storing or removing of a value and testing for buffer emptiness). In our particular case, there is considered a one buffer referred by the label reg. As the capacity of the bus memory is limited to just one, the buffer has just one cell. The first pattern defines the storing operation provided that whenever the buffer reg is not full and a signal is sent to the bus from a port ranked by 2, the received signal can be stored into the buffer reg. The second pattern specifies destructive transmission of a signal stored in the buffer reg provided that if the buffer is not empty then the signal can be removed from the buffer and emitted to some port ranked by 2. Finally, the third pattern tests for emptiness of the buffer reg provided that whenever the buffer reg is empty, a signal can be emitted to some port ranked by 1.

With respect to the link ranking of the Arb_Bus included in our example, by instantiating the bus class specified above we achieve the same bus behaviour as the one depicted in Figure 3.6. Thus, we have demonstrated the general power of bus class specification. For more details on the bus specification language we refer the reader to Chapter 6.

3.4 Thesis Roadmap

In this section we give a roadmap of the thesis to simplify reading of the thesis with respect to this introductory chapter.

Explanation of all basic specification features and principles of VCN is given in Chapter 4. Especially, all types of many-to-one gates are presented there (Section 4.3.3).

Precise definition of VCN syntax is given in Chapter 5. All requirements stating how the structure of a well-defined (syntactically consistent) VCN architecture looks like (Definition 5.27) are presented formally in that chapter.

Architectural interoperability checking framework is developed in Chapter 8. The problem of horizontal interoperability is introduced in Section 8.4.2 and the respective checking methodology is stated and proved in Theorem 8.26. The problem of vertical interoperability is explained in Section 8.4.1 and the respective checking methodology is given at the end of the chapter. The interoperability checking methodology employs the behavioural semantics of VCN architectures that is defined in Chapter 7.

Details of bus class specification are given in Chapter 6 where the bus class specification language is defined formally. An algorithm for construction of instances of bus classes is also presented in that chapter.

Chapter 4

VCN: Principles and Features

In this chapter, all principles and features of VCN are introduced informally.

4.1 Structure of VCN

Being a visual formalism, VCN introduces a graphical notation. More precisely, two dimensional geometric space is used to represent VCN syntactic structures. In contrast to linear, i.e. textual, notation, two dimensional space allows expressing of complex information more efficiently. In other words, suitably defined graphical notation can be in this particular sense more succinct than textual syntax and can be easily captured by the user. However, without any precise mathematical definition giving unambiguous meaning to the graphical notation, graphical notation is inexact.

Following that fact, we firstly present the diagrammatic notation of VCN, and then we define its precise unambiguous representation in the form of algebraic terms. It is a common approach, which is, for example, known from the work on formalisation of Statecharts [Har87]. The graphical notation of VCN networks is based on the notation which has been firstly introduced in [CDS00] and further refined in [Saf02]. We generalise that notation and extend it to satisfy nontrivial requirements of architectural specification. We focus on universality of VCN, in particular, we define the VCN language to be abstract in the sense of its potential application in different domains of system design.

It is worth noting that the VCN graphical notation (and therefore its formal representation) represents only the static topology of the concurrent system under design. Behavioural aspects of the system are supposed to be captured separately in some suitable visual or textual formalism, which is compatible with VCN semantics. In other words, the behavioural aspects of architectures are completely treated in competence of VCN behavioural model, whereas the structural aspects of architectures fall under the VCN visual syntax.

With respect to the separation of structural and behavioural aspects we distinguish two kinds of VCN terms:

- *structural terms* – representing the static *structure* (formal representation of diagrams themselves)
- *behavioural terms* – representing *behaviour* (i.e., the semantics of the component computational model, or the semantics of the coordination behaviour).

In this chapter, the basic principles of VCN language concerning both structural and behavioural aspects are introduced. The notion of structural terms is then precisely defined in Chapter 5. Chapter 6 and Chapter 7 are dedicated to behavioural semantics of VCN. The former of the two chapters is focused on behavioural model of connectors, while the latter contains precise definition of component and architecture composition behavioural terms.

4.2 Basic Principles

There are two main kinds of design methodologies that are usually used for specification and design of a component-based concurrent system. These are the bottom-up and the top-down methodology. To support these methodologies by a graphical notation, the feature of hierarchy is necessary. Therefore, we introduce hierarchical approach to specification of component-based systems in the VCN formalism.

4.2.1 Hierarchy of Networks and Leaves

At the most abstract level of view, the hierarchy of VCN introduces two separate layers — *computation layer* and *coordination layer*. The computation layer focuses on computational aspects of a system under design, while the coordination layer deals with interaction aspects. The principle of such a layered structure respects the nature of component-based system design, and is inspired by the work concerning software architecture description [AG97].

In our setting, the computation layer is treated as the low level layer of system specification, upon which the coordination layer is bottomed. Thus from the designers point of view, both the top-down and the bottom-up design methodologies can be applied during system design using the VCN. On one hand, VCN language allows the computation layer to be taken as a supplementary layer, which can be added to the system hierarchy later during the design (the top-down approach). On the other hand, the computation of components can be specified at first, while the coordination layer can be added later (the bottom-up approach).

Coordination layer

The main idea of the VCN coordination layer, which revises the concept introduced in Wright [AG97], is to describe interaction aspects of a static component-based architecture. The notion of such a static structure concerns topology of concurrently running *components* permanently coordinated by specific *connectors*. This topology is determined by the point-to-point *links* which connect component interface ports to connectors. The particular VCN construct that represents such topologies of components and connectors is called *network*. An example of a network is depicted in Figure 4.1.

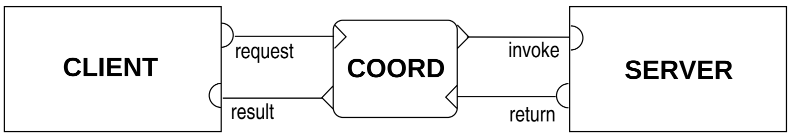


Figure 4.1: A network of two components coordinated by a connector (bus)

Similarly to Wright, connectors are treated as first-class citizens (at the same level as components). In the VCN setting, connectors are called *buses*. Buses represent coordination mechanisms which control component interaction.

Computation layer

Component computation is described using VCN *leaves*. VCN leaves are cornerstones of the computation layer. In our setting, leaves are assumed to be abstract computation models of system components. More particularly, a leaf is an atomic element in the VCN structure and can be specified in an arbitrary formalism for formal description of reactive computation. It is assumed that the kind of formalism which can be used for this purpose is compatible with the used semantic model of VCN. In general, the potential set of such compatible formalisms includes any reactive computation description language which can be encoded into a labelled transition system. Details of this semantic compatibility will be given in the next chapter, where the semantic models of VCN will be described.

Network hierarchy

The computation layer makes the bottom most level of the VCN hierarchy. It is determined by the set of all the leaves which are used in the particular system design. As it has been mentioned above, leaves directly represent computation of components and are interconnected by buses to form network topologies.

In a natural sense, such a network topology of leaves and buses is viewed as a black box with complex behaviour hidden inside (defined by computation of leaves coordinated by buses). More specifically, not only leaves represent computation, but also entire network topologies do so. This idea leads us to set the notion of a component to be more abstract than is the notion of a leaf. In particular, either a leaf or a network can be sensed as a component in the VCN style of thinking.

The possibility of taking a network as a component (instead of taking only a leaf) allows the coordination layer to have more levels of hierarchy. In consequence, entire network hierarchy of a particular top most network has a form of a tree with nodes representing its components at each level.

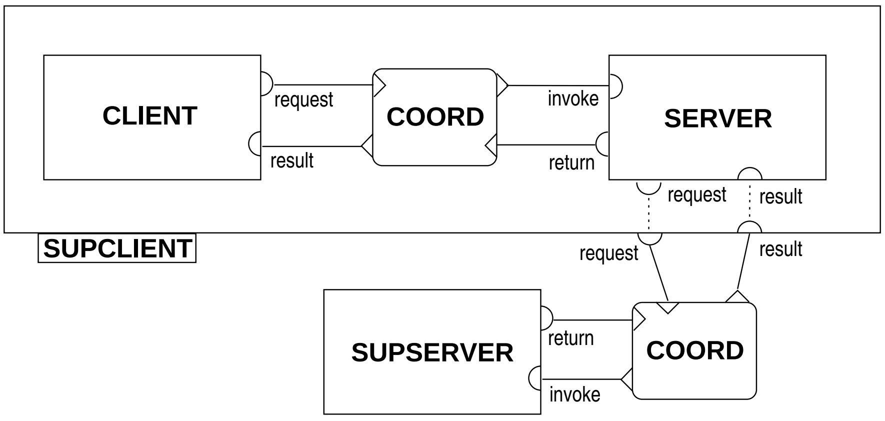


Figure 4.2: A network as a body of a higher level component

2 In the example in Figure 4.2 the network representing a hierarchical client/server architecture is depicted. The server component of the network of a plain client/server architecture from Figure 4.1 is extended to behave as both a server (for the subsidiary client) and a client (for communication with the higher level server). The entire network is considered as a client component of the higher level client/server architecture. The component bodies of the $Client$, $Server$, and $Supserver$ components represent leaves of the entire hierarchy. The tree structure of the whole system is depicted in Figure 4.3.

4.3 Elementary Entities

First of all, we introduce entities related with *components*, which are elementary structures in the scope of the VCN network. The purpose of components is description of computational aspects. Consequently, we introduce entities related to coordination aspects – *buses* and *links*.

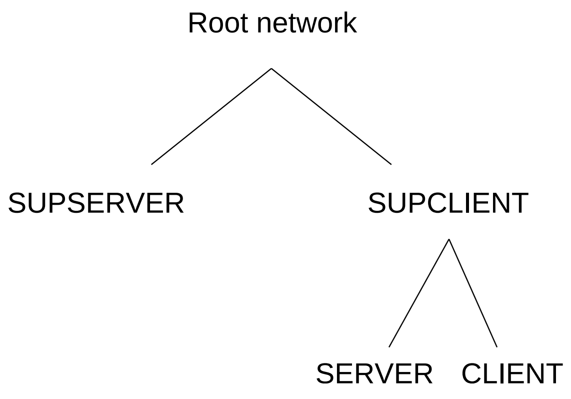


Figure 4.3: Example of VCN hierarchy

As it has been suggested before, the notion of VCN component is rather conceptual than constructive. More precisely, its meaning is gluing three constructive entities together to form a single logical entity. The relevant constructive component entities are *component body*, *component interface*, and *gate*. Intuition behind these entities is illustrated in Figure 4.4.

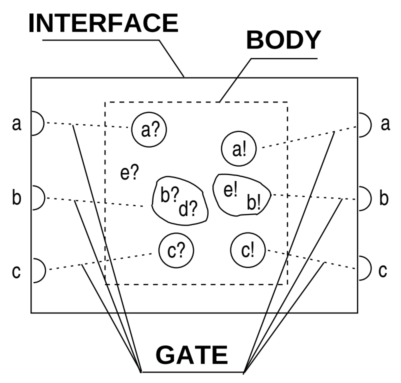


Figure 4.4: Scheme of component entities

Component body captures component computation. Component interface identifies the component as a black box with a specified set of *ports*. That way, component interface makes the border between the component internal computation and the environment. By the notion of environment we mean other components in the particular network topology which can interact with the component.

Gate makes the glue layer which binds a particular component body to a component interface. The main purpose of a gate is to allow construction of a component by embedding an arbitrary (already predefined) component body into a given interface.

4.3.1 Component Body

Component body makes the main functional element of a component. It allows embedding of computation behaviour to VCN structure. In more particular, component body represents the model of component reactive computation. By its nature, the component body is the entity which incorporates the component behavioural model to VCN networks.

From the structural point of view, every component body is characterised by the finite set of observable *events* which can occur during the component computation. With respect to the nature of reactive behaviour, two basic kinds of computation events are distinguished — input events (reception of a signal or data) and output events (transmission of a signal or data). To represent the internal non-reactive activities of components (i.e., computation unobservable by the environment), the special event denoted τ is introduced. An occurrence of any event during the computation is called an *action*. Thus in our setting, an event is sensed as abstraction of its occurrences (actions). Each observable event is identified by its name and type. Moreover, each action is identified by the respective event name and type.

With respect to VCN hierarchy, the basic kind of component body is a *leaf*. Leaf is sensed as an atomic element with respect to the hierarchy, and is directly characterised by the set of particular computation events. This static structural abstract characterisation is assumed to be further refined by a suitable semantic model of dynamic computational behaviour, e.g., the operational semantics presented in Chapter 7.

As it has been mentioned above, component body can be not only a leaf, but also a network. At the bottom most networks of the VCN hierarchy, bodies of all components have the form of leaves. However, at any higher level of the hierarchy, arbitrary network can form a component body. In that case, the component computation is inferred from the network topology by composition of individual elements computation. The resulting network computation is characterised by events in the same way like the leaf computation. All the coordinative behaviour of the network composition is implicitly assumed to be considered as the internal activity (occurrence of the τ event).

Graphical representation of a higher level component body is implicitly realised by the VCN network diagram. The representation of a leaf component body can be realised by explicit embedding of some suitable graphical formalism (e.g., a variant of Statecharts).

4.3.2 Component Interface

Component interface is graphically represented as a rectangular box. Its purpose is to determine the black box view of the component. In more particular, it is responsible for identification of all observable component events. Most notably, every occurrence of such an observable event (an action) is sensed as stimuli for interaction of a component with its environment. Such interaction can be of two forms:

- input action — accepting of some input (instant reception of an event from the environment)
- output action — producing of some output (instant emission of an event to the environment)

To capture the place where actions of some event occur, we introduce the notion of a *port*. Ports are supposed to be named locations on which input and output actions occurring during the component computation take place.

Respecting the two kinds of events, two types of ports are distinguished — *input* and *output ports*. Each port is graphically represented as a semicircle drawn on the border of an interface box. Orientation of the semicircle notifies naturally port type. The port name is placed outside the interface box near the semicircle. In the scope of a particular interface, names of ports of either kind are required to be unambiguous. More precisely, there can be two ports of the same name included in the interface only if they are of different type.

To illustrate the principle of component interface, in the left part of Figure 4.5 there is a component representing a client of a simple client/server system. Its component body is a leaf characterised by one output event $request!$ and one input event $result?$. The computation of this leaf is sketched by a simple statechart in the right part of the figure. In addition, the internal event $init$ occurs in the first step of the computation. This event is supposed to be internal event of the leaf. The respective ports for both observable events are depicted on the component interface box.

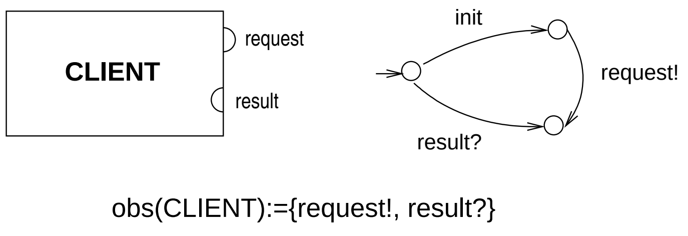


Figure 4.5: Client component with one input and one output port

In modelling of a more complex component, it can be useful to distinguish among its various *roles*. Typically, specific parts of a component are acting in different computational tasks. Such tasks are focused on particular component behaviour the basic properties of which can be determined independently of the properties of other tasks. For example, imagine a component representing a node of a hierarchical client/server system. This component can serve as a server for some subsidiary client, while to fulfil its server task it interacts as a client with some superior server. Such a component is depicted in Figure 4.6. The left ports of the interface are responsible for the server role of the component and the right ports are involved in its client role. In the right part of the figure there is a statechart illustrating the computation of the component. The two roles comprise independent properties, especially concerning the order of invocation of the respective events. The client role of the component imposes an invocation order to the $request!$ and the $result?$ events provided that both events are assumed to be invoked in just the specified order. Analogously, the server role defines the ordering of the $invoke?$ and the $return!$ events.

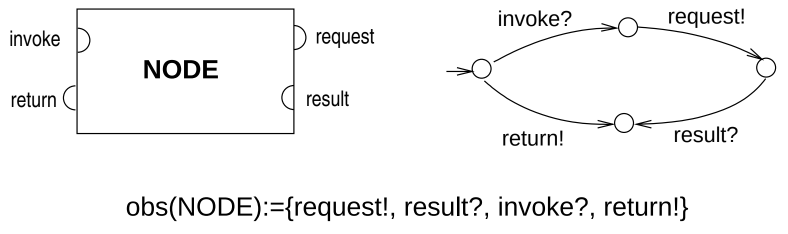


Figure 4.6: A component performing two different roles

The feature of organising ports into roles can be included in VCN component description by the following way. Each interface port can be optionally extended with a *role name*. Role name is assumed to be placed near the relevant port semicircle inside the interface box. The role name denotes the logical assignment of a port to a role. By convention, the following requirements must be satisfied. At first, a port can be assigned to at most one role. Secondly, if any port of a particular interface has assigned a role, then all the other ports of the respective role must have the role name defined.

In our example, the two roles of the $Node$ component are denoted *cl* and srv. Resulting interface with the relevant role names is depicted in Figure 4.7.

To complete the description of the component interface entity, we have to refer the reader to Section 7.2.6, where the expressiveness issues are discussed and compared with other work. Here we would like to highlight the fact that the VCN formalism is not exceedingly focused on the component interface description, but it is specifically aimed to description of coordination aspects of the system architecture. However, roles can be refined by sophisticated interface specification methods such as interface au-

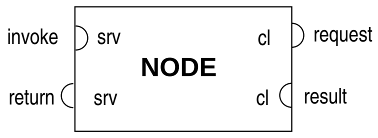


Figure 4.7: Interface ports extended with role names

tomata [dAH01a] or port protocols in Wright [AG97]. To keep our graphical notation clear, but yet sufficiently expressive and easily extensible, we consider the above mentioned principle of component interface specification. Although we do not introduce an interface model of VCN, we included the possibility of organising interface ports in terms of roles to the syntax of the language. In our future work we aim to establish the interface model of VCN in terms of refining component interfaces with state-transition semantics and enable mutual checking of the behavioural model defined in this thesis against such interface model.

In the last note on component interface entity, we would like to make a reference to other elements of the coordination layer which are closely related with interfaces, i.e., *links*. Links are responsible for component-to-bus connections and will be described later in this section. Here we only underscore the fact that each interface port can be equipped with at most one link in the scope of network. The presence of a link reserves the port to be used only for interaction with the particular bus. Conversely, if a particular port has no link assigned, the respective event which is related with such a port is observable by the environment of the entire network. This property of a port is an important concept of the network hierarchy. Therefore, we use the term *free port* for each port that embodies the property of having no link assigned.

4.3.3 Component Gate

If we consider a component body S characterised by the set of its observable events denoted $obs(S)$, the way of how these computation observables are related to ports of the component interface is justified by the notion of *gate*. In general, gate can be comprehended as a group of relations which relate the observable events of the component body to the interface ports. More precisely, gate is from the structural point of view constituted of a set of *gate mappings* of various kind. From the semantical point of view, gate determines a function, so-called *gate function*, which takes the set of component body (white box) observables and according to their mapping to interface ports it returns observable (black box) events. this way, the gate function transforms the white box view of a component to the black box view. The scheme illustrating the gate functionality is depicted in Figure 4.8.

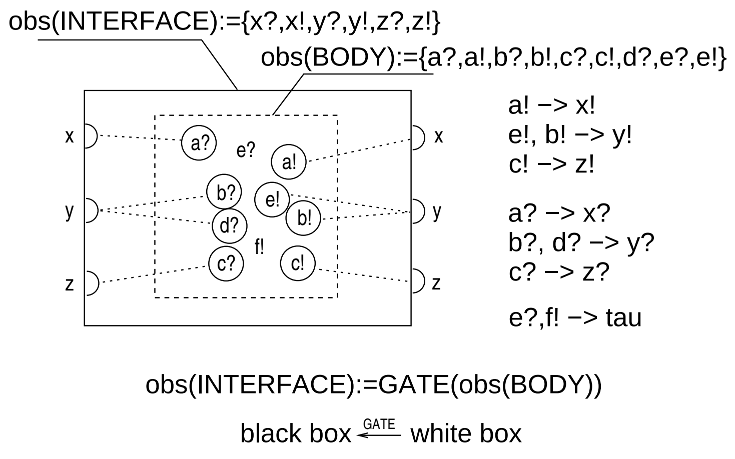

obs(INTERFACE):={x?,x!,y?,y!,z?,z!}
obs(BODY):={a?,a!,b?,b!,c?,c!,d?,e?,e!}
x
y
z
a?
e?
a!
b?
e!
d?
b!
f!
c?
c!
x
y
z
a! –> x!
e!, b! –> y!
c! –> z!
a? –> x?
b?, d? –> y?
c? –> z?
e?,f! –> tau
obs(INTERFACE):=GATE(obs(BODY))
black box ← GATE white box

Figure 4.8: Scheme of gate functionality

In general, the gate function is supposed to embody the following properties:

- many-to-one character — more than one event can be associated to a particular port, the semantic issues of this property are discussed below;
- event type preservation — input events can be mapped just only to input ports and output events just only to output ports;
- total function — each observable event for which no gate mapping is defined is supposed to be mapped to τ and reflected as an internal event from the black box viewpoint;
- surjective function — each interface port must be targeted by some gate mapping, this property ensures natural consistency of the interface by requiring all the interface ports to have defined their subtle meaning.

With respect to VCN hierarchy we have to distinguish two different cases of how the range of a gate function can be obtained. If the component body has form of a leaf then the gate is only a logical entity and has no graphical representation. Its meaning is direct one-to-one mapping of observable leaf events to the interface ports provided that each event is renamed to the name of the target port. Events which are not mapped to any port are hidden and observed as the τ event.

If the component body has form of a network, gate is the key entity which glues together two different levels of network hierarchy. In this case, it is worth recalling the fact that observable events of the component body are inferred from the composition of computation of individual network components. Since every coordinative behaviour is not observable by the network environment, all the network observable events can occur only on free ports. In other words, the set of network observables coincides with the set of all free ports of that network. Thus, gate has in this case a graphical representation which is realised by the set of dashed lines connecting free ports of the lower level network components to ports of the superior interface. A simple example of such a gate is demonstrated in Figure 4.2. There the gate makes the local server component events $request!$ and $return?$ to be observable on the interface of the TOPCLIENT component. Names of both events are preserved in this particular case. Each of the both dashed lines in the figure represents a particular gate mapping.

As it has been declared in the above mentioned list of gate properties, gate can contain many-to-one mappings in general. This property is meaningful in situations when the component body has the form of a network. The purpose of many-to-one gate mapping is to identify free ports of several components in the subsidiary network with one particular port of the superior interface. Such a many-to-one gate mapping has to be consistent in the sense that for each subsidiary component there can be at most one free port included in the gate domain. We impose this constraint to prevent over-complicating of VCN behavioural model. Additionally, with respect to the event type preserving property, another natural constraint of gate consistency is that all the ports of a particular gate mapping must be of the same kind (all either output or input).

To introduce other properties of gate mappings, we have to state beforehand some specific aspects of the VCN behavioural model. More particularly, three different types of many-to-one gate mappings are distinguished according to the intended behavioural semantics. The type of a particular many-to-one gate mapping is determined by the respective symbol inside the target port semicircle. With respect to the intended behavioural semantics, the possible mapping types can be the following:

- $+$ *asynchronous gate mapping* — the interface-level event is constituted of **just one** of the component body events involved,
- $\times$ *synchronous gate mapping* — the interface-level event is constituted just of **all** the component body events involved (occurring simultaneously, the interface-level event notifies their synchronisation);
- $\cup$ *universal gate mapping* — the interface-level event is constituted of **maximal nonempty subset** of the component body events involved (events in the particular subset are supposed to occur simultaneously, the interface-level event notifies their synchronisation – the maximality of the set ensures that there is no enabled event omitted in the synchronisation; see the example below for details).

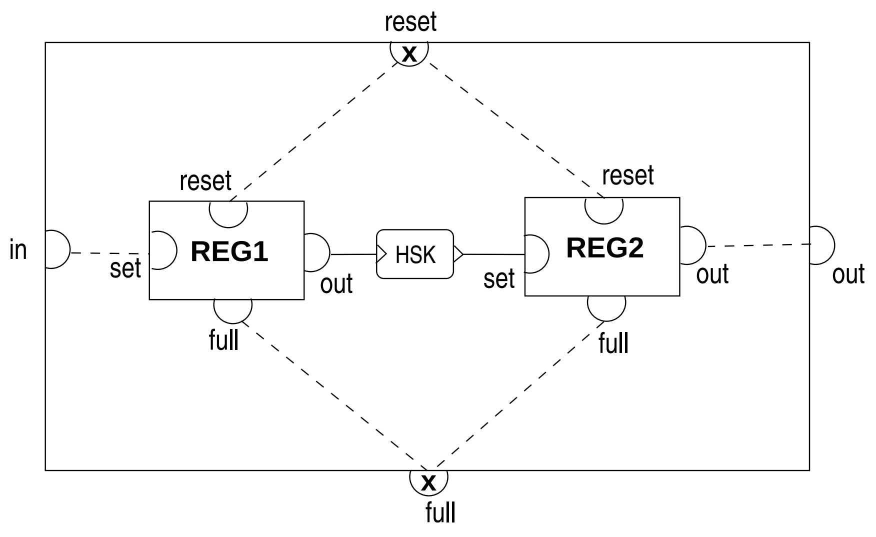


Figure 4.9: Examples of synchronous gate mappings

Two examples of the synchronous gate mapping are illustrated in Figure 4.9, where a network representing a two-place FIFO system is depicted. Each group of dashed lines leading to the same interface port represents a particular many-to-one gate mapping. The fact that these mappings are synchronous is denoted by $'\times'$ symbol inside the target ports. In this particular case, there is one input and one output synchronous mapping. Both mappings are of 2-to-1 character. The input synchronous mapping ensures relaying of the $reset$ event to both register components. The output mapping models the signalling of the global $full$ event just only if both register components are notifying their fullness by their local $full$ events. To all appearance, the following event refinement is enforced by the gate in this example (each event is taken prefixed by the relevant component name to avoid name conflicts):

- $REG1.reset \times REG2.reset \mapsto reset$ – the network $reset$ input event is refined to synchronous occurrence of the respective $reset$ input events in both components;
- $REG1.full \times REG2.full \mapsto full$ – the network $full$ output event is refined to synchronous occurrence of the respective $full$ output events in both components.

It is worth noting that synchronous mapping forces synchronisation of computation in several components in a particular network. Therefore, the synchronous gate affects the overall network computation.

To overcome possible collisions of source event names in the gate mapping from the above mentioned example, all events are annotated by the relevant component name which is unique in the scope of the network topology. The technical requirement of having component names unique in the scope of a network topology would have been too strict and unintuitive for the designer, especially because the component name refers to a network or a leaf which can form a body of more than one component of the particular network topology (component reuse). Therefore all components in the particular network are implicitly indexed by unique numbers. Events are then formally annotated by these numbers (see the Definition 5.4 in Chapter 5). However, not to unnecessarily complicate our examples in this section, herein we will always take the events in gate mappings prefixed by the particular component name and assume unambiguity of component names in the scope of a particular network topology.

In Figure 4.10 there are examples of asynchronous gate mappings depicted. The figure shows a network representing the server part of a client/server system. The server part is hierarchically refined by two server components which back up each other. Asynchronous mapping is determined by the '+' symbol. Similarly to the previous example, there is one input and one output mapping, both of 2-to-1 character. The input mapping allows relaying of the service invocation request to just one of the servers. The output mapping then relays the result to the server part output port.

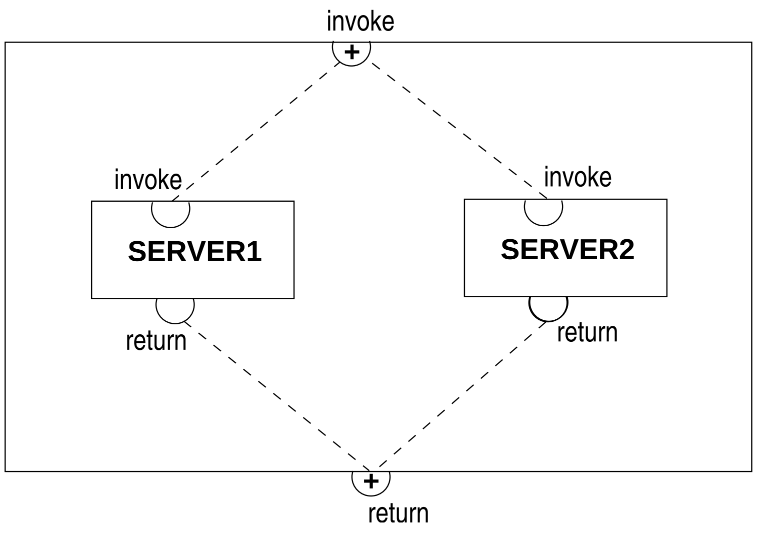


Figure 4.10: Examples of asynchronous gate mappings

The following event refinement is enforced by the asynchronous mapping in this example:

- $SERVER1.invoke \mapsto invoke$ or $SERVER2.invoke \mapsto invoke$ – the network $invoke$ input event is refined to either one (and just one) of the subsidiary components $invoke$ events;
- $SERVER1.result \mapsto result$ or $SERVER2.result \mapsto result$ – the network $result$ output event is refined to either one (and just one) of the subsidiary components $result$ events.

The key property of the asynchronous mapping is that there is no synchronisation considered. Thus, asynchronous gate mappings do not affect the overall network computation.

An example of a universal gate mapping is illustrated in Figure 4.12. There is a network representing a robot production line depicted. All of the robot components are supposed to fulfil the same task. Moreover, assume that each robot has two different states distinguished in its computation. The state-transition diagram describing the robot computation is depicted in Figure 4.11.

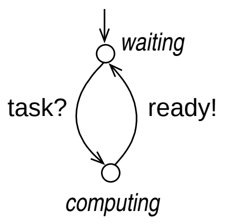


Figure 4.11: Computation model of the $ROBOT$ component

In the first state, it is in standby mode waiting for task to be received in terms of the $task$ event. In the second state, it computes the task and turns back to the standby mode notifying the state change by the $ready$ event. Every task is expected to be transfered to all the robots which are ready to compute. To capture this property, the universal input gate mapping is employed. By that way, the $task$ event is refined into any possible synchronous composition of the task events of the three components. However, if only one of the robots is ready to compute, this composition is degraded to a single event. Similarly, information about the readiness of the robots is also manipulated in terms of the universal gate mapping. To prevent the product line from being inefficiently delayed, the $ready$ event is accepted simultaneously and atomically from all the robots which have already finished the task. More precisely, the following event refinement is enforced by the universal mappings included in this example:

- $ROBOT1.task \mapsto task$ or $ROBOT2.task \mapsto task$
  $ROBOT1.task \times ROBOT2.task \mapsto task$
  $ROBOT1.task \times ROBOT3.task \mapsto task$

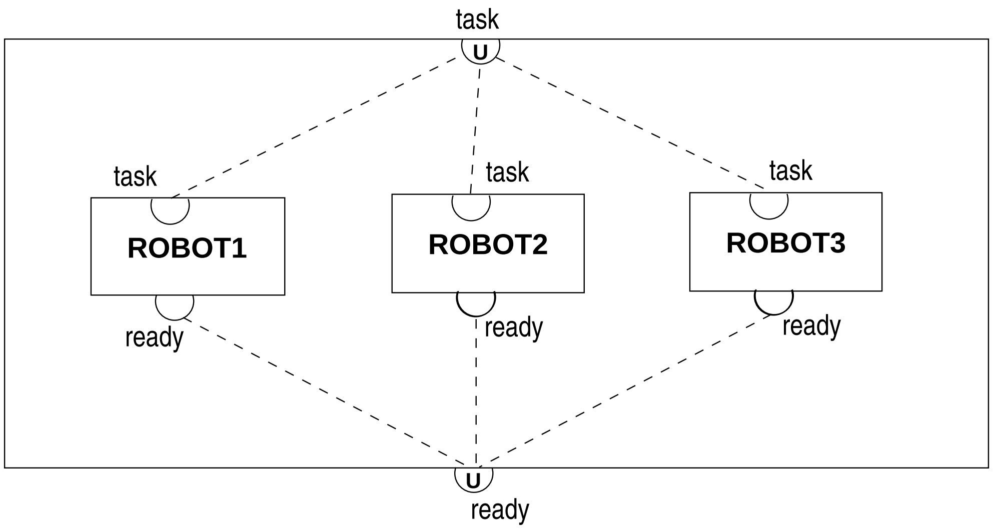


Figure 4.12: Examples of universal gate mappings

$ROBOT2.task \times ROBOT3.task \mapsto task$
$ROBOT1.task \times ROBOT2.task \times ROBOT3.task \mapsto task$

The network $task$ input event is refined to synchronous occurrence of the maximal possible (w.r.t. the current situation of individual component computation) number of the respective components $task$ input events.

- The situation concerning $ready$ event is analogous. The only difference is that the events considered here are of the output type.

Similarly as synchronous mappings, universal mappings affect the overall network computation. As it can be seen from the principle described above, the intuition of state transition based behavioural model of component computation is needed to demonstrate the purpose of universal gate mapping. This notifies that the concept of this kind of gate mapping is deeply wedded to the behavioural model of VCN.

The aspects of many-to-one gate mappings are generalised and summarised in Table 4.1. In the part marked (1) there is shown the semantics of asynchronous mapping. In the part (2) and (3) the semantics of synchronous mapping and universal mapping is described, respectively. All the mappings are shown generalised to arbitrary number of connections (n-to-1 mapping). The (a) parts of all figures correspond to input mappings whereas the (b) parts correspond to output mappings.

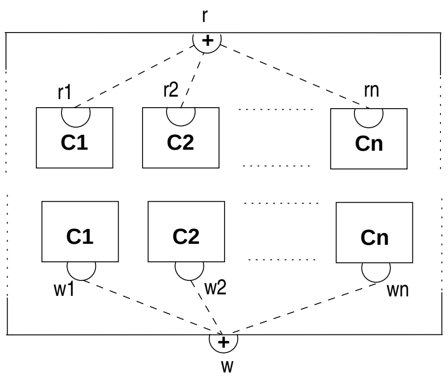
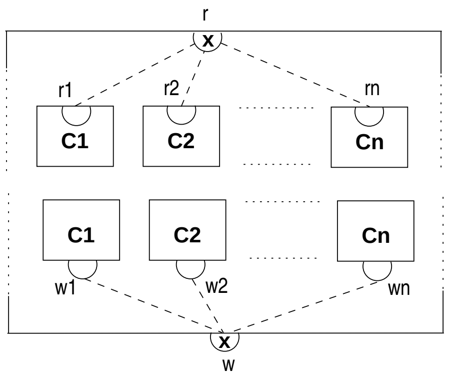
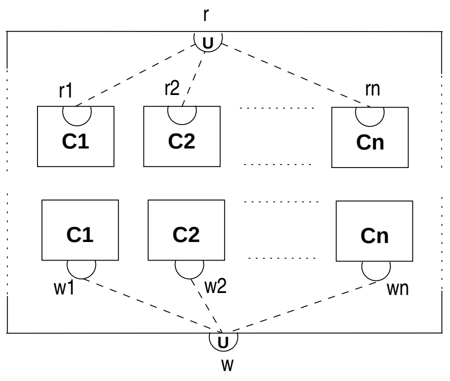
1a) 1b) 	$\{r_i \mapsto r \parallel i \in \{1,...,n\}\}$ $\{w_i \mapsto w \parallel i \in \{1,...,n\}\}$
2a) 2b) 	$\prod_{i\in\{1,...,n\}} r_i \mapsto r$ $\prod_{i\in\{1,...,n\}} w_i \mapsto w$
3a) 3b) 	$\{\prod_{i\in I} r_i \mapsto r \parallel \emptyset \neq I \subseteq \{1,...,n\}\}$ $\{\prod_{i\in I} w_i \mapsto w \parallel \emptyset \neq I \subseteq \{1,...,n\}\}$

Table 4.1: Many-to-one gate mapping variants

4.3.4 Bus and Link Elements

As it has been mentioned in Section 4.2.1, VCN networks consist of components and buses connected together by links. Here we introduce the notion of buses and links in detail.

A *bus* represents some coordination model in a particular place in a network. From the architectural point of view, buses represent architectural connectors, and similarly as components they are stand-alone members of VCN networks. For graphical representation of a bus we use a rectangle with rounded corners.

Links make the glue layer between components and buses. They are responsible for creating network topologies of components interconnected by buses. Each *link* is represented as a solid line connecting the particular component interface port with a bus. With respect to the nature of ports, each link is either input or output according to the type of the relevant port. To emphasise visually this fact, a small triangle oriented in the respective direction is placed at the intersection of the link and the bus rectangle. Arbitrary number of links can be connected to a bus, which allows various components in the network to be coordinated by a single bus. Moreover, a number of links can be employed to connect a component to a particular bus. However, links are assumed to satisfy the condition that there can be at most one link leading from any particular component interface port.

A simple network of two components coordinated by one bus has been illustrated in Figure 4.1. In that example, the coordination mechanism represented by the $COORD$ bus is expected to allow two kinds of atomic *cooperations* of the $SERVER$ and $CLIENT$ components:

1. atomic relaying the client request to server (binary handshake),
2. atomic relaying the result from server to client (binary handshake).

In general, a bus can be sensed as a set of all the possible cooperations which represent the particular coordination mechanism. Each cooperation has the meaning of atomic multi-synchronisation of actions occurring on the ports involved in the cooperation. A cooperation is defined as a set of particular component ports which are linked to the respective bus. In other words, each cooperation consists of a subset of output ports and a subset of input ports. To describe cooperations in a well arranged way we use the syntax of the form

$$\langle \mathrm{set\_of\_output\_ports}/\mathrm{set\_of\_input\_ports} \rangle.$$

The left side of the cooperation is called *input section* and the right side is called *output section*. The following two cooperations express the $COORD$ bus from the above mentioned example:

1. `request/invoke`
2. `result/return`

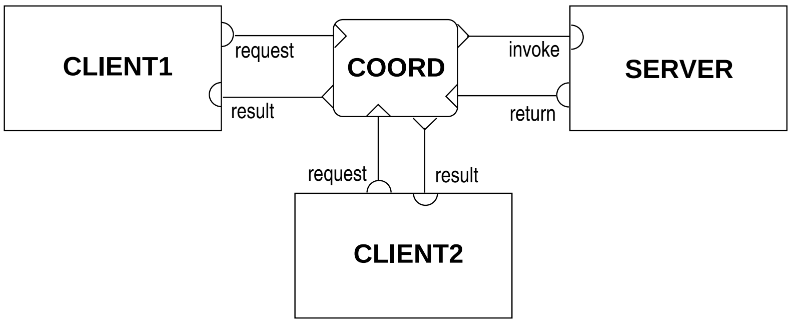


Figure 4.13: Example of multi-synchronisation bus

To demonstrate a more complicated example of a bus, let us extend the previous example to a network topology of two clients cooperating with a server. The resulting network is depicted in Figure 4.13. The server computation is expected not to care about the identification of the clients. In this cooperation protocol it is in competence of each client component to determine if the result replied back belongs to it or not. More specifically, both clients receive the result whenever it is sent (in terms of atomic multi-synchronisation) and each of them decides whether to accept it or ignore. The $COORD$ bus can be now extended to handle such cooperation. Similarly to the gate descriptions, we take each port prefixed by the respective component name to avoid name conflicts. All the cooperations of the $COORD$ bus are now the following:

1. relaying the first client request to the server (binary handshake)
   `CLIENT1.request/SERVER.invoke`
2. relaying the second client request to the server (binary handshake)
   `CLIENT2.request/SERVER.invoke`
3. broadcasting the result to both clients (multi-synchronisation)
   `SERVER.return/CLIENT1.result, CLIENT2.result`

In general, a bus is determined by an arbitrary finite set of cooperations. Cooperations of the particular bus contain ports of components connected to the bus. The topology of a bus and all the components directly connected to it by links is called *star topology*. To overcome possible collisions of port names in a particular cooperation, all ports in cooperations from the example above are annotated by the relevant component name which is unique in the scope of the particular star topology. As we have already noted in the previous subsection, the technical requirement of having component names unique in the scope of a network topology would be too strict and therefore all components in the particular network are implicitly indexed by unique numbers. Ports in cooperations are exactly annotated by these numbers (for details see Definition 5.2 in Chapter 5). However, for the same simplification reasons as in the case of gates, for the purpose of this chapter we will always consider the ports in cooperations prefixed by the particular component name and assume unambiguity of component names in the scope of a particular star topology.

To describe the notion of cooperation in more detail, we focus on its relationship with observable component events at first. Then, with respect to this relationship, we declare the required properties of cooperations. The relationship of a cooperation with the relevant events occurring on the linked ports has the character of mutual matching. The meaning of this matching is that a cooperation, in order to be performed, has to satisfy a particular set of conditions. Especially, we say that the set of component events *matches* the cooperation if all the output events from the set match the input section (output component ports) of the cooperation and the all input events from the set match the output section (input component ports) of the cooperation. The set of component events taken into account is assumed to be just the set of all component events of the star topology which are *enabled* in the particular state of the respective component computation. An event is enabled in a particular computation state if there is some relevant action offered to be performed in the next step of the respective component computation. The sense of a cooperation is atomic synchronisation of actions of all the events involved. Naturally, such an atomic synchronisation is supposed to be an internal system action unobservable by the environment. But what events can exactly match a particular cooperation? In Figure 4.14 there is a network topology of two producing and three consuming components depicted. The purpose of the $COORD$ bus is to atomically communicate the content of one of the put events to all producers prepared for receiving it. In other words, it is a multicasting coordination mechanism — a weak variant of broadcast in which not all consumers must necessarily receive the information, but only the maximal prepared group of them has to do so. In Table 4.2 there are all the cooperations defined by $COORD$ bus listed in the left column. In the right column, all the possible sets of component events are shown. Each component set matches just the cooperation on the same line. In next paragraphs we discuss the details of the cooperation matching.

The first requirement (*completeness*) of cooperation matching is that the cooperation must be complete, so that on each port which appear in the cooperation there must occur a relevant component event. Thus, all ports forming the cooperation must be exactly involved in synchronisation. The

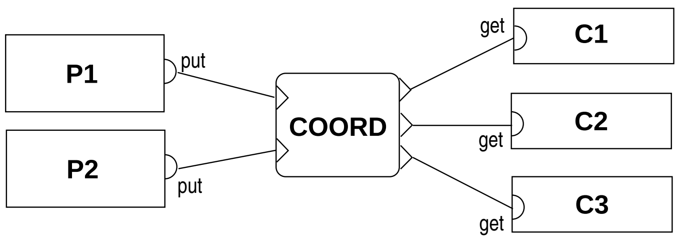


Figure 4.14: Example of a multicasting bus

Cooperations of $COORD$	Sets of enabled component events
$\mathtt{P_1.put/C_1.get}$	$\{\mathtt{P_1.put!}, \mathtt{C_1.get?}\}$
$\mathtt{P_1.put/C_2.get}$	$\{\mathtt{P_1.put!}, \mathtt{C_2.get?}\}$
$\mathtt{P_1.put/C_3.get}$	$\{\mathtt{P_1.put!}, \mathtt{C_3.get?}\}$
$\mathtt{P_1.put/C_1.get}, \mathtt{C_2.get}$	$\{\mathtt{P_1.put!}, \mathtt{C_1.get?}, \mathtt{C_2.get?}\}$
$\mathtt{P_1.put/C_1.get}, \mathtt{C_3.get}$	$\{\mathtt{P_1.put!}, \mathtt{C_1.get?}, \mathtt{C_3.get?}\}$
$\mathtt{P_1.put/C_2.get}, \mathtt{C_3.get}$	$\{\mathtt{P_1.put!}, \mathtt{C_2.get?}, \mathtt{C_3.get?}\}$
$\mathtt{P_1.put/C_1.get}, \mathtt{C_2.get}, \mathtt{C_3.get}$	$\{\mathtt{P_1.put!}, \mathtt{C_1.get?}, \mathtt{C_2.get?}, \mathtt{C_3.get?}\}$
$\mathtt{P_2.put/C_1.get}$	$\{\mathtt{P_2.put!}, \mathtt{C_1.get?}\}$
$\mathtt{P_2.put/C_2.get}$	$\{\mathtt{P_2.put!}, \mathtt{C_2.get?}\}$
$\mathtt{P_2.put/C_3.get}$	$\{\mathtt{P_2.put!}, \mathtt{C_3.get?}\}$
$\mathtt{P_2.put/C_1.get}, \mathtt{C_2.get}$	$\{\mathtt{P_2.put!}, \mathtt{C_1.get?}, \mathtt{C_2.get?}\}$
$\mathtt{P_2.put/C_1.get}, \mathtt{C_3.get}$	$\{\mathtt{P_2.put!}, \mathtt{C_1.get?}, \mathtt{C_3.get?}\}$
$\mathtt{P_2.put/C_2.get}, \mathtt{C_3.get}$	$\{\mathtt{P_2.put!}, \mathtt{C_2.get?}, \mathtt{C_3.get?}\}$
$\mathtt{P_2.put/C_1.get}, \mathtt{C_2.get}, \mathtt{C_3.get}$	$\{\mathtt{P_2.put!}, \mathtt{C_1.get?}, \mathtt{C_2.get?}, \mathtt{C_3.get?}\}$

Table 4.2: Multicasting cooperations and the matching component events

second requirement (*consistency*) limits the number of component ports involved in a cooperation to at most one port per each component. The last requirement (*maximality*) deals with limiting of possible nondeterminism in choosing the particular cooperation if there is more than one which matches a specific set of component events (i.e., all satisfy both of the requirements above). We require the cooperation containing maximal number of involved events to be chosen. In our example, if we take into account the set of events $\{\mathtt{P_1.put}, \mathtt{C_1.get}, \mathtt{C_2.get}, \mathtt{C_3.get}\}$, the only cooperation which is matched by this set is $\mathtt{P_1.get/C_1.get}, \mathtt{C_2.get}, \mathtt{C_3.get}$. None of the cooperations placed above this one in the table can be chosen. Thus the nondeterminism of coordination matching is limited just to the set of maximal cooperations w.r.t. the particular set of events. It is worth noting that no other synchronisation than listed in Table 4.2 can be performed in our example.

With respect to the requirements above, we conclude claiming that in VCN each cooperation has to satisfy the following properties:

- *atomicity* — each cooperation is sensed as an uninterruptible discrete event,
- *completeness* — all ports in the cooperation are exactly involved in the relevant synchronisation (there is some action occurring on each of these ports),
- *closeness* — each cooperation is treated as an internal τ-action unobservable by the environment of the network topology involved in the cooperation,
- *consistency* — in each cooperation at most one port of a particular component can appear,
- *maximality* — the maximal cooperation is chosen for the particular set of enabled component events.

The character of coordination models from the above mentioned examples is similar in its nature to coordinative aspects of gates introduced in previous subsection. In Figure 4.15 it is shown how cooperations of buses from the previous examples can be visualized as groups of internal lines connecting the inputs of a bus to its outputs.

1. The first bus is the $COORD$ bus from Figure 4.1. Here the dashed line represents the `request/invoke` cooperation and the solid line the `return/result` cooperation.
2. The $COORD$ bus from Figure 4.13 is depicted as the second bus. The dashed lines joined by the '+' connector represent the $\mathtt{CLIENT}_1\mathtt{.request/SERVER.invoke}$ and $\mathtt{CLIENT}_2\mathtt{.request/SERVER.invoke}$ cooperations.

   The solid lines joined by the '$\times$' connector represent the $\mathtt{SERVER.return/CLIENT}_1\mathtt{.result}, \mathtt{CLIENT}_2\mathtt{.result}$ cooperation.
3. The third bus is the multicasting bus taken from Figure 4.14. The dashed '$\cup$'-joined group of lines corresponds to the set of cooperations from the first line of Table 4.2 and similarly the group of solid lines corresponds to the second line of the table.

In all the examples above, the $COORD$ bus models passive coordination. In other words, it is in competence of component computation to implement the protocol of correct cooperation. To allow modelling of protocol cooperation completely in competence of buses, their expressiveness must be extended with sufficient behavioural logic. To this end, sets of

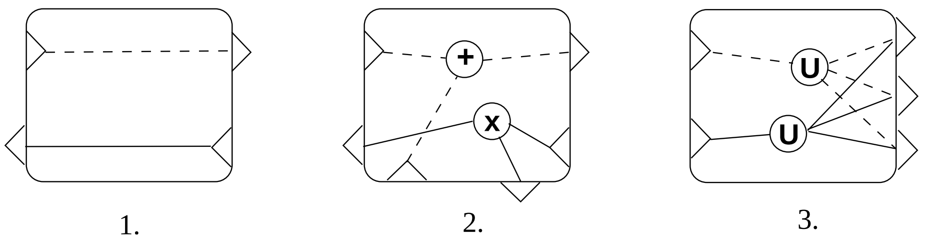


Figure 4.15: Schemes of three different versions of the $COORD$ bus

cooperations can be extended with state-transition control (see Chapter 7 for details). This way, buses which model complex protocols (active coordination models) of component cooperation can be defined. Moreover, asynchronous coordination models (buffers) can be also modelled. However, in this chapter we abstract from the coordinative dynamism and treat buses just as static sets of cooperations. This abstraction allows us to introduce structural aspects of VCN without dealing with details related to behavioural aspects, which form the other dimension of VCN semantics.

Finally, we would like to highlight heterogeneity of the coordination framework presented here. In the context of a particular network, several buses of different coordination models can be employed. This possibility allows to mix different coordination mechanisms at level of a single network. In other words, each level of coordination layer in network hierarchy can be heterogeneous with respect to a variety of coordination models employed.

It is also worth noting that in our setting buses and links are allowed to interconnect components only at a single level of network hierarchy. The decision why we define buses in such a way is based on the intention of component-based architectural description methodology to enable the design and its analysis to be compositional. To capture this intention, we separate the notions of inner-level (horizontal) and inter-level (vertical) coordination in our hierarchical formalism. Inner-level connections are realised by a very complex notion of buses while inter-level connections are captured by gates incorporating only simple models of coordination as it has been described in previous subsection. On the one hand, such a setting simplifies definition of compositional behavioural model for VCN, and on the other hand, it goes well with the intuition of hierarchical architectural design.

Notes on Value-passing Interaction

Throughout the thesis we deal with a version of VCN language which abstracts from passing of values during component interaction. However, the semantics of buses is defined in such a way that the value-passing feature can be easily added. This can be realized similarly as in value-passing versions of CCS [Mil89] or CSP [Hoa85]. In particular, an input value variable can be added to every input event occurrence and each output event can be equipped with some particular value. Semantics of cooperations can be then lifted to allow assignment of output values to particular input variables.

4.3.5 Bus Classes and Link Ranks

For buses representing common coordination primitives such as handshake or broadcast, and also for some complicated coordination models which are typically reused throughout the design, VCN offers a general bus specification method. More particularly, VCN provides the concept based on the idea of a general template, so-called *bus class*, which represents a particular coordination model (e.g., handshake, broadcast, ...). Such a template is general in the sense that it is not fixed to a concrete set of input and output ports. From the template, particular bus cooperations can be automatically generated for concrete cases of network topologies.

To illustrate the intuition of bus classes, let us revisit the example of a network from Figure 4.9 representing a FIFO component. It contains two registers which communicate by handshake. Instead of direct specification of the HSK bus by declaring the set of respective cooperations including just the $\mathtt{REG}_1\mathtt{.out}/\mathtt{REG}_2\mathtt{.in}$ cooperation, the bus can be determined by setting of its class. In this case, the handshake class, denoted $\mathcal{B}_{\mathrm{HSK}}$, is applied. If the class is properly defined then all the designer has to do in order to add the hand-shake coordination model to the design is to instantiate the bus class. In the next paragraph, we focus just on the principle of bus classes and their instantiation.

In general, a bus class is defined with two parameters — the set of related output component ports and the set of related input component ports. Because the output component ports are viewed as inputs to the bus, we denote `In` the set of related output component ports (set of bus inputs). Analogously, the set of related input component ports (set of bus outputs) is denoted `Out`. Now with respect to these parameters the bus class can be defined by a constraint of these parameters and a predicate characterising the relevant cooperations. The predicate is typically quantified over the both parameters. In our example, the handshake bus class $\mathcal{B}_{\mathrm{HSK}}(\mathtt{In}, \mathtt{Out})$ has the following form:

- $\mathtt{In}, \mathtt{Out} \neq \emptyset$
- $\forall i \in \mathtt{In},\ o \in \mathtt{Out} :\ i/o \in \mathcal{B}_{\mathrm{HSK}}$

In similar way, bus classes for some other common coordination primitives are shown in Table 4.3.

Broadcast	**Multicast**
$\mathcal{B}_{\mathrm{BCST}}(\mathtt{In},\mathtt{Out}) :=$ • $\mathtt{In},\mathtt{Out} \neq \emptyset$ • $\forall i \in \mathtt{In} :\ i/\mathtt{Out} \in \mathcal{B}_{\mathrm{BCST}}$	$\mathcal{B}_{\mathrm{MCST}}(\mathtt{In},\mathtt{Out}) :=$ • $\mathtt{In},\mathtt{Out} \neq \emptyset$ • $\forall i \in \mathtt{In},\ O \subseteq \mathtt{Out},\ O \neq \emptyset :$ $i/O \in \mathcal{B}_{\mathrm{MCST}}$

Table 4.3: Bus classes for synchronous broadcast and multicast

The bus classes mentioned above, if used in settings of greater number of ports, lead to buses with high number of cooperations. For example, the class of handshake buses will generate the bus depicted in Figure 4.16 for two input and two output ports (if we assume all the bus outputs and inputs to be of different components).

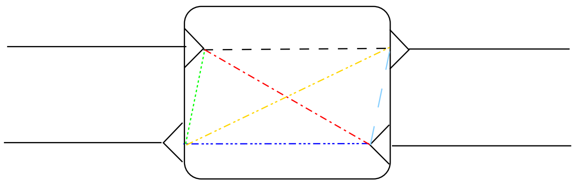


Figure 4.16: Cooperations of the four port bus generated from $\mathcal{B}_{\mathrm{HSK}}$ class

To allow the principle of bus classes to be used for generating of buses such as the first *COORD* bus in Figure 4.15, some general mechanism of marking of bus inputs and outputs has to be added. Therefore, we introduce the notion of *link ranks*. Any link can be optionally provided with a natural number identifying the link rank. In our example of the handshake bus with two input and two output ports we can rank each of its links. The relevant network extended with link ranks is depicted in Figure 4.17.

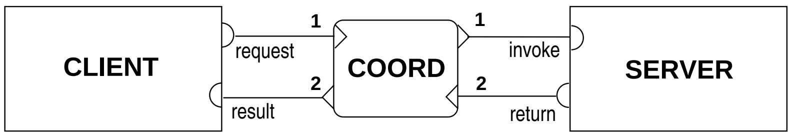


Figure 4.17: Example of a network with ranked links

The In and Out parameters of bus classes are now extended with a rank function. Rank function returns for each bus input or output the rank number of the respective link. We denote the rank of a specific port p as $\mathtt{rank}(p)$.

The handshake bus class can be now extended to the version which cooperates only the pairs of events occurring on ports of the same rank. The relevant definition of the bus class $\mathcal{B}_{\mathrm{HSK}}(\mathtt{In}, \mathtt{Out}, \mathtt{rank})$ can be now extended with a rank constraint:

- $\mathtt{In}, \mathtt{Out} \neq \emptyset$
- $\forall i \in \mathtt{In},\ o \in \mathtt{Out} :\ i/o \in \mathcal{B}_{\mathrm{HSK}} \wedge \mathtt{rank}(i) = \mathtt{rank}(o)$

In general, we assume that the bus class with a rank constraint can be applied only if each link of the particular bus has a rank defined.

Now if we apply the extension of the bus class $\mathcal{B}_{\mathrm{HSK}}$ defined above to specification of the four port $COORD$ bus from our example, the bus will contain the cooperations as depicted in Figure 4.18.

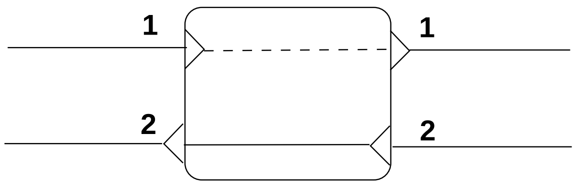


Figure 4.18: Bus specified by the handshake class with a rank constraint

Bus Class Specification Language

All bus classes mentioned above were defined by universal quantification which ranged over the particular set of bus input (resp. output) ports. Moreover, in the predicate of the multicast bus class there appeared an universal quantifier which ranged over subsets of the set of bus outputs. In order to make specification of bus classes simpler and encodeable into comprehensible ASCII text files, we define the so-called *bus class specification language* (see Definition 6.5 in Chapter 6) in which we fix a simple syntax of predicates, i.e., we make universal quantifiers implicit. Hence the universal quantification is omitted in predicates specified in the bus class specification language. In Table 4.4 there are all the above-mentioned bus classes shown encoded using the bus class specification language.

A predicate is represented in the bus specification language as a list of so-called cooperation patterns. Each cooperation pattern consists of an input section (in front of the slash) and an output section (behind the slash). The input section contains a list of mutually different bus input variables, so-called *input handles*, each of which is implicitly quantified over the set `In`. Similarly, the output section is represented as a list of bus output variables, so-called *output handles*, each of which is quantified over the set `Out`. There is a special constant '`all`' which can be used in the input (or output)

Binary Handshake	**Ranked Binary Handshake**
$\mathcal{B}_{\mathrm{BCST}}(\mathtt{In},\mathtt{Out}) := \{$ $\mathtt{In},\mathtt{Out} \neq \emptyset$ $\mathtt{i/o}$ $\}$	$\mathcal{B}_{\mathrm{MCST}}(\mathtt{In},\mathtt{Out},\mathtt{rank}) := \{$ $\mathtt{In},\mathtt{Out},\mathtt{rank} \neq \emptyset$ $\mathtt{i/o} \wedge \mathtt{rank(i)} = \mathtt{rank(o)}$ $\}$
Broadcast	**Multicast**
$\mathcal{B}_{\mathrm{BCST}}(\mathtt{In},\mathtt{Out}) := \{$ $\mathtt{In},\mathtt{Out} \neq \emptyset$ $\mathtt{i/all}$ $\}$	$\mathcal{B}_{\mathrm{MCST}}(\mathtt{In},\mathtt{Out}) := \{$ $\mathtt{In},\mathtt{Out} \neq \emptyset$ $\mathtt{i/}\triangleright\mathtt{all}$ $\}$

Table 4.4: Bus classes specified using the bus class specification language

section to denote the list of all bus inputs (resp. outputs) included in `In` (resp. `Out`). The operator '$\triangleright$' placed behind the slash denotes a so-called *right universal unfolding operator* which expands the respective cooperation pattern for all non-empty subsets of the output section. Similarly, the symmetric left universal unfolding operator '$\triangleleft$' placed in front of the slash can be used for expansion of the input section provided that both unfolding operators can be combined in a particular cooperation pattern.

To conclude the summary of bus specification language features, we have to emphasize that cooperation patterns can be additionally equipped with so-called *memory handles* which introduce memory manipulation operations. This allows specification of buses equipped with internal memory, i.e., asynchronous kinds of coordination models can be specified in terms of such extended bus class specifications.

Chapter 5

VCN: Structure

In this chapter we focus on definition of the structural part of VCN – the formal representation of VCN networks realised in the language of structural terms.

5.1 Formal Representation

The goal of this section is to establish formal representation of the visual notation of networks introduced in previous chapters. To this end, the theory of VCN *structural terms* is developed. Structural terms give the VCN graphical notation unambiguous semantics defined in terms of sets, relations, and other elementary notions of set theory. We prefer this approach to abstract data type based formalisation. The reason for that lies in the fact that notions of the set theory fit well the nature and richness of VCN syntactic constructs without unnecessary complicating the formalisation.

5.1.1 Components, Interfaces, and Gates

As it has been mentioned in the previous section, each component is determined by its interface, body, and gate. Component interface consists of *ports*. Each component body is characterised by its *alphabet* — the set of *events* the component can engage in during its computation. *Gate* relates a body of a particular component to the component interface. The component body can be a leaf, in the case of a primitive component, or a network of sub-components, in the case of compound component.

We begin with definition of basic notions and we also establish the elementary notation of the VCN theory.

Ports

First of all we present formal definition of ports. Each port is given by its label and type (input or output port). Conceptually, we fix a countable set of labels and on its basis we define two countable sets globally representing inexhaustible repositories of ports of both kinds.

Definition 5.1 *Fix $\mathcal{L}$ a countable set of* labels *and assume $\tau \notin \mathcal{L}$. Define* ports *as members of the set $\mathcal{P} \overset{\text{df}}{=} \mathcal{L} \times \{in, out\}$. Further define projections of $\mathcal{P}$, the set of* output ports $\mathcal{W} \overset{\text{df}}{=} \{p \in \mathcal{P} \parallel p = \langle l, out \rangle, l \in \mathcal{L}\}$*, and the set of* input ports $\mathcal{R} \overset{\text{df}}{=} \{p \in \mathcal{P} \parallel p = \langle l, in \rangle, l \in \mathcal{L}\}$.

As it has been mentioned in the previous section, by reason of avoiding ambiguity of port labels in the scope of network, we annotate each port with an index denoting its component whenever it is necessary. We assume that each component in a network is identified by a unique natural number. By such annotation, the required port label unambiguity in the scope of entire network topology is guaranteed. Moreover, unambiguity of port labels connected to a particular bus is also guaranteed.

Definition 5.2 *Define set of* annotated ports $\mathcal{P}^{\sharp} \overset{\text{df}}{=} \{\langle p, i \rangle \parallel p \in \mathcal{P}, i \in \mathcal{N}\}$. *Further define the respective projections as sets of* annotated output *and* annotated input ports*, $\mathcal{W}^{\sharp} \overset{\text{df}}{=} \{\langle w, i \rangle \parallel w \in \mathcal{W}, i \in \mathcal{N}\}$, and $\mathcal{R}^{\sharp} \overset{\text{df}}{=} \{\langle r, i \rangle \parallel r \in \mathcal{R}, i \in \mathcal{N}\}$.*

Notation 5.3 *For some $i \in \mathcal{N}$ we denote the respective sets of ports annotated by i in the following way:*

- $\mathcal{P}^{\sharp_i} \overset{\text{df}}{=} \{\langle p, i \rangle \parallel p \in \mathcal{P}\}$
- $\mathcal{W}^{\sharp_i} \overset{\text{df}}{=} \{\langle w, i \rangle \parallel w \in \mathcal{W}\}$ and $\mathcal{R}^{\sharp_i} \overset{\text{df}}{=} \{\langle r, i \rangle \parallel r \in \mathcal{R}\}$

Members of the set $\mathcal{P}$ are usually represented by symbols $p, p_1, p_2, \ldots$, members of $\mathcal{W}$ by $w, w_1, w_2, \ldots$, and, finally, members of $\mathcal{R}$ by $r, r_1, r_2, \ldots$ Note the important fact that $\mathcal{W} \cap \mathcal{R} = \emptyset$. By the notation $p^{\sharp_i}$ for some $i \in \mathcal{N}$ we mean $\langle p, i \rangle \in \mathcal{P}^{\sharp}$. Thus an input port $r \in \mathcal{R}$ annotated by $i \in \mathcal{N}$ is denoted $r^{\sharp_i}$. Annotated output ports are denoted in the same way. Whenever the annotation number i is not important to be specified in a particular context for an annotated port $p^{\sharp_i}$, we omit the upper index '$\sharp i$' and write simply 'p'.

For a specific set of all (unannotated) ports $P \subseteq \mathcal{P}$, the set of all these ports additionally annotated by $i \in \mathcal{N}$ is denoted $P^{\sharp_i}$. For all annotation numbers the notation $P^{\sharp}$, $P^{\sharp} \overset{\text{df}}{=} \bigcup_{i \in \mathcal{N}} P^{\sharp_i}$, is used.

Events

Ports are related with another elementary notion of VCN — *events*. Ports represent a structural (syntactical) notion whereas events represent a behavioural (semantical) notion. The relation between a port and an event is depicted in Figure 5.1. A port is a place in the component architecture

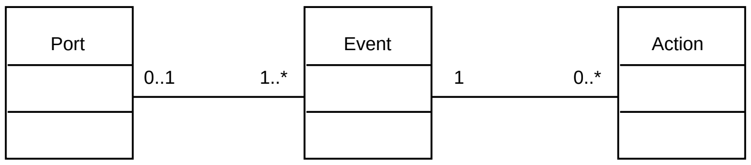


Figure 5.1: Relations between ports and events

where specific group of events (determined by a particular gate mapping) can occur. The difference between the term "event" and the term "event occurrence" is treated by the notion of "action". Wherever it is important to take such a slight distinction into account, the term "action" is used to refer an event occurrence. Each event can be observed on a particular port of a component only if it is related with that port by gate. Intuitive meaning of an event is atomic transmission/reception of some piece of information to/from the particular port. More specifically, events are atomic elements of component computation responsible for transmitting a value (a signal level change) to the component environment, or receiving a value (a signal level change) from the component environment. To simplify the definitions, we abstract from value passing feature of events. In such an abstract setting, an event occurrence means an action of sending or receiving a signal. Moreover, we define a specific untyped event denoted τ which is used to handle internal computation actions.

Formally, each event is determined by its label and type (input and output) in the similar way like a port. In consequence, in the following definition we establish inexhaustible repositories of input and output events. They differ from ports only in their notation. Input and output events are determined by a label followed by the symbol '?' and '!', respectively. We also define annotation of events by natural numbers. Event annotation has the meaning of unambiguous marking of events occurring in different components in a particular network. This way, conflicts of event labels in the scope of a network are avoided. We will use event annotation whenever such conflicts can potentially arise.

Definition 5.4 *Define* $\mathcal{E}_{\mathcal{W}} \overset{\mathrm{df}}{=} \{w! \parallel w \in \mathcal{L}\}$ *the set of* output events *and* $\mathcal{E}_{\mathcal{R}} \overset{\mathrm{df}}{=} \{r? \parallel r \in \mathcal{L}\}$ *the set of* input events*. Further define the set of all* events *as the countable set* $\mathcal{E} \overset{\mathrm{df}}{=} \mathcal{E}_{\mathcal{W}} \cup \mathcal{E}_{\mathcal{R}} \cup \{\tau\}$.

For a particular set of ports $P \subseteq \mathcal{P}$, the set of events homonymous with ports in *P is denoted $\mathcal{E}(P)$ and defined $\mathcal{E}(P) \overset{\text{df}}{=} \{w! \parallel \langle w, out\rangle \in P\} \cup \{r? \parallel \langle r, in\rangle \in P\}$.*

Define the set of annotated events $\mathcal{E}^\sharp \overset{\text{df}}{=} \{\langle e, i\rangle \parallel e \in \mathcal{E} \setminus \{\tau\}, i \in \mathcal{N}\}$. *For some $i \in \mathcal{N}$ the set of all events annotated by i is denoted $\mathcal{E}^{\sharp_i}$ and defined $\mathcal{E}^{\sharp_i} \overset{\text{df}}{=} \{\langle e, i\rangle \| e \in \mathcal{E} \setminus \{\tau\}\}$.*

Sets of annotated output events *and* annotated input events *are denoted in similar way as unannotated versions, analogously to its port counterparts.*

Finally, for a particular set of annotated ports $P \subseteq \mathcal{P}^\sharp$, the set of annotated events homonymous with annotated ports in *P is denoted $\mathcal{E}^\sharp(P)$ and defined $\mathcal{E}^\sharp(P) \overset{\text{df}}{=} \{\langle w!, i\rangle \parallel \langle\langle w, out\rangle, i\rangle \in P\} \cup \{\langle r?, i\rangle \parallel \langle\langle r, in\rangle, i\rangle \in P\}$.*

Note 5.5 *Note that the internal event 'τ' is not included in the repository of annotated events. The reason for that is based on the intuition of higher-level sense of annotated events w.r.t. VCN hierarchy. In particular, the meaning of τ-events is explicit at the level of leaves whereas at higher levels it is solely implicit (denoting multisynchronization). Therefore we treat τ-events separately from observable annotated events to handle their exclusiveness.*

Notation 5.6 *For a particular set of unannotated events $E \subseteq \mathcal{E}$ such that $\tau \notin E$, the set of these events annotated by $i \in \mathcal{N}$ is denoted $E^{\sharp_i}$, $E^{\sharp_i} \overset{\text{df}}{=} \{e^{\sharp_i} \parallel e \in E\}$.*

To avoid over-complicating of future definitions, we consider a special kind of annotation, the 0-annotation, to handle events for which the annotation is not necessary but technically required due to syntactic reasons. Formally, denote $\mathcal{E}^{\sharp_0}$ the following set of annotated events, $\mathcal{E}^{\sharp_0} = \{e^{\sharp_0} \parallel e \in \mathcal{E} \setminus \{\tau\}\}$.

Component Interface

A component interface is determined by a group of ports. Each port can be assigned to a particular *role* in the context of an interface. Relations among all component elements are illustrated in Figure 5.2.

Notation 5.7 *Denote* Roles *a countable set of* roles. *The members of this set are denoted $\rho_1, \rho_2, \ldots$*

Definition 5.8 *Define* (component) interface *I as a pair $\langle P, \varrho\rangle$, $P \neq \emptyset$ of finite set of ports $P \subset \mathcal{P}$ and a role mapping $\varrho : P \to$* Roles, *satisfying either one of the following conditions:*

- *for all $p \in P$ $\varrho(p)$ is defined*
- *for all $p \in P$ $\varrho(p)$ is undefined ($\varrho(p) = \perp$).*

Notation 5.9 *For interface $I = \langle P, \varrho\rangle$ we denote $ports(I)$ the set of its ports, $ports(I) \overset{\text{df}}{=} P$, and $role(I)$ its role mapping function, $role(I) \overset{\text{df}}{=} \varrho$.*

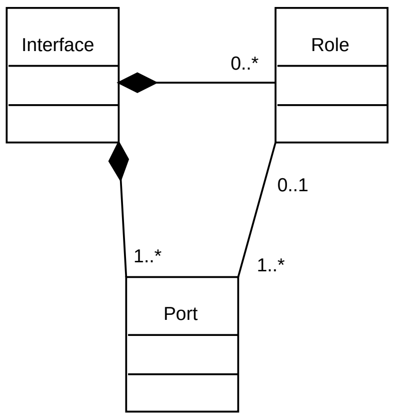


Figure 5.2: Relationships among interfaces, roles, and ports

Component Body

The main functional part of a component is its body. *Component body* is a logical element which is represented either by a *leaf* or a *network*. The common characteristic property which determines component body in both cases is the finite *alphabet* of events which the component can perform during its computation.

The most basic form of a component body is a *leaf*. Every leaf is supposed to represent a model or an implementation of a computational unit (e.g, a reactive program). In particular, that can be any structure which gives semantics to events in terms of actions occurring in discrete time instants during computation. For example, it can be either some abstract logic specification (e.g., a formula of a temporal logic) or some operational model (e.g., a state-transition system). In this thesis we focus on the latter case, which is treated in Chapter 7. The alphabet of a particular leaf is given as a finite set of all events which can occur during the leaf computation.

An important fact concerning the component body alphabet is annotation of events. Event annotation is crucial in the case when the component body is represented by a network. In the case of a leaf component body, no event annotation is necessary. However, to keep the notion of component body alphabet uniform, we assume events of leaf alphabet to be implicitly annotated by 0 (w.r.t. Notation 5.6).

Definition 5.10 *For each component body S denote $\alpha(S)$ its* alphabet, *defined as a finite set of (annotated) events $\alpha(S) \subset \mathcal{E}^\sharp$ and satisfying:*

- *If S is a leaf then $\alpha(S) \subset \mathcal{E}^{\sharp_0}$.*
- *If S is a network then $\alpha(S) \subset \mathcal{E}^\sharp \setminus \mathcal{E}^{\sharp_0}$.*

Notation 5.11 *The set of all leaves is denoted* Leaves.

The more complicated form of a component body is network. We refer the reader to Section 5.1.4, where the notion of network and its alphabet is defined.

Gate

Another element of a component structure is a *gate*. As it has been explained in Section 4.3.3, gate makes the glue layer. It relates the white box view of the component (body) with its black box view (interface). Gate has the meaning of a function, so called *gate function*, which for events observable at the component body level (white box event) returns specific events at the interface level (black box events). The gate function can also hide some component body events by mapping them to the unobservable τ event. Formally, the gate function is defined as a set of *gate mappings*, each of which can be additionally equipped with information about its *type*. Gate mappings provide a formal representation of dashed lines — the graphical notation of gates. In the following definition, the notions of gate, gate mapping, gate mapping type, and gate function are established.

Notation 5.12 *For an arbitrary set X we denote $\|X\|$ its cardinality.*

Definition 5.13 *Let I a component interface, $p \in ports(I)$ its port, and S a component body. A* gate mapping *g* of the body *S* to the port *p is a partial function g, $g : \alpha(S) \to \{p\}$, satisfying:*

- $$\forall e \in \alpha(S) : \; e \in \mathcal{E}_{\mathcal{W}^\sharp} \Leftrightarrow p \in \mathcal{W} \tag{5.1}$$
- $$\exists e \in \alpha(S) : \; g(e) = p \tag{5.2}$$
- $$\text{If for some } i \in \mathcal{N} \text{ there exists } e^{\sharp_i} \in dom(g) \text{ then} \tag{5.3}$$
  $$\forall e'^{\sharp_i} \in dom(g),\, e'^{\sharp_i} \neq e^{\sharp_i} : \; g(e'^{\sharp_i}) \text{ is not defined.}$$
- $$\text{If } S \in \text{Leaves then } \|dom(g)\| = 1. \tag{5.4}$$

For gate mapping g we denote $type(g) \in \{+, \times, \cup, \bot\}$ its gate mapping type.

Define a gate *G* relating the body *S* to the interface *I as a pair $G \overset{\text{df}}{=} \langle map_G, type_G \rangle$ of a nonempty finite set map_G of gate mappings and a function $type_G$ determining type of each mapping, all satisfying the following conditions:*

- *For each $g \in map_G$:*
  - *If $\|dom(g)\| = 1$ then $type_G(g) = \bot$.*
  - *If $\|dom(g)\| > 1$ then $type_G(g) \neq \bot$.*

- *For each $p \in ports(I)$:*

$$\exists g \in map_G : g^{-1}(p) \neq \emptyset \wedge \forall g' \in map_G, g' \neq g : g'^{-1}(p) = \emptyset \tag{5.5}$$

- *For all $e \in \alpha(S)$ either one of the following conditions holds:*

$$\exists g \in map_G : g(e) \text{ def.} \wedge \forall g' \in map_G, g' \neq g : g'(e) \text{ not def.} \tag{5.6}$$
$$\forall g \in map_G : g(e) \text{ not def.} \tag{5.7}$$

Finally, for a gate G relating the body S to the interface I define a gate function $gate_G$ *as a function $gate_G : \alpha(S) \to \mathcal{E}$ defined for each $e \in \alpha(S)$ by the following equations:*

- *$gate_G(e) = g(e)$ if $g \in map_G$ such that $g(e) = p$ for some $p \in ports(I)$;*
- *$gate_G(e) = \tau$, otherwise.*

Additionally, in the former case we denote $type(gate_G(e))$ the type of the particular mapping g, $type(gate_G(e)) \overset{\text{df}}{=} type_G(g)$. In the latter case, we declare $type(gate_G(e)) \overset{\text{df}}{=} \perp$.

Notation 5.14 *The set of all gates is denoted* Gates.

The condition (5.1) in the above-mentioned definition states formally the event type preservation requirement introduced in Section 4.3.3. The condition (5.2) ensures that a gate mapping cannot be an empty function and the condition (5.3) guarantees a gate mapping not to be defined simultaneously for any two different events of one sub-component in the component body. The condition (5.4) requires a gate mapping to be of the one-to-one kind in the case when the body is a leaf. The requirement on the types of gate mappings in a particular gate declares that each many-to-one mapping must be of nontrivial type. The condition (5.5) ensures surjectivity of the gate function. Conditions (5.6) and (5.7) guarantee correctness of a gate mapping function provided that for each component body event there can be at most one gate mapping defined. The gate function which is generated from a gate is a total function and for each component body event for which no gate mapping is defined the gate function returns the internal τ-event.

Lemma 5.15 *For each gate G the gate function $gate_G$ is correctly defined. Moreover, it is a surjective and total function.*

Proof:
Let G be a gate relating the body S with the interface I. Surjectivity follows obviously from the condition (5.5). The totality is achieved directly in the definition of $gate_G$ function.

Correctness:

It has to be proved that $\forall e_1, e_2 \in gate_G.\, e_1 = e_2 \Rightarrow gate_G(e_1) = gate_G(e_2)$. We follow the proof by contradiction.

Assume $e_1 = e_2$, and claim by contradiction $gate_G(e_1) \neq gate_G(e_2)$. Applying the prescription of $gate_G$ from definition 5.13 we get the following two cases:

- $gate_G(e_1) = g(e_1)$ for some $g \in map_G$ such that $g(e_1) = p$ for some $p \in ports(I)$, and $gate_G(e_2) = g'(e_2)$ for some $g' \in map_G$ such that $g'(e_2) = p$ for some $p \in ports(I)$.

  In this case, $g, g' \in map_G$ must be both defined for the same event $e_1 = e_2$. By conditions (5.6) and (5.7) such a situation cannot arise. Hence we have a contradiction.

- $gate_G(e_1) = \tau$ and $gate_G(e_2) = \tau$

  Here the contradiction arises directly.

□

Note that ports are considered annotated in the domain of gate mappings. The reason for that is to avoid any conflicts of event labels if the component body is a network. In such a situation a many-to-one gate mapping may be employed. Especially, two or more ports of the same label, but of different components, can be included in the domain of the gate mapping. By annotation it is guaranteed that the gated ports (and respective events) have mutually different labels.

If the component body is a leaf then annotation of its events is not necessary. However, to simplify definitions we do not treat this situation differently. As we have stated above (Definition 5.10), we assume a homogeneous 0-annotation of all leaf events in the context of a component body. Moreover, this fact enables us to use the following notation.

Notation 5.16 *Let $S \in$ Leaves a leaf. Further let G a gate of an arbitrary component $C = \langle S, I, G \rangle$ which contains S as its component body. For any event $e^{\sharp_0} \in \alpha(S)$ we use the notation $gate_G(e)$ to abbreviate the application of a gate function to a 0-annotated event in the following way:*

$$gate_G(e) = p \overset{df}{\Leftrightarrow} gate_G(e^{\sharp_0}) = p$$

Such abbreviation is absolutely correct and does not violate any suggestions declared in Section 4.3.3 because the 0-annotation realises nothing more than one-to-one renaming of all the observable leaf events. According to these facts and the requirement 5.4 of definition 5.13, the notion of gate in the case of a leaf component body considers only one-to-one gate mappings of the type $\perp$.

Component

Finally we formalise the *component* — the logical element which groups together component body, interface, and gate. Relationships among these notions have one-to-one character, as it is illustrated in Figure 5.3. Component structure is formally captured in the form of so-called *component structural term.*

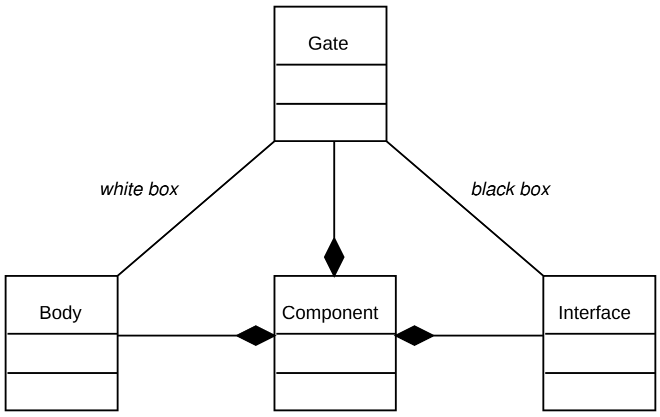


Figure 5.3: Relationships between a component and its parts

Definition 5.17 *Define a* component structural term *C as a tuple $C \overset{\mathrm{df}}{=} \langle S, I, G\rangle$ where*

- *S is a component body.*
- *I is a component interface.*
- *G is a* gate *relating the body S to the interface I.*

Define the white box view of component *C, denoted $wbox(C)$, as the alphabet consisting of events observable in the component body, $wbox(C) \overset{\mathrm{df}}{=} \alpha(S)$.*

Finally, define the black box view of component *C, denoted $bbox(C)$, as the alphabet of events observable on the interface $bbox(C) \overset{\mathrm{df}}{=} gate_G(wbox(C))$.*

Notation 5.18 *Let $\mathbf{CT}_{\mathrm{st}}$ denote the set of all component terms.*

Let $\langle C_1, ..., C_n\rangle$ be a tuple of component terms for some $n \in \mathcal{N}$. Denote $ports(\langle C_1, ..., C_n\rangle) \subset \mathcal{P}^{\sharp}$ the set of all ports of all components of the tuple, $ports(\langle\langle S_1, I_1, G_1\rangle, ..., \langle S_n, I_n, G_n\rangle\rangle) \overset{\mathrm{df}}{=} \bigcup_{i=1}^{n} ports(I_i)^{\sharp_i}$.

For a component $C \equiv \langle S, I, G\rangle$ denote $I(C) \overset{\mathrm{df}}{=} I$ its interface.

5.1.2 Buses and Links

In this subsection, we formalise the notion of buses and links. As it has been mentioned in Section 4.3.5, buses are key elements of network construction, i.e., they make counterparts of components. Relations among all network construction elements are depicted in Figure 5.4.

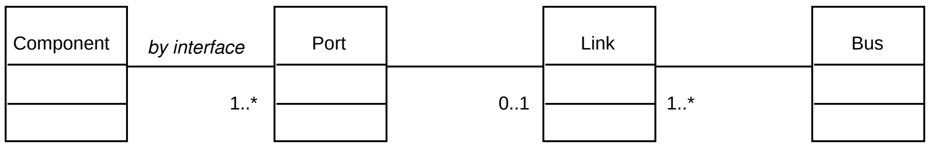


Figure 5.4: Relationships among buses, links, ports and components

First of all, we define the notion of *cooperation*. A cooperation represents a fundamental entity on which the notion of bus is based.

Definition 5.19 *Let $W^\sharp \subset \mathcal{W}^\sharp$ and $R^\sharp \subset \mathcal{R}^\sharp$ finite set of ports. Define* cooperation *as the pair $\langle W^\sharp, R^\sharp\rangle$, denoted $\langle W^\sharp / R^\sharp\rangle$, satisfying:*

$$\forall i, j \in \mathcal{N}.\, {p_1}^{\sharp i}, {p_2}^{\sharp j} \in W^\sharp \cup R^\sharp \Rightarrow i \neq j \qquad (5.8)$$

The sets $W^\sharp$ and $R^\sharp$ are called input *and* output section *of the cooperation, respectively.*

We say that a port p is included in the cooperation $c := \langle W^\sharp / R^\sharp\rangle$*, and write $p \in c$, if either $p \in W^\sharp$ or $p \in R^\sharp$. The set of all cooperations is denoted* Coops, $\mathrm{Coops} \overset{\mathrm{df}}{=} 2^{\mathcal{W}^\sharp}_{\mathrm{fin}} \times 2^{\mathcal{R}^\sharp}_{\mathrm{fin}}$.

In general, either of both sections in the cooperation is allowed to be empty. The condition (5.8) is important to achieve consistency of cooperations and simplifies the definition of semantics (see Section 7.2.5 of Chapter 7). The nature of this property goes with the aim that we would like to avoid of specifications which allow more than one action of a particular component to be coordinated by some cooperation atomically in just one computation step.

The following definition characterizes the notion of buses.

Definition 5.20 *Assume* Buses *denotes a countable set of all buses. Members of* Buses *are typically denoted $B, B_1, B_2, \ldots$*

Let $B \in$ Buses *a bus. Define* set of cooperations characterizing the bus *B, denoted coop(B), as an arbitrary finite set of cooperations, coop$(B) \subset$* Coops, *coop$(B) \neq \emptyset$.*

Input of the bus *B is denoted $In(B)$ and defined $In(B) \overset{\mathrm{df}}{=} \bigcup\{W^\sharp \parallel \langle W^\sharp / R^\sharp\rangle \in coop(B)\}$. Similarly,* output of the bus *B is denoted $Out(B)$ and defined $Out(B) \overset{\mathrm{df}}{=} \bigcup\{R^\sharp \parallel \langle W^\sharp / R^\sharp\rangle \in coop(B)\}$.*

In the following definition we formalize the notion of links.

Definition 5.21 *Define the* link relation L *as the relation* $L \subset \mathcal{P}^\sharp \times \mathrm{Buses}$ *satisfying for each* $B \in \mathrm{Buses}$ *each of the following conditions:*

$$\forall i,j \in \mathcal{N}, {p_1}^{\sharp_i}, {p_2}^{\sharp_j} \in \mathcal{P}^\sharp . \langle {p_1}^{\sharp_i}, B\rangle \in L \wedge \langle {p_2}^{\sharp_j}, B\rangle \in L \Rightarrow p_1 \neq p_2 \vee i \neq j \quad (5.9)$$

$$\begin{aligned} &\forall p \in \mathcal{P}^\sharp .\, \langle p, B\rangle \in L \Rightarrow \exists c \in B.\, p \in c \\ &\forall w \in \mathcal{W}^\sharp .\, w \in In(B) \Rightarrow \langle w, B\rangle \in L \qquad (5.10) \\ &\forall r \in \mathcal{R}^\sharp .\, r \in Out(B) \Rightarrow \langle r, B\rangle \in L \end{aligned}$$

Members of some link relation L *are called* links. *The set of all link relations is denoted* Links. *Furthermore, for any* $B \in \mathrm{Buses}$ *and any* $L \in \mathrm{Links}$ *the set of all its links is denoted* $links(B,L)$ *and defined by the following construction:*

$$links(B,L) \overset{\mathrm{df}}{=} \{l \parallel \exists p \in \mathcal{P}^\sharp .\, l = \langle p, B\rangle \in L\}.$$

The condition (5.9) in the definition above ensures that at most one link can be defined for a particular component interface port. The set of conditions (5.10) guarantees consistency of embedding a particular bus into a particular link relation. More precisely, all the ports linked to a particular bus must be reflected by some cooperation and for each port of any cooperation of a particular bus there must be a link between the bus and that port.

Notation 5.22 *For a particular link* $l \in L$ *of some link relation* L *we denote its port as* $port(l) \overset{\mathrm{df}}{=} p$, $l \equiv \langle p, B\rangle$, *where* $B \in \mathrm{Buses}$. *Moreover, for a given link relation* L *the set of ports of all links of a particular bus* B *is denoted* $ports(B)$, $ports(B) \overset{\mathrm{df}}{=} \{port(l) \parallel l \in links(B,L)\}$.

For a particular bus B*, a component* C *and a link relation* L *we denote* $Llinks(B,C)$ *the set of all links connecting ports of* $I(C)$ *to the bus* B, $Llinks(B,C) \overset{\mathrm{df}}{=} \{l \in L \parallel l \equiv \langle p, B\rangle,\ p \in ports(I(C))\}$.

The key requirement imposed on ports as a precondition of hierarchical connection of ports to a higher-level component interface states that no link can be assigned to lower-level ports involved in such a vertical connection. The following definition treats this property of ports formally.

Definition 5.23 *Let* $C_i = \langle S_i, I_i, G_i\rangle$ *a component term for some* $i > 0$ *and let* $L \in \mathrm{Links}$ *a link relation. We say that the port* $p \in ports(I_i)$ *is a* free port w.r.t. L *if and only if for every bus* $B \in \mathrm{Buses}$ *it holds that* $\langle p^{\sharp_i}, B\rangle \notin L$.

Denote $freeports(C_i, L)$ *the set of all free ports of the component* C_i *w.r.t.* L.

For a tuple of component terms $\langle C_1, C_2, ..., C_n\rangle$ *for some* $n \in \mathcal{N}$, *denote* $freeports(\langle C_1, C_2, ..., C_n\rangle) \subset \mathcal{P}^\sharp$ *the set of free ports of all components in the tuple w.r.t.* L*:*

$$freeports(\langle C_1, C_2, ..., C_n\rangle, L) \overset{\mathrm{df}}{=} \bigcup_{i \in \{1,...,n\}} freeports(C_i, L)^{\sharp_i}$$

5.1.3 Link Ranking

To extent cooperations with an abstract port addressing mechanism, the notion of *link ranking* has been introduced. The goal of link ranking is alternative identification of a port in the context of a network topology. The identification by ranks is totally independent of identification by port labels. This independence is crucial. A link rank determines specific meaning of a port notifying a bus about how the port should be treated in a cooperation. By changing a rank of a particular port link in the design, the way of how the port identifies itself to a bus can be changed. The mechanism of link ranking is used by special bus classes which are sensitive to ranks.

First of all, we define the notion of link ranking formally. Then we introduce some auxiliary notions which enable us to simplify the use of link rankings in further definitions.

Definition 5.24 *Let* $L \in \mathrm{Links}$ *be some link relation. Define the* link ranking *as a function* $lrank : L \rightarrow Rank_{\perp}$ *which assigns a rank to each link in* L*, satisfying the following conditions:*

1. $\forall l \in L.\, lrank(l)$ *is defined*
2. *for any* $B \in \mathrm{Buses}$ *it holds that either*
   - $\forall l \in links(B, L).\, lrank(l) = \perp$
   - *or* $\forall l \in links(B, L).\, lrank(l) \neq \perp$.

Notation 5.25 *For the given link set* L *the set of all link rankings which can be defined for that set is denoted* $lranks(L)$*. The general set of all link rankings is denoted* Lranks.

To simplify the algorithm for construction of bus class instances (Algorithm 6.18 given in Chapter 6), we introduce a function $rank_B$ which for a each port connected to a particular bus B returns the rank of the respective link relation.

Definition 5.26 *For the particular bus* B*, link relation* L*, and link ranking* $lrank$ *we define the function* $rank_B : In(B) \cup Out(B) \rightarrow Rank_{\perp}$ *by the following prescription:*

$$rank_B(p) \overset{\mathrm{df}}{=} lrank(l) \text{ where } l \in L \text{ and } port(l) = p.$$

5.1.4 Networks and Leaves as Structural Terms

Finally we define the language of structural terms which formally represent static syntax of VCN. More specifically, we define terms representing VCN networks and leaves.

Definition 5.27 *For sets of atomic elements* Leaves, Gates, Links, Buses, *and* Lranks, *introduced in previous definitions, define the set of* structural terms $\mathbf{T}_{st}$ *as the least set satisfying:*

1. $A \in \mathbf{T}_{st}$*, where* $A \in$ Leaves, *and it is called a* leaf term.
2. $N \in \mathbf{T}_{st}$*, if* N *is a* network term *defined as a tuple* $N \overset{\text{df}}{=} \langle \bar{C}, \bar{B}, L, lrank \rangle$ *where*
   - $\bar{C} = \langle C_1, \ldots, C_n \rangle$ *for some* $n > 0$ *is a tuple of component terms, for each* $i \in \{1, ..., n\}$*,* $C_i = \langle S_i, I_i, G_i \rangle$ *is a component structural term satisfying:*
     (a) $S_i \in \mathbf{T}_{st}$
     (b) $G_i = \langle map_G, type_G \rangle$ *a gate in which each mapping* $g \in map_G$ *has the signature* $g : obs(S_i) \to ports(I_i)$ *where* $obs(S_i)$ *is defined in the following way:*
       (i) $obs(S_i) \overset{\text{df}}{=} \alpha(S_i)$*, if* $S_i \in$ Leaves;
       (ii) $obs(\langle \bar{C}, \bar{B}, L, lrank \rangle) \overset{\text{df}}{=} \mathcal{E}^\sharp(freeports(\bar{C}, L))$*, otherwise.*
   - $\bar{B} = \langle B_1, \ldots, B_m \rangle$ *for some* $m \geq 0$ *is a tuple of buses*
   - $L \in$ Links *a link relation satisfying*

$$L \subseteq ports(\bar{C}) \times \{B_i \parallel i \in \{1, ..., m\}\}$$

   - $lrank \in lranks(L)$ *is a link ranking.*

Notation 5.28 *Let* $N = \langle \langle C_1, .., C_i, .., C_n \rangle, \langle B_1, ..., B_m \rangle, L, Lrank \rangle$ *a network term and let* $C' \in \mathbf{CT}_{\mathrm{st}}$ *a component. By the notation* $N[C_i := C']$ *denote the network which differs from* N *only in its* i*th component, so that the component* C_i *is replaced with* C'*. Formally,* $N[C_i := C'] \overset{\text{df}}{=} \langle \langle C_1, .., C_{i-1}, C', C_{i+1}, .., C_n \rangle, \langle B_1, .., B_m \rangle, L, Lrank \rangle$*.*

Let $B' \in$ Buses *a bus. By the notation* $N[B_j := B']$ *denote the network term which differs from* N *only in its* j*th bus, so that the bus* B_j *is replaced with* B'*. Formally,* $\mathtt{N}[B_j := B'] \overset{\text{df}}{=} \langle \vec{C}, \langle B_1, .., B_{j-1}, B', B_{j+1}, .., B_m \rangle, L, Lrank \rangle$*.*

Further for a network term N *denote* $L(N)$ *its link relation.*

Let us discuss reasons why to represent links as an independent notion treated at the same level as buses and components (in the scope of network). On one hand, the notion of link is by its nature very closed to the notion of bus, because two different sets of links distinguish any two buses in a network. In consequence, the first idea was to define a link set as a part of the bus declaration. However, the set of all links in a network is naturally very often changed during the system design. In the case of some link addition or deletion, it would be infeasible to search the network structure to find the relevant bus to change its set of links. To avoid that insufficiency we represent links as an explicit notion and treat them explicitly in the scope of a network, i.e., at the same level as components and buses.

To ensure that the definition 5.27 of the set $\mathbf{T}_{st}$ of structural terms is correct, we have to prove that $\mathbf{T}_{st}$ is the least fixed point of the inductive construction from the above definition. To capture the construction of $\mathbf{T}_{st}$, we define a constructor function F, which realises precisely the definition 5.27. Afterwards, we prove that $\mathbf{T}_{st}$ is the least fixed point of F.

Definition 5.29 *Define a* constructor F *of VCN structural terms as a function* $F : 2^{\mathbf{T}_{st}} \to 2^{\mathbf{T}_{st}}$ *satisfying:*

1. $F(\emptyset) \overset{\text{df}}{=} \{A \parallel A \in \text{Leaves}\}$
2. *For* $\mathcal{T} \neq \emptyset$ *define:*

$$F(\mathcal{T}) \overset{\text{df}}{=} (F_N \circ F_C)(\mathcal{T}) \cup \bigcup_{\mathcal{T}' \subset \mathcal{T}} F(\mathcal{T}')$$

where

- $F_C : 2^{\mathbf{T}_{st}} \to 2^{\mathbf{CT}_{\text{st}}}$ *is* component constructor

  For each $\mathcal{T} \subseteq \mathbf{T}_{st}$, $\mathcal{T} \neq \emptyset$*:*

  $$F_C(\mathcal{T}) \overset{\text{df}}{=} \bigcup_{S \in \mathcal{T}} \{\langle S, I, G\rangle \in \mathbf{CT}_{\text{st}} \parallel I \text{ an interface }, G \text{ a gate}\}$$

  where each gate G respects the property (b) *of the definition 5.27.*

- $F_N : 2^{\mathbf{CT}_{\text{st}}} \to 2^{\mathbf{T}_{st}}$ *is* network constructor

  For each $\mathcal{C} \subseteq \mathbf{CT}_{\text{st}}$, $\mathcal{C} \neq \emptyset$, *define:*

  $$F_N(\mathcal{C}) \overset{\text{df}}{=} \bigcup_{\substack{\{C_1,\ldots,C_n\} \subseteq \mathcal{C} \\ n \geq 1}} \{\langle\langle C_1', \ldots, C_m'\rangle, \bar{B}, L, lrank\rangle \parallel m \geq n\}$$

  where

  1. *For each* $i \in \{1, \ldots, m\}$, $C_i' := C_j$, *for some* $j \in \{1, \ldots, n\}$.
  2. *For each* $j \in \{1, \ldots, n\}$ *there exists* $i \in \{1, \ldots, m\}$, $C_i' = C_j$.
  3. $\bar{B}$, L, $lrank$ *are arbitrary sets satisfying the relevant properties stated in definition 5.27.*

Lemma 5.30 *The constructor F is monotonic.*

Proof: Let $\mathcal{T}_1, \mathcal{T}_2 \in \mathbf{T}_{st}$ and assume $\mathcal{T}_1 \subseteq \mathcal{T}_2$.

With respect to the definition of F we have:

$$F(\mathcal{T}_1) \overset{\text{df}}{=} (F_N \circ F_C)(\mathcal{T}_1) \cup \bigcup_{\mathcal{T}' \subset \mathcal{T}_1} F(\mathcal{T}')$$

$$F(\mathcal{T}_2) \overset{\text{df}}{=} (F_N \circ F_C)(\mathcal{T}_2) \cup \bigcup_{\mathcal{T}' \subset \mathcal{T}_2} F(\mathcal{T}')$$

(1) Let $C \in F_C(\mathcal{T}_1)$. From the definition of F_C and the assumption above it follows that $C \in F_C(\mathcal{T}_2)$. Hence, $F_C(\mathcal{T}_1) \subseteq F_C(\mathcal{T}_2)$.

(2) Similarly let $\mathcal{C}_1, \mathcal{C}_2 \subseteq \mathbf{CT}_{\text{st}}$ and assume $\mathcal{C}_1 \subseteq \mathcal{C}_2$.

Let $S \in F_N(\mathcal{C}_1)$. From the definition of F_N and the assumption above it follows that $S \in F_N(\mathcal{C}_2)$. Hence, $F_N(\mathcal{C}_1) \subseteq F_N(\mathcal{C}_2)$.

From (1) and (2) it follows that $F_N(F_C(\mathcal{T}_1)) \subseteq F_N(F_C(\mathcal{T}_2))$.
Note that $\mathcal{T}_1 \subseteq \mathcal{T}_2 \Rightarrow \bigcup_{\substack{\mathcal{T}' \subset \mathcal{T}_1 \\ \mathcal{T}' \neq \emptyset}} F(\mathcal{T}') \subseteq \bigcup_{\substack{\mathcal{T}' \subset \mathcal{T}_2 \\ \mathcal{T}' \neq \emptyset}} F(\mathcal{T}')$.
Hence, $F(\mathcal{T}_1) \subseteq F(\mathcal{T}_2)$, and therefore, F is monotonic.

Lemma 5.31 *The constructor F is continuous.*

Proof: From lemma 5.30 we know F is monotonic. Additionally, we have to prove that F preserves least upper bounds of infinite sequences.

Let $\mathcal{T}_1, \mathcal{T}_2, \ldots$ be an (infinite) sequence satisfying $\forall i \in \mathcal{N}.\ \mathcal{T}_i \subseteq 2^{\mathbf{T}_{st}} \wedge \mathcal{T}_i \subseteq \mathcal{T}_{i+1}$. Applying associativity of the set union operation the following equations are derived:

$$\bigcup_{i \in \mathcal{N}} F(\mathcal{T}_i) = \bigcup_{i \in \mathcal{N}} (F_N(F_C(\mathcal{T}_i)) \cup \bigcup_{\mathcal{T}' \subset \mathcal{T}_i} F(\mathcal{T}')) = \bigcup_{i \in \mathcal{N}} F_N(F_C(\mathcal{T}_i)) \cup \bigcup_{\mathcal{T}' \subset \bigcup_i \mathcal{T}_i} F(\mathcal{T}')$$

$$F(\bigcup_{i \in \mathcal{N}} \mathcal{T}_i) = F_N(F_C(\bigcup_{i \in \mathcal{N}} \mathcal{T}_i)) \cup \bigcup_{\mathcal{T}' \subset \mathcal{T}_i} F(\mathcal{T}')) = F_N(F_C(\bigcup_{i \in \mathcal{N}} \mathcal{T}_i)) \cup \bigcup_{\mathcal{T}' \subset \bigcup_i \mathcal{T}_i} F(\mathcal{T}')$$

Now it remains to be proved that the composition $F_N \circ F_C$ is continuous.

$$F_C(\bigcup_{i \in \mathcal{N}} \mathcal{T}_i) = \bigcup_{S \in \bigcup_i \mathcal{T}_i} \{\langle S, I, G \rangle \in \mathbf{CT}_{\text{st}} \parallel I \text{ an interface }, G \text{ a gate}\}$$

$$\bigcup_{i \in \mathcal{N}} F_C(\mathcal{T}_i) = \bigcup_{i \in \mathcal{N}} \bigcup_{S \in \mathcal{T}_i} \{\langle S, I, G \rangle \in \mathbf{CT}_{\text{st}} \parallel I \text{ an interface }, G \text{ a gate}\}$$

Associativity of the set union leads to the following equation:

$$F_C(\bigcup_{i \in \mathcal{N}} \mathcal{T}_i) = \bigcup_{i \in \mathcal{N}} F_C(\mathcal{T}_i)$$

Following equations are derived by application of definition of F_N:

$$F_N(F_C(\bigcup_{i \in \mathcal{N}} \mathcal{T}_i)) = \bigcup_{\substack{\{C_1,...,C_n\} \subseteq F_C(\bigcup_i \mathcal{T}_i) \\ n \geq 1}} \{\langle\langle C'_1, ..., C'_m\rangle, \bar{B}, L, lrank\rangle \parallel m \geq n\}$$

$$\bigcup_{i \in \mathcal{N}} F_N(F_C(\mathcal{T}_i)) = \bigcup_{i \in \mathcal{N}} \bigcup_{\substack{\{C_1,...,C_n\} \subseteq F_C(\mathcal{T}_i) \\ n \geq 1}} \{\langle\langle C'_1, ..., C'_m\rangle, \bar{B}, L, lrank\rangle \parallel m \geq n\}$$

Finally, from associativity of the set union we get continuity of $F_N \circ F_C$:

$$F_N(F_C(\bigcup_{i \in \mathcal{N}} \mathcal{T}_i)) = \bigcup_{i \in \mathcal{N}} F_N(F_C(\mathcal{T}_i))$$

Hence, we have proved that F is continuous.

Theorem 5.32 *The set of structural terms $\mathbf{T}_{st}$, defined as the least set satisfying properties stated in definition 5.27, is the least fixed point of the constructor F.*

Proof: The set $2^{\mathbf{T}_{st}}$ ordered by the set inclusion relation $\subseteq$ is a complete lattice. Therefore, applying the Kleene theorem, there exists a least fixed point μF of the monotonic and continuous function $F : 2^{\mathbf{T}_{st}} \to 2^{\mathbf{T}_{st}}$:

$$\mu F \overset{\text{df}}{=} \bigcup_{i \in \mathcal{N}} F^i(\emptyset)$$

Note that the set $\mathbf{T}_{st}$ has been defined as the least set satisfying the properties in definition 5.27. By careful studying of the definition of the constructor F, we argue that F is a recursive function which corresponds precisely to the inductive construction presented in definition 5.27. Hence, $\mu F = \mathbf{T}_{st}$.

Corollary 5.33 *The inductive definition 5.27 of structural terms is correct.*

5.2 Additional Notes

In this chapter we have defined a precise formal base for the VCN visual notation. Before definitions of the coordination and behavioural models of VCN diagrams which will be presented in next chapters, we discuss the notion of abstractness which is allowed by the definition of structural terms.

Let us recall the reason why we introduced structural terms, in particular, giving unambiguous representation to the VCN visual notation. In Definition 5.27, it is abstracted from detail meaning of leaves and buses. In other words, members of sets Leaves and Buses have been defined as purely syntactic notions. In contrary, elementary VCN notions of gates, interfaces, links, and ranks have been defined with the concrete meaning given in terms of sets (Definition 5.8), relations (Definition 5.21), or functions (Definition 5.13 and Definition 5.24). We say that all those notions are *implicit* in VCN.

However, a fixed predefined meaning of leaves, buses, and gate types is not in charge of definition of structural terms. Therefore, we say that these elements are *explicit* in VCN. To establish concrete semantics for all structural terms, the meaning of these explicit elements has to be provided at first. Such a property is introduced to keep the VCN visual notation abstract and independent of its behavioural semantics.

The coordination model and the behavioural model which are defined in next chapters give the structural terms a behavioural semantics based on the notion of state-transition systems. Behavioural semantics of buses is defined in Section 6.1 of Chapter 6. Behavioural model of leaves is given in Section 7.2.1 of Chapter 7. In Section 7.2.3 of that chapter, both the behavioural semantics of buses and the behavioural model of leaves are combined in order to establish a behavioural model of entire VCN architectures. This is achieved in Section 7.2.4 by inference rules which define the semantics of component decomposition (i.e., behaviour of individual types of gate mappings) and the semantics of inner-level component coordination (i.e., mutual interconnection of components and buses).

The behavioural model defined in this thesis is computation-oriented (like CCS and CSP) and does not consider the notion of roles introduced in this chapter. We leave for future research the development of an interface-oriented model (like Wright or Interface Automata [dAH01a]) for VCN which would employ the notion of roles. The relation between both approaches of the behavioural model definition is nicely explained in [dAH01b]. The basic idea behind the interface-oriented model is based on refining each component interface with a state-transition semantics.

Chapter 6

VCN: Coordination Model

The coordination model in VCN is identified by the notions of buses and bus classes. In Chapter 4, the notion of bus classes and their instances (particular buses) has been introduced informally, and in Chapter 5, the notions of cooperations and buses have been formally defined. In this chapter, we assign a state-transition semantics to buses (Section 6.1), and subsequently, we focus on the notion of bus classes precisely (Section 6.2). Especially, we define the bus class specification language. In consequence, we show how bus instances are generated from bus classes.

A proposal of the research provided in this chapter has been previously published in [SS05].

6.1 Semantics of Bus Instances

As it has been proposed in Chapter 4, the intended meaning of a particular bus $B \in \mathrm{Buses}$ included in a network is to represent a specific coordination model for components which together with the bus make a particular part of the network – a so-called *star topology*. To that end, we have defined in Chapter 5 the set of cooperations $coop(B)$ which characterises the bus B concerning all its possible cooperations. In this section, we refine the set of cooperations with a behavioural model in terms of a transition system.

The overall intuition about such notion of behavioural model of buses is illustrated by an example in Figure 6.1, where a model of a weather condition information system is depicted. The model is represented as a VCN network with four components $Temp\_sensor$, $Humi\_sensor$, which represent weather conditions sensors, the $Display$ component representing the LCD panel reporting the current weather information, and the $Switch$ component which initiates entire system computation. The coordination model of these components is determined by the bus $COORD$. The meaning of this bus is coordination behaviour which combines atomic broadcast, synchronous channel, inhibitor, and a one-place buffer. Note that

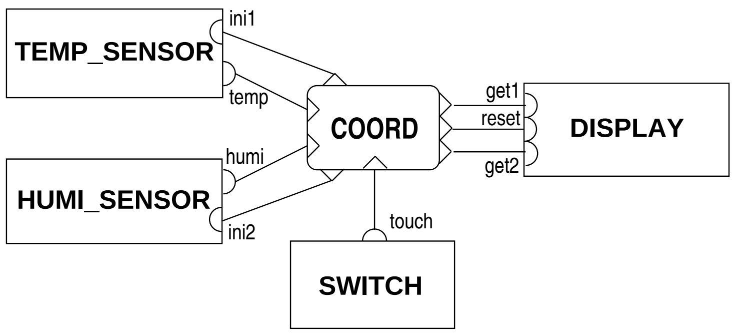


Figure 6.1: Behavioural model of a bus

the framework for capturing of the three former (stateless) coordination mechanisms has been already defined in the previous chapter (list of cooperations). However, that framework does not suffice for definition of the latter mechanism, because of the requirement of some state-transition control. The overall principle of the coordination model determined by the bus $COORD$ in this particular example can be summarised in terms of the following phases:

1. In the initial phase, pushing of the $touch$ trigger causes broadcasting of an initiation signal through the $ini1, ini2$, and $reset$ ports to the corresponding components. This broadcasting is performed atomically in an indivisible time instant to ensure quick and uninterruptible reaction of the system to the initiation signal.

2. After initiation, the coordination model is waiting for the information to be signalled on the producers ($Temp\_sensor$ and $Humi\_sensor$) output ports. In this phase, it is also capable of receiving the $touch$ signal, to preserve blocking of the initiation switch. However, this signal is lost whenever it is transmitted (initiation cannot be performed in this phase).

   When the $temp$ and $humi$ information appears, the coordination model acts like both a synchronous channel (relaying the $temp$ information to the $get1$ port) and a one-place buffer (storing the $humi$ information to the internal memory).

3. After filling of the buffer, the information stored in the buffer is transmitted to the $get2$ port. It ensures that the $get1$ and $get2$ ports are filled with the appropriate information in a sequence of a given order (i.e., the information of the current weather conditions can appear always in the predefined order on the display panel).

4. The coordination model returns to the initial phase (the phase (1)).

The coordination behaviour described above can be formally captured by a labelled transition system with each of its transition labels defined as a particular cooperation. For the above mentioned example, the transition system is depicted in Figure 6.2. To illustrate the relation between the transition system and the description above, numbers of the specific coordination phases are written inside the circles representing states.

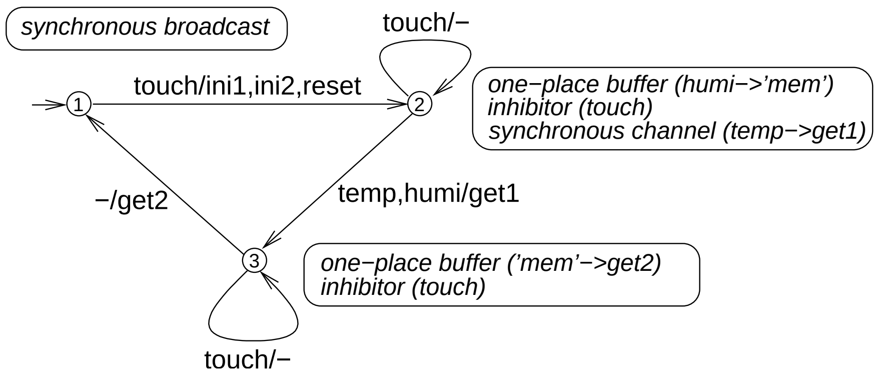


Figure 6.2: Transition system representing the $COORD$ bus behaviour

We call such a variant of transition system *coordination machine*. The cooperation machine is declared formally by the following definition.

Definition 6.1 *Let* $B \in$ Buses *a bus. A* cooperation machine *of* B, *denoted* $cm(B)$, *is a finite labelled transition system defined as* $cm(B) \overset{\text{df}}{=} \langle Q, coop(B), T, q_0 \rangle$.

The set of all cooperation machines is denoted CMS.

Notation 6.2 *As the alphabet of a particular coordination machine* $cm(B)$ *is given by the definition of the relevant bus* B, *we will usually abbreviate the cooperation machine simply as a triple* $\langle Q, T, q_0 \rangle$ *instead of a quadruple* $\langle Q, coop(B), T, q_0 \rangle$.

For a given bus $B \in$ Buses, *the set of all states of the cooperation machine* $cm(B)$ *is denoted* $Q(B)$. *Further, the initial state of the cooperation machine* $cm(B)$ *is denoted* $\Phi_B(B)$.

The fact that $\langle q, \langle W/R \rangle, q' \rangle \in T$ *for some* $q, q' \in Q(B)$ *and* $\langle W/R \rangle \in coop(B)$ *is denoted* $q \overset{W/R}{\to}_B q'$.

Additionally, the set of all transitions which the bus can perform in some state $q \in Q(B)$ *is denoted* $en(q)$, $en(q) \overset{\text{df}}{=} \{\langle W/R \rangle \in coop(B) \parallel \exists q' \in Q(B).\, q \overset{W/R}{\to}_B q'\}$.

Note 6.3 *The finiteness requirement of the cooperation machine is imposed to enable automatised analyses of VCN networks. This requirement deals exclusively* *with the set of states, finiteness of the transition relation T is then automatically achieved due to the finiteness of the set of cooperations characterising a particular bus.*

It is worth emphasising some principal properties of the cooperation machine approach of coordination model specification. In contrast to the notion of a common transition system, in which a single discrete event fortifies the level of its atomicity, the most characteristic property of this state transition system variant is lifting of the atomicity to a set of events occurring together in a single discrete uninterruptible coordination action (cooperation).

6.2 Bus Class Specification Language

Bus classes have the purpose of parametrised templates from which particular buses (bus instances) are generated for given parameter values. Each bus class is written by the notation $\texttt{bus\_class\_name}(parameters)$ and determined by its *type*, *parameter constraint*, and a *bus predicate*. Specification of a particular bus class $\mathcal{B}$ has the following structure:

$$\mathcal{B}(parameters) := \{$$
$$\bullet\ parameter\_constraint$$
$$\bullet\ bus\_predicate$$
$$\}$$

Parameters of the bus class can be the following:

- `In` ... set of bus inputs (compulsory)
- `Out` ... set of bus outputs (compulsory)
- `rank` ... link ranking of the bus inputs and outputs (optional)
- `capacity` ... capacity of the bus internal memory (optional)

With respect to the type – the particular setting of the parameters, four kinds of bus classes are distinguished.

1. $\mathcal{B}(\texttt{In}, \texttt{Out})$ denotes a *rank-free bus class with no associated memory*. This is the most trivial kind of bus classes which is suitable for specification of basic synchronous coordination models.
2. $\mathcal{B}(\texttt{In}, \texttt{Out}, \texttt{rank})$ denotes a *ranked bus class with no associated memory*. This is a more general kind of bus class which is suitable for specification of synchronous coordination models where addressing of inputs and outputs is necessary. More particularly, predicates in this kind of bus classes can reason about ranks of individual inputs and outputs and relate them with each other.

3. $\mathcal{B}(\mathtt{In}, \mathtt{Out}, \mathtt{capacity})$ represents a *rank-free bus class equipped with memory*. The memory is organised in an arbitrary number of buffers all having the capacity determined by the last parameter value. This kind of bus class allows specification of elementary asynchronous coordination models.

4. $\mathcal{B}(\mathtt{In}, \mathtt{Out}, \mathtt{rank}, \mathtt{capacity})$ denotes the most expressive kind of bus classes – a *ranked bus class equipped with memory*. Here both addressing of inputs and outputs and presence of memory are allowed. This feature implies using of these bus classes for specification of asynchronous coordination models where distinguishing among individual inputs and outputs is necessary.

The relative expressiveness of all bus class types is depicted in Figure 6.3. The least expressive bus class type is depicted at the bottom of the lattice. All the relationships are strict. Formal arguments explaining the reasons why the lattice is correct will be given at the end of this section (from the trivial purely syntactic point of view) and in the next chapter (from the more complicated semantic point of view).

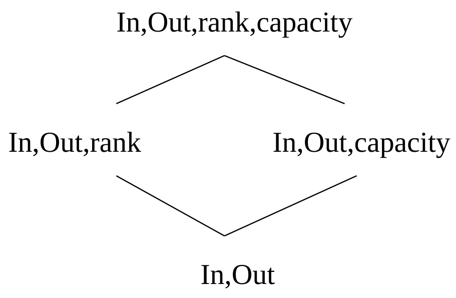


Figure 6.3: Expressiveness relationships among bus class types

Parameter Constraint

The parameter constraint specifies what exact parameter values are acceptable by the bus class. In particular, it can be any variable and quantifier-free second-order predicate logic formula with constants `In`, `Out`, and `capacity` representing the respective parameters. Constants `In` and `Out` are interpreted as finite sets of ports of the respective kind. The constant `rank` is interpreted as a finite unary function assigning a rank to particular ports (see Definition 5.24). Finally, the `capacity` constant is interpreted as a nonzero natural number denoting the capacity of the bus memory. The three former-most constants are second-ordered, but always limited to interpret only finite objects. This fact enables us to enrich the parameter constraints with the predicate $card$ that reasons about cardinality of its parameter (which can be one of the constants above).

Example 6.4 *An example of a parameter constraint is the formula:*

$$\texttt{In} \neq \emptyset \wedge \texttt{Out} \neq \emptyset \wedge card(\texttt{In}) = 1 \wedge \texttt{rank} \neq \emptyset \wedge \texttt{capacity} <= 5$$

Such a formula limits the particular instances of the respective bus class to have at least one input and just one output, further requires a link ranking to be applied, and finally limits the capacity of the memory to store maximally 5 *entries.*

The purpose of a bus constraint is to state requirements about the parameter values for which a bus instance can be constructed. This way, the bus constraint compactly communicate to the user information about the conditions under which the particular bus class can be used, on the one hand. It can also avoid from unnecessary execution of bus instance construction in cases when the bus class is being misused, on the other hand.

In general, from what has been stated above we feel a need for an algorithmic procedure deciding weather a given bus constraint is satisfied or not for a given parameter interpretation. For this purpose we restricted bus constraints to quantifier-free formulae. However, we impose no other specific syntactic restrictions. In this thesis we do not discuss details of such a decision procedure. For our purposes it suffices to know that such a decision procedure exists. But as no quantifiers are allowed and as all the constants are interpreted in domains of finite objects, such a procedure surely exists.

Bus Predicate

Next we focus on the most crucial part of the bus class specification — the bus predicate. Bus predicate is the heart of each bus class. It is responsible for specification of bus cooperations and the cooperation machine which controls them. For a particular bus class, all the cooperations of each bus instance have to satisfy the conditions specified in the bus predicate. To capture the syntax of bus predicates, at first we define the *bus specification language.*

Definition 6.5 *Let* Var_{in} *and* Var_{out} *be the countable sets of bus input variables, and bus output variables, respectively. Additionally assume* Var_{in} *and* Var_{out} *to be mutually disjoint. Furthermore, let Loc be the countable set of location labels and assume* $Loc \cap \mathcal{N} = \emptyset$*. The* bus specification language*, denoted* $\mathbf{BCT}$*, is defined as a set of well-founded finite terms given by the following grammar:*

- input handle

  $\texttt{I} ::= \texttt{i} \parallel \texttt{i}{\uparrow}\texttt{k} \parallel \texttt{i}{\Uparrow}\texttt{k} \parallel \texttt{i}{\uparrow}@ \parallel \texttt{i}{\Uparrow}@$
  where $\texttt{k} \in$ *Loc a memory location label,*
  and $\texttt{i} \in Var_{\text{in}}$ *a bus input variable*

- output handle

  $\texttt{O} ::= \texttt{o} \parallel \texttt{o}{\uparrow}\texttt{k} \parallel \texttt{o}{\downarrow}\texttt{k}$
  $\parallel \{\texttt{o}_1, ..., \texttt{o}_\texttt{n}\}{\uparrow}\texttt{k} \parallel \{\texttt{o}_1, ..., \texttt{o}_\texttt{n}\}{\downarrow}\texttt{k}$
  $\parallel \texttt{o}{\uparrow}@ \parallel \texttt{o}{\downarrow}@$
  $\parallel ?\texttt{k}$
  where $\texttt{k} \in$ *Loc a memory location label,*
  and $\texttt{o}, \texttt{o}_1, ..., \texttt{o}_\texttt{n} \in Var_{\text{out}}$ *bus output variables for some* $\texttt{n} \in \mathcal{N}$

- input and output handles list

  $list\texttt{I} ::= - \parallel \texttt{I},list\texttt{I}$
  $list\texttt{O} ::= - \parallel \texttt{O},list\texttt{O}$

- input section

  $sec\texttt{I} ::= list\texttt{I} \parallel \texttt{all}$

- output section

  $sec\texttt{O} ::= list\texttt{O} \parallel \texttt{all}$
  $\parallel \texttt{all}{\uparrow}\texttt{k} \parallel \texttt{all}{\downarrow}\texttt{k}$
  where $\texttt{k} \in$ *Loc a memory location label*

- cooperation generator

  $\texttt{C} ::= sec\texttt{I}/sec\texttt{O} \parallel sec\texttt{I}{\lhd}/sec\texttt{O} \parallel sec\texttt{I}/{\rhd}sec\texttt{O} \parallel sec\texttt{I}{\lhd}/{\rhd}sec\texttt{O}$

- rank constraint

  $\texttt{R} ::= \texttt{rank}(\texttt{x}) = \texttt{rank}(\texttt{y}) \parallel \texttt{rank}(\texttt{x}) = n$
  $\parallel \neg\texttt{R} \parallel \texttt{R} \wedge \texttt{R}$
  where $n \in \mathcal{N}$ *a rank constant*
  and $\texttt{x}, \texttt{y} \in Var_{\text{in}} \cup Var_{\text{out}}$ *bus input or output variables*

- cooperation pattern

  $\texttt{P} ::= \texttt{C} \parallel \texttt{C}, \texttt{R}$

- bus predicate

  $\mathtt{B} ::= \mathtt{P} \parallel \mathtt{B} \vee \mathtt{B}$

We say that input and output handles of the form $\mathtt{i}{\uparrow}\mathtt{k}$, $\mathtt{i}{\Uparrow}\mathtt{k}$, $\mathtt{i}{\uparrow}@$, $\mathtt{i}{\Uparrow}@$, $\mathtt{o}{\uparrow}\mathtt{k}$, $\mathtt{o}{\downarrow}\mathtt{k}$, $\mathtt{o}{\uparrow}@$, $\mathtt{o}{\downarrow}@$, $\{\mathtt{o}_1, ..., \mathtt{o}_n\}{\uparrow}\mathtt{k}$, $\{\mathtt{o}_1, ..., \mathtt{o}_n\}{\downarrow}\mathtt{k}$, *and* $?\mathtt{k}$ *are* memory input (resp. output) handles. *Handles* $\mathtt{i}{\uparrow}@$, $\mathtt{i}{\Uparrow}@$, $\mathtt{o}{\uparrow}@$, *and* $\mathtt{o}{\downarrow}@$ *are additionally called* rank-bounded memory input (resp. output) handles. *The identifier '@' and the location label $k \in Loc$ occurring in the above mentioned handles are called* location identifiers. *The output handles of the form* $\{\mathtt{o}_1, ..., \mathtt{o}_n\}{\uparrow}\mathtt{k}$ *and* $\{\mathtt{o}_1, ..., \mathtt{o}_n\}{\downarrow}\mathtt{k}$ *are called* multi-handles. *For* $\mathtt{n} = 0$ *these multi-handles together with the handle* $?\mathtt{k}$ *are called* tests.

The operator $\lhd/\rhd$ *is called* universal unfolding. *The operators* $\lhd/$ *and* $/\rhd$ *are called* left *and* right universal unfolding, *respectively.*

A term $\mathtt{B}$ *is* well-defined *iff it satisfies the following constraints:*

1. *First of all we require all input and output variables appearing in arbitrary cooperation pattern to be mutually different. This requirement ensures correct treatment of cooperation patterns which are intended to be interpreted as sets of cooperations. In such an interpretation, any input or output variable appearing in the context of some handle has to represent particular ports included in cooperations generated from the pattern. As each section of a cooperation is a set, also interpretations of cooperation generator sections must be sets. In the context of a particular section, each port generated for any input handle (resp. output handle) must be unique.*

2. *The meaning of a memory input handle is the instruction of storing of an input value (in the non-value passing model this means the information that the respective event occurred) into the memory buffer addressed by the respective location identifier. Hence multiple occurrences of a particular location identifier in the input section have no sense. The only exception is the identifier '@' which denotes the memory location number to be equal to the actual rank of the respective input. Thus we require each well-defined term to include only generators with memory input handles containing mutually different location identifiers or the identifier '@'.*

3. *In similar way, syntax of output handles is also restricted. Each cooperation generator which contains an input handle of the form* $\mathtt{i}{\uparrow}k$ *or* $\mathtt{i}{\Uparrow}k$*, where* $k \in Loc$ *is some particular memory location, is not allowed to contain this location in its output section. In other words, all memory location labels appearing in the output section of a generator must be different from the location labels introduced in the input section of that generator, and moreover, they also must be mutually different. This requirement ensures avoidance of specifications of the form* $\mathtt{i}{\uparrow}k/\mathtt{o}{\downarrow}k$ *in which the possible meaning can be unclear (there appears a question concerning such a cooperation generator*

that asks for which of the two operations should precede the other — the input or the output?). Note that also the meaningless specifications of the form $-/\mathtt{o}_1{\downarrow}\mathtt{k}, \mathtt{o}_2{\uparrow}\mathtt{k}$ *are excluded by this requirement.*

4. *To enable definition of a unique semantics for each bus specification, we have to enforce another requirement concerning the syntax of cooperation generators. This requirement goes with the intuition that it has sense to output data only from such a memory location to which a data can be stored. For location variables and constants we can achieve this restriction at the syntactic level – we require that for each memory location label* $\mathtt{k} \in Loc$ *occurring in some output handle there exists a cooperation pattern with a generator which contains an input handle referring the location* $\mathtt{k}$.

   However, to achieve correct treatment of rank-bounded memory output handles, the syntactic requirement above cannot be applied. The reason for that is dependence on the exact values for which a particular bus instance is generated from a bus class specification. Correctness of using of the rank-bounded memory output handles must be achieved at the semantics level (see Definition 6.14).

5. *This requirement concerns the syntax of rank constraints. More specifically, we require each rank constraint of each cooperation pattern of the form* $\mathtt{C}, \mathtt{R}$ *to contain only the bus input and the bus output variables which occur in* $\mathtt{C}$.*By such a requirement, the well-definedness of all rank constraints is achieved.*

6. *Finally, in each subterm of the form* $sec\mathtt{I}\lhd/sec\mathtt{O}$ *the input section is required to be either* `all` *or contain no memory input handles. Similarly, the output section of each subterm of the form* $sec\mathtt{I}/\rhd sec\mathtt{O}$ *is required to contain no memory location reference. Subterms of the form* $sec\mathtt{I}\lhd/\rhd sec\mathtt{O}$ *are required to satisfy both of the requirements above. By this requirement, arguments of highly succinct operators* $\lhd/$, $/\rhd$, *and* $\lhd/\rhd$ *are restricted. The respective restrictions permits the universal cooperation unfolding to be applied only to a memory-less section. We believe that this restriction simplifies the comprehension and use of the bus specification language.*

Note 6.6

Another requirement concerning link ranking should be considered in relation with rank-bounded memory handles. In particular, the intended meaning of a link ranking in such a situation is associating a memory location with each rank-bounded handle contained in a generator. To satisfy the need of mutually different memory locations stated in requirements (2) and (3) of the previous definition, we require a ranking to mark each input and output variable in the generator with a unique number. However, such a requirement cannot be realised at the syntactic level and hence must be considered at the semantics level.

By the definition above the language for description of bus predicates has been established. Interpretation of bus predicates is realised in the domain of a specific kind of labelled transition systems. Instances of bus specification — particular buses — are given just in terms of such interpretation.

At the beginning of this section we classified bus specifications into types reflecting the setting of specification parameters allowed. The meaning of specification parameters is such that particular values of specification parameters determine interpretation of bus specification language identifiers in the following way:

- `In` — determines interpretation for all bus input variables Var_{in},
- `Out` — determines interpretation for all bus output variables Var_{out},
- `rank` — determines interpretation for the `rank` operator,
- `capacity` — determines interpretation of all the binary operators in the syntax of memory handles ($\uparrow$,$\Uparrow$,$\downarrow$).

In the following definition we characterize each bus class type by a respective sublanguage of $\mathbf{BST}$.

Definition 6.7

- *A* rank-free memory-less bus class *is defined by arbitrary terms of the bus specification language denoted* $\mathbf{BST}_{\mathtt{In},\mathtt{Out}}$ *which is defined as a sublanguage of* $\mathbf{BST}$ *satisfying the following restrictions:*
  - *Input and output handles identifiers are restricted to include no memory handles (the operators* $\uparrow$, $\Uparrow$, $\downarrow$, *and* ? *are excluded from the signature of the language).*
  - *Output section is restricted to the grammar:*

    $$sec\mathtt{O} ::= list\mathtt{O} \parallel \mathtt{all}$$

  - *Rank constraint is completely excluded from the language grammar, hence cooperation patterns have the form:*

    $$\mathtt{P} := \mathtt{C}$$

  - *There are no other restrictions considered.*
- *A* rank-sensitive memory-less bus class *is defined by an arbitrary term of the language denoted* $\mathbf{BST}_{\mathtt{In},\mathtt{Out},\mathtt{rank}}$ *and defined as the sublanguage of* $\mathbf{BST}$ *which corresponds to the language* $\mathbf{BST}_{\mathtt{In},\mathtt{Out}}$ *additionally extended in the following way:*
  - *The full rank constraint identifier* `R` *of* $\mathbf{BST}$ *is added.*

- *Cooperation patterns are extended to:*

$$\mathtt{P} := \mathtt{C} \mid\mid \mathtt{C}, \mathtt{R}$$

- *There are no other extensions considered.*

- *A* rank-free bus class with memory *is defined by an arbitrary term of the language denoted* $\mathbf{BST}_{\mathtt{In},\mathtt{Out},\mathtt{capacity}}$ *and defined as the sublanguage of* $\mathbf{BST}$ *which corresponds to the language* $\mathbf{BST}_{\mathtt{In},\mathtt{Out}}$ *additionally extended in the following way:*
  - *Input and output handles include all memory handles of* $\mathbf{BST}$ *with the only exception of rank-bounded handles (handles of the form* $\mathtt{i}\uparrow @$, $\mathtt{i}\Uparrow @$, $\mathtt{o}\uparrow @$, *and* $\mathtt{o}\downarrow @$*).*
  - *Output section is completely defined by the* $sec\mathtt{O}$ *identifier of* $\mathbf{BST}$.
  - *There are no other extensions considered.*

- *A* rank-sensitive bus class with memory *is defined by an arbitrary term of the entire bus specification language* $\mathbf{BST}$*. To emphasise the fact that all the bus specification features are included in the language we denote the language* $\mathbf{BST}$ *alternatively as* $\mathbf{BST}_{\mathtt{In},\mathtt{Out},\mathtt{rank},\mathtt{capacity}}$.

The following claim gives the lattice depicted in Figure 6.3 the meaning from the syntactic point of view.

Claim 6.8

1. *The language* $\mathbf{BST}_{\mathtt{In},\mathtt{Out},\mathtt{rank},\mathtt{capacity}}$ *is a strict sup-language of the languages* $\mathbf{BST}_{\mathtt{In},\mathtt{Out},\mathtt{rank}}$ *and* $\mathbf{BST}_{\mathtt{In},\mathtt{Out},\mathtt{capacity}}$.
2. *The language* $\mathbf{BST}_{\mathtt{In},\mathtt{Out}}$ *is a strict sub-language of the languages* $\mathbf{BST}_{\mathtt{In},\mathtt{Out},\mathtt{rank}}$ *and* $\mathbf{BST}_{\mathtt{In},\mathtt{Out},\mathtt{capacity}}$.
3. *The languages* $\mathbf{BST}_{\mathtt{In},\mathtt{Out},\mathtt{rank}}$ *and* $\mathbf{BST}_{\mathtt{In},\mathtt{Out},\mathtt{rank},\mathtt{capacity}}$ *are syntactically incomparable.*

Proof: All the statements in the claim follow directly from Definition 6.7.

6.2.1 Semantics of Bus Classes

In the previous subsection, the language $\mathbf{BST}$ for specification of bus classes has been established. In this subsection, we show how particular bus instances (also called buses or models of bus classes) are generated from such specifications. In particular, we establish an algorithm which assigns a set of cooperations $coop(B)$ and a cooperation machine $cm(B)$ to a given bus B according to a bus class specification $\mathcal{B}(\mathtt{In}, \mathtt{Out}, [\mathtt{rank}, \mathtt{capacity}])$ and particular parameter values (square brackets denote the optional presence of respective parameters). In other words, we generate a model of a bus class specification. Such a model is defined as a cooperation machine satisfying the bus class specification interpreted for particular parameter values. More precisely, for a given bus B determined by its inputs $In(B)$, outputs $Out(B)$, their ranking $rank_B$ (if required), and some capacity $\kappa \in \mathcal{N}$ (if required), the respective model is defined by the assignment $cm(B) ::= \mathcal{B}(In(B), Out(B), [rank_B, \kappa])$. An algorithm of how such a model is constructed is the main result of this section.

Parameter and Variable Interpretations

Before we present the algorithm for generation of bus class instances, we have to determine interpretation of bus specification variables and parameters. As it has been intuitively stated in the previous paragraph, bus class parameters `In` and `Out` are interpreted as finite sets of output and input ports, respectively. The $rank$ parameter is interpreted as a function assigning to each member of these finite sets a rank, and thus depends on exact values of the parameters `In` and `Out`. The capacity parameter is interpreted as a nonzero natural number setting the capacity of the bus memory. In the following definition the interpretation of bus class parameters is introduced formally.

Notation 6.9 *The notation* $2^{\mathcal{W}^\sharp}_{\text{fin}}$ *(resp.* $2^{\mathcal{R}^\sharp}_{\text{fin}}$*) denotes the set of all finite subsets of the infinite set of output (resp. input) annotated ports.*

Definition 6.10 *Define* interpretation of bus class parameters *as an arbitrary injective function* $\mathcal{V}_p$ *defined over the domain* $\{\texttt{In}, \texttt{Out}, \texttt{rank}, \texttt{capacity}\}$ *with the range satisfying the following conditions:*

- $\mathcal{V}_p(\texttt{In}) \in 2^{\mathcal{W}^\sharp}_{\text{fin}}$
- $\mathcal{V}_p(\texttt{Out}) \in 2^{\mathcal{R}^\sharp}_{\text{fin}}$
- $\forall x \in \mathcal{V}_p(\texttt{In}) \cup \mathcal{V}_p(\texttt{Out}).\, \mathcal{V}_p(\texttt{rank})(x) \in \mathcal{N}$
- $\mathcal{V}_p(\texttt{capacity}) \in \mathcal{N}$ *such that* $\mathcal{V}_p(\texttt{capacity}) > 0$

Note 6.11 *Each parameter interpretation of a rank-free or a memory-less bus class is supposed to have the domain restricted to the respective subset of parameters.*

Next we define interpretation of variables which occur in bus predicates. Note that according to Definition 6.5 from the previous chapter, three different kinds of variables are distinguished. Hence the interpretation reflects the nature of all those variable kinds.

Definition 6.12 *Define* interpretation of bus predicate variables *as an injective function $\mathcal{V}_v : Var_{\mathrm{in}} \cup Var_{\mathrm{out}} \to \mathcal{W}^\sharp \cup \mathcal{R}^\sharp$ satisfying the following conjunction of requirements:*

$$\forall \mathtt{x} \in dom(\mathcal{V}_v).\, \mathtt{x} \in Var_{\mathrm{in}} \Rightarrow \mathcal{V}_v(\mathtt{x}) \in \mathcal{W}^\sharp \wedge \mathtt{x} \in Var_{\mathrm{out}} \Rightarrow \mathcal{V}_v(\mathtt{x}) \in \mathcal{R}^\sharp$$

Further let $sec\mathtt{I}$ be an input section of some cooperation pattern and $\mathcal{V}_p$ a bus class parameter interpretation. We denote by $\mathcal{V}_v(sec\mathtt{I})$ the variable interpretation of the input section $sec\mathtt{I}$ defined in the following way:

- $\mathcal{V}_v(sec\mathtt{I}) \overset{\mathrm{df}}{=} \emptyset$, *if $sec\mathtt{I} \equiv$ '$-$';*
- $\mathcal{V}_v(sec\mathtt{I}) \overset{\mathrm{df}}{=} \mathcal{V}_p(\mathtt{In})$, *if $sec\mathtt{I} \equiv$ '$\mathtt{all}$';*
- $\mathcal{V}_v(sec\mathtt{I}) \overset{\mathrm{df}}{=} \{\mathcal{V}_v(\mathtt{i_1}), ..., \mathcal{V}_v(\mathtt{i_n})\}$ *where $\{\mathtt{i_1}, ..., \mathtt{i_n}\}$ are all the input variables appearing in $sec\mathtt{I}$, otherwise.*

Analogously, let $sec\mathtt{O}$ be an output section of some cooperation pattern. We denote by $\mathcal{V}_v(sec\mathtt{O})$ the variable interpretation of the output section $sec\mathtt{O}$ defined in the following way:

- $\mathcal{V}_v(sec\mathtt{O}) \overset{\mathrm{df}}{=} \emptyset$, *if $sec\mathtt{O} \equiv$ '$-$';*
- $\mathcal{V}_v(sec\mathtt{O}) \overset{\mathrm{df}}{=} \mathcal{V}_p(\mathtt{Out})$, *if $sec\mathtt{O} \equiv$ '$\mathtt{all}{*}\mathtt{k}$' where $* \in \{\uparrow, \Uparrow\}$, or $sec\mathtt{O} \equiv$ '$\mathtt{all}$';*
- $\mathcal{V}_v(sec\mathtt{O}) \overset{\mathrm{df}}{=} \{\mathcal{V}_v(\mathtt{o_1}), ..., \mathcal{V}_v(\mathtt{o_n})\}$ *where $\{\mathtt{o_1}, ..., \mathtt{o_n}\}$ are all the input variables appearing in $sec\mathtt{O}$, otherwise.*

In contrary to bus class parameters, occurrences of bus predicate variables are of two kinds — *introduction* and *use occurrences*. Formal definition of these notions is the following.

Definition 6.13 *We say that an occurrence of a bus input variable $\mathtt{i} \in Var_{\mathrm{in}}$ in a bus predicate $\mathtt{B}$ is an* introduction occurrence *iff it is a part of an input section of arbitrary form.*

An occurrence of a bus output variable $\mathtt{o} \in Var_{\mathrm{out}}$ in a bus predicate $\mathtt{B}$ is a use occurrence *iff it appears in an output section of arbitrary form.*

Finally, an occurrence of a variable $\mathtt{x} \in Var_{\mathrm{in}} \cup Var_{\mathrm{out}}$ in a bus predicate $\mathtt{B}$ is a use occurrence *iff it appears in a rank constraint.*

Introduction occurrence of some variable in a specification gives the variable its interpretation. Syntactically, in well-defined predicates, each use occurrence of a variable requires its introduction occurrence to be included somewhere in the predicate (in a cooperation generator of some cooperation pattern). Introduction occurrences assign each variable a value which is determined by current values of the bus class parameters. Hence the interpretation of bus predicate variables depends on the actual interpretation of parameters. This relationship between the two interpretations allows us to build a concrete model for a given abstract bus specification.

Bus Instance Construction

The model (the particular cooperation machine) is assigned to a bus class and the given parameter values by the algorithm which traverses the bus predicate linearly and for each cooperation generator and a rank constraint constructs the respective part of the cooperation machine. The preliminary assumption of the construction is that the given parameter values have to satisfy the parameter constraint. In the following definition, the notion of the model (bus instance) is introduced formally and the algorithm of its construction is consequently presented.

Definition 6.14 *Let $\mathcal{B}(\mathtt{In}, \mathtt{Out}, [\mathtt{rank}, \mathtt{capacity}]) = \{\varphi, \beta\}$ a bus class with φ a parameter constraint and β a bus predicate. Further let $\mathcal{V}_p$ a parameter interpretation which satisfies the parameter constraint φ. Define the* model (bus instance) *of the bus class $\mathcal{B}$, written $\mathcal{B}(\mathcal{V}_p(\mathtt{In}), \mathcal{V}_p(\mathtt{Out}), [\mathcal{V}_p(\mathtt{rank}), \mathcal{V}_p(\mathtt{capacity})])$, as a cooperation machine $cm_{\mathcal{B}_{\mathcal{V}_p}} \overset{\mathrm{df}}{=} \langle Q, 2^{\mathcal{V}_p(\mathtt{In})} \times 2^{\mathcal{V}_p(\mathtt{Out})}, T, q_0 \rangle$ which is constructed by Algorithm 6.18.*

The algorithm of bus instance construction has basically two phases which are applied to each cooperation pattern of the particular bus predicate:

1. Computation of variable interpretations.

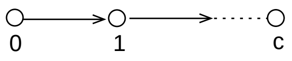
2. Computation of the cooperation machine.

In the phase (1) the set of all possible interpretations of bus predicate variables, denoted $Ints$, which occur in input and output handles of the current cooperation pattern, is computed. Interpretation of bus input variables is realised according to the given interpretation of the `In` parameter of the bus class. Analogously, bus output variables are interpreted according to the interpretation of `Out` parameter. If a rank constraint is additionally present in the pattern then the possible variable interpretations are limited only to those ones which satisfy the rank constraint.

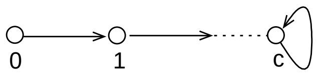
The phase (2) comprises construction of fragments of the resulting cooperation machine (the model). For each particular cooperation pattern `P` the respective fragment of the model is constructed. This fragment is determined by the transition relation $T_{\mathtt{P}}$. In general, states of the resulting cooperation machine are determined with respect to character of the bus class. If the bus class is memory-less then the state space consists of only one state and the transition relation is defined by reflexive transitions each of which is given by applying a precomputed variable interpretation (any member of the set $Ints$ from the previous phase) to the cooperation generator of the current cooperation pattern. Additionally, if some universal unfolding operator is included in the generator then the transition relation is extended to include all the respective sub-cooperations.

If the bus class contains a memory then the state space construction is more intricated. In general, the number of states is given by the expression $(y+1)^x$ where x is the number of all different memory locations introduced throughout the bus predicate, and y is the capacity of the memory determined by the actual interpretation of the `capacity` parameter. As each memory location is organised as a buffer, each of the memory input and output handles has the state-transition semantics defined as depicted in Table 6.1 (if considered separately of the other handles in the cooperation generator). In the table, the constant $c \in \mathcal{N}$ denotes the required memory capacity.

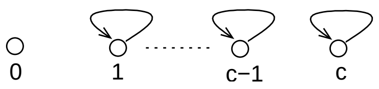
- $\mathtt{i}\!\uparrow\!\mathtt{k}$, $\mathtt{i}\!\uparrow\!@$



- $\mathtt{i}\Uparrow\mathtt{k}$, $\mathtt{i}\Uparrow@$



- $\mathtt{o}\!\uparrow\!\mathtt{k}$, $\mathtt{o}\!\uparrow\!@$,$\{\mathtt{o}_1,...,\mathtt{o}_n\}\!\uparrow\!\mathtt{k}$



- $\mathtt{o}\!\downarrow\!\mathtt{k}$, $\mathtt{o}\!\downarrow\!@$, $\{\mathtt{o}_1,...,\mathtt{o}_n\}\!\downarrow\!\mathtt{k}$

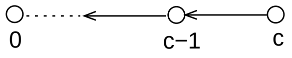


Table 6.1: Scheme of the semantics of input and output memory handles

In one cooperation generator a number of various memory handles can be specified. The intended semantics of such a combination of memory handles is given by the specific product of the state-transition semantics of the respective memory handles. For better orientation in the entire state space we denote each state with the index containing the information about the actual status of each of the memory locations. In particular, assume $\{k_1, ..., k_m, k_{m+1}, ..., k_n\}$ where $n \geq m$ is the set of all memory locations needed for interpretation of the current cooperation pattern where $\{k_1, ..., k_m\} \subset Loc$ are locations directly appearing in the bus predicate and $\{k_{m+1}, ..., k_n\} \subset \mathcal{N}$ are locations induced by rank interpretation of input rank-bounded handles, and $\kappa = \mathcal{V}_p(\mathtt{capacity})$ is the predefined memory capacity. We denote each state as $q_{\langle k_1,x_1\rangle,\langle k_2,x_2\rangle,...,\langle k_n,x_n\rangle}$ (abbreviated as $q_{k_1x_1k_2x_2...k_nx_n}$) where x_i for any $i \in \{1,...,m\}$ is either 0 or a positive number denoting the number of entries currently stored in the memory location k_i. Naturally, x_i cannot be greater than κ. The specificity of the semantics product of all handles in the cooperation pattern then lies in the following handle combination principles:

- Handle combination in the input section — input part T_i of the transition relation fragment T_P is generated in such a way that all input memory handles of the input section generate transitions representing simultaneous reading of the data incoming to the relevant bus inputs and storing this data to respective memory locations. Thus, if $\{k_1,...,k_m\}$ are locations appearing in common memory handles and $\{k_{m+1},...,k_n\}$ where $n \geq m$ are locations induced by rank interpretation of rank-bounded handles, generated transitions relate each source state of the form $q_{k_1x_1...k_mx_mk_{m+1}x_{m+1}...k_nx_n}$ with a target state of the form $q_{k_1x'_1...k_mx'_mk_{m+1}x'_{m+1}...k_nx'_n}$ where the following conditions are satisfied:
  - For each j for which the particular memory handle is of the form $i{\uparrow}k_j$ or k_j is a location induced by a rank-bounded handle $i{\uparrow}@$ it must hold $x_j < \kappa$ and $x'_j = x_j + 1$. The former is an input condition ensuring that no data may be written to a full memory by a common input handle and the latter is an output condition which ensures increasing the counter of entries stored in the particular location.
  - For each j where the memory handle has the form $i{\Uparrow}k_j$ or k_j is a location induced by a rank-bounded handle $i{\Uparrow}@$ it must hold that either $x_j < \kappa \wedge x'_j = x_j + 1$ or $x_j = \kappa \wedge x'_j = x_j$. These conditions ensure the correct semantics of $\Uparrow$-handles which allows rewriting of the data in the memory in the case when the memory is full or which behaves like a $\uparrow$-handle, otherwise.
  - A special condition has to be imposed on the combination of arbitrary two rank-bounded input handles. More specifically, we need to satisfy the requirement that two input handles does not refer to the same memory location. For the combination of rank-bounded handles with common memory handles, the required exclusivity of memory location pointers is achieved automatically by requiring $Loc \cap \mathcal{N} = \emptyset$. However, the memory location pointer of a rank-bounded handle depends on the particular ranking, i.e., the current interpretation of the `rank` parameter. To avoid generating of nonsensical transitions, we require all rank-bounded input handles in a particular input section to have mutually different ranks. Hence input variable interpreta-

tions which violate this requirement cannot be included in the set of possible variable interpretations $Ints$.

- Handle combination in the output section — output part T_o of the transition relation fragment $T_{\mathtt{P}}$ is generated in such a way that all output memory handles of the output section generate transitions representing simultaneous reading the data stored in the respective memory locations and writing this data to the relevant bus outputs. Thus, if $\{k_1, ..., k_n\}$ are locations appearing in common memory handles and $\{k_{n+1}, ..., k_{n+m}\}$ are locations induced by rank interpretation of rank-bounded handles, generated transitions relate each source state of the form $q_{k_1x_1...k_nx_nk_{n+1}x_{n+1}...k_{n+m}x_{n+m}}$ with a target state of the form $q_{k_1x'_1...k_nx'_nk_{n+1}x'_{n+1}...k_{n+m}x'_{n+m}}$ where the following conditions are satisfied:
  - For each j for which the particular memory handle is of the form $o{\uparrow}k_j$ or k_j is a location induced by a rank-bounded handle $o{\uparrow}@$ it must hold $x_j > 0$ and $x'_j = x_j$. The former is an input condition ensuring that no data may be read from an empty memory and the latter is an output condition which ensures the non-destructive behaviour of $\uparrow$-handles.
  - For each j where the memory handle has the form $o{\downarrow}k_j$ or k_j is a location induced by a rank-bounded handle $o{\downarrow}@$ it must hold $x_j > 0$ and $x'_j = x_j - 1$. These conditions ensure the correct semantics of $\downarrow$-handles which realise destructive reading of the data from the respective memory location.
  - Both kinds of output multi-handles $\{\mathtt{o_1}, ..., \mathtt{o_n}\}{\uparrow}\mathtt{k}$ and $\{\mathtt{o_1}, ..., \mathtt{o_n}\}{\downarrow}\mathtt{k}$ are treated in the same way like their common counterparts $\mathtt{o}{\uparrow}\mathtt{k}$ and $\mathtt{o}{\downarrow}\mathtt{k}$, respectively. The fact that there is a set of output variables instead of a single variable influences only the form of transition labels (w.r.t. all variable interpretations allowed). If the respective set is empty, i.e., we deal with a test handle, according to Definition 6.12 the considered interpretations have no effect on the respective part of transition labels.
  - The test output handle $?\mathtt{k}$ has a special semantics — it forces a restrictive requirement on the source states of generated transitions. This requirement states that only the transitions which have the respective component in the source state of the form $k0$. This way, a transition is added only if it evolves from a state with the particular memory location $\mathtt{k} \in Loc$ empty.
- Combination of input and output sections — the transition relation fragment $T_{\mathtt{P}}$ is constructed by composition of both parts T_i and T_o.

If any of the two sections in the particular cooperation generator is of the form $'-'$ or '`all`' then the respective transition relation part is defined as a total relation. It conforms to the fact that this kind of sections does not impose any restrictive requirements on generation of transitions.

Note that above we have considered handles which span all memory locations of a particular bus class. More precisely, we have dealt with a cooperation pattern in which a memory handle for each particular memory location is included. In general, cooperation patterns do not typically have such a special form. Hence a more general mechanism which gives semantics to cooperation patterns of arbitrary form has to be considered. The main idea of such a general mechanism relies on the fact that for a cooperation generator which contains memory locations $\{k_1, ..., k_m\}$ where $m < n$ all the relevant transitions can be generated in terms of the above mentioned scheme, but must be additionally extended to the state space where the memory counters of the remaining locations $\{k_{m+1}, ..., k_n\}$ can contain any value between 0 and κ. If this extension is applied to any combination of handles and also to combination of entire input and output sections for construction of any transition relation fragment T_{P} then the resulting transition relation reflects all kinds of cooperation generators.

Moreover, with this general mechanism the two conditions treating the output section handle combination can be employed also for interpretation of the output sections of the forms $\texttt{all} * k$ and $\{\texttt{o}_1, ..., \texttt{o}_\texttt{n}\} * \texttt{k}$ where $*$ represents either $\uparrow$ or $\downarrow$. The only difference is that only one memory location has to be treated in these cases.

On the principles introduced above, the algorithm of bus instance construction is based. Before we present this algorithm formally, we have to introduce some elementary notions concerning operators over tuples. As tuples are used as descriptors of states of bus instance state space, such operations enables us to formally describe the construction of bus instance cooperation machine.

Notation 6.15 *Let $n \in \mathcal{N}$ and let $Prods$ be a product $Prods \overset{\mathrm{df}}{=} \prod_{i=1}^{n}(X_i \cup \{\bot\})$ of arbitrary sets $X_1, ..., X_n$ such that $\forall i \in \{1, ..., n\}.\, \bot \notin X_i$. Further let $\langle x_1, x_2, ..., x_n\rangle \in Prods$ be a n-tuple satisfying $\forall i \in \{1, ..., n\}.\, x_i \neq \bot$.*

- *Arbitrary n-tuple $\langle x_1, x_2, ..., x_n\rangle \in Prods$ is alternatively denoted $\prod_{i=1}^{n} x_i$.*
- *Let $\langle y_1, ..., y_n\rangle \in Prods$ an n-tuple satisfying $\forall i \in \{1, ..., n\}.\,(j_i \in Proj \Rightarrow y_i = x_{j_i}) \wedge (j_i \notin Proj \Rightarrow y_i = \bot)$ where $k \leq n$ and $Proj \overset{\mathrm{df}}{=} \{j_1, ..., j_k\} \subseteq \{1, ..., n\}$. We write $\langle y_1, ..., y_n\rangle \prec \langle x_1, ..., x_n\rangle$ and say that $\langle y_1, ..., y_n\rangle \prec \langle x_1, ..., x_n\rangle$ is a* projection given by the projection set *$Proj$.*

- *Let* $\langle u_1, ..., u_n\rangle \prec \langle x_1, ..., x_n\rangle$ *a projection given by the projection set* $\{i_1, ..., i_k\} \subseteq \{1, ..., n\}$ *for some* $k \leq n$ *and* $\langle v_1, ..., v_n\rangle \prec \{x_1, ..., x_n\}$ *a projection given by the projection set* $\{j_1, ..., j_{n-k}\} \subseteq \{1, ..., n\}$*. We write* $\langle u_1, ..., u_n\rangle \bowtie \langle v_1, ..., v_n\rangle$ *if and only if* $\{j_1, ..., j_{n-k}\} \cap \{i_1, ..., i_k\} = \emptyset$*.*

- *Let* $\langle y_1, ..., y_n\rangle \in Prods$ *an* n*-tuple such that* $\langle y_1, ..., y_n\rangle \prec \langle x_1, ..., x_n\rangle$*. We denote* $\langle y_1, ..., y_n\rangle \rtimes \langle x_1, ..., x_n\rangle$ *the* n*-tuple defined by the following expression*

$$\langle y_1, ..., y_n\rangle \rtimes \langle x_1, ..., x_n\rangle \overset{\text{df}}{=} \langle u_1, ..., u_n\rangle$$

satisfying $\forall i \in \{1, ..., n\}. (y_i = \bot \Rightarrow u_i \overset{\text{df}}{=} x_i) \wedge (y_i \neq \bot \Rightarrow u_i \overset{\text{df}}{=} \bot)$.

- *Finally, let* $\langle u_1, ..., u_n\rangle, \langle v_1, ..., v_n\rangle \in Prods$ *two tuples such that* $\langle u_1, ..., u_n\rangle \prec \langle x_1, ..., x_n\rangle$*,* $\langle v_1, ..., v_n\rangle \prec \langle x_1, ..., x_n\rangle$*, and* $\langle u_1, ..., u_n\rangle \bowtie \langle v_1, ..., v_n\rangle$*. Define the* union of the tuples $\langle u_1, ..., u_n\rangle$ *and* $\langle v_1, ..., v_n\rangle$*, denoted* $\langle u_1, ..., u_n\rangle \oplus \langle v_1, ..., v_n\rangle$*, by the expression:*

$$\langle u_1, ..., u_n\rangle \oplus \langle v_1, ..., v_n\rangle \overset{\text{df}}{=} \langle x_1, ..., x_n\rangle$$

The following definition introduces the notion of transition relations composition. This notion is a cornerstone for construction of the transition relation fragments for individual cooperation patterns of the bus class specification. In terms of this composition, individual transitions implementing the semantics of specific groups of input and output handles contained in a particular pattern are composed together. The intuition behind such a composition is based on the preliminary requirement that all the memory locations referred in each cooperation generator are mutually different. Moreover, the composition is defined to operate on the entire state space of the bus instance. In particular, we do not take the respective semantics of individual handles as defined above and compose them to form the entire state space. We rather take the entire state space and put all the possible transitions to it while respecting the semantics of individual handles. This way, the individual aspects of particular handle combinations are comfortably realised.

The intuition about the transition composition is demonstrated by an example in Figure 6.4. There is a transition relation fragment describing the semantics of the cooperation pattern $\mathtt{i_1}\!\uparrow\!\mathtt{k_1}, \mathtt{i_2}\!\Uparrow\!\mathtt{k_2}/-$. Transition relations T_1 and T_2 denoting semantics of the respective input handles are composed to the transition relation fragment T depicted in the right side of the figure.

Definition 6.16 *Let* $\mathcal{K} = \{k_1, ..., k_n\} \subset Loc \cup \mathcal{N}$ *and let* Q *be a set of states defined as a product* $Q \overset{\text{df}}{=} \prod_{i=1}^{n} X_i$ *of sets* $X_1, ..., X_n$ *such that* $n \in \mathcal{N}$ *and each* $X_i \subseteq \mathcal{K} \times \{0, ..., \kappa\}$ *such that* $X_i \overset{\text{df}}{=} \{k_i\} \times \{0, ..., \kappa\}$ *where* $\kappa \in \mathcal{N}$*.*

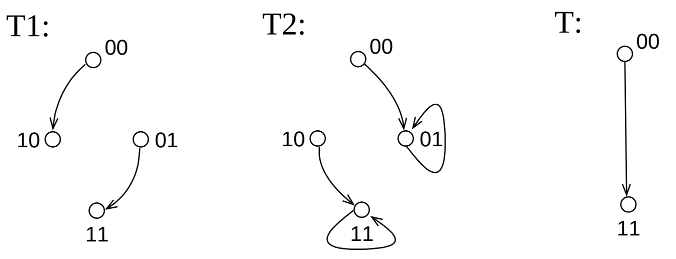


Figure 6.4: Composition of transition relations, $T := T_1 \otimes^{\{k_1\}}_{\{k_2\}} T_2$

- *Denote each state* $\langle\langle k_1, x_1\rangle, ..., \langle k_n, x_n\rangle\rangle \in Q$ *by the notation* $q_{k_1 x_1 k_2 x_2 ... k_n x_n}$.
- *Let* $K = \{k_{j_1}, ..., k_{j_k}\} \subseteq \mathcal{K}$ *where* $\{j_1, ..., j_k\} \subseteq \{1, ..., n\}$ *and* $q_u \in Q$. *We denote* $\pi(u, K)$ *the projection of* u *to locations determined by the set* K *and defined:*

$$\pi(u, K) \overset{\text{df}}{=} \langle y_1, ..., y_n\rangle$$

where $\langle y_1, ..., y_n\rangle \prec \langle u_1, ..., u_n\rangle$ *is given by the projection set* $\{j_1, ..., j_k\}$.

- *Further let* $T_1, T_2 \subseteq Q \times Q$ *relations. And* $K_1, K_2 \subseteq \mathcal{K}$ *such that* $K_1 \cap K_2 = \emptyset$.

Define composition of relations T_1 and T_2 w.r.t. K_1 and K_2, *written* $T_1 \otimes^{K_1}_{K_2} T_2$, *as a relation* $T_1 \otimes^{K_1}_{K_2} T_2 \subseteq Q \times Q$ *satisfying:*

$$T_1 \otimes^{K_1}_{K_2} T_2 := \{\langle q_u, q_v\rangle \parallel \langle q_{u_1}, q_{v_1}\rangle \in T_1 \wedge \langle q_{u_2}, q_{v_2}\rangle \in T_2$$

$$\wedge u = u_1 = u_2 \tag{6.1}$$

$$\wedge \pi(u_1, \mathcal{K} \setminus K_1) = \pi(v_1, \mathcal{K} \setminus K_1) \tag{6.2}$$

$$\wedge \pi(u_2, \mathcal{K} \setminus K_2) = \pi(v_2, \mathcal{K} \setminus K_2) \tag{6.3}$$

$$\wedge v = ((\omega_1 \oplus \omega_2) \rtimes u) \oplus (\omega'_1 \oplus \omega'_2)\} \tag{6.4}$$

where

$$\omega_1 \overset{\text{df}}{=} \pi(u_1, K_1),\ \omega_2 \overset{\text{df}}{=} \pi(u_2, K_2)$$

$$\omega'_1 \overset{\text{df}}{=} \pi(v_1, K_1),\ \omega'_2 \overset{\text{df}}{=} \pi(v_2, K_2)$$

To ensure that the relation $T_1 \otimes^{K_1}_{K_2} T_2$ from the definition above is correctly defined relation on $Q \times Q$ we have to discuss the condition which determines the target states of the relation. In particular, we have to ensure that the expression 6.4 is correctly defined. There are two requirements to be discussed — correctness of use of the tuple union operator '$\oplus$', and a requirement that v contains no undefined component $\perp$.

The first requirement is achieved by analysis of each individual occurrence of the '$\oplus$' operator in the expression 6.4. Starting from the left, the fact that K_1 and K_2 are disjunct implies $\omega_1 \bowtie \omega_2$. Hence the subterm $(\omega_1 \oplus \omega_2)$ is correct. The same arguments apply to $(\omega_1' \oplus \omega_2')$. Moreover, from the conditions 6.1, 6.2, and 6.3 follows $((\omega_1' \oplus \omega_2') \rtimes u) = ((\omega_1 \oplus \omega_2) \rtimes u)$ and hence $((\omega_1 \oplus \omega_2) \rtimes u) \bowtie (\omega_1' \oplus \omega_2')$. Thus all occurrences of the '$\oplus$' operator in the expression 6.4 are correct.

The similar reasons ensure the second requirement. It is easily observed that the equation $((\omega_1' \oplus \omega_2') \rtimes u) = ((\omega_1 \oplus \omega_2) \rtimes u)$ implies that the expression 6.4 defines a tuple which contains no $\perp$ component. The reason for that is the definition of the '$\rtimes$' operator which returns complement of a tuple contained in its sup-tuple. Such a complement operator requires the sup-tuple to contain no component of the form $\perp$ (see notation 6.15) and satisfies the following property:

$$\forall x, y, z \in Prods.\, x \rtimes y = z \Rightarrow x \oplus z = y$$

Hence if we consider the expression $((\omega_1 \oplus \omega_2) \rtimes u) \oplus (\omega_1 \oplus \omega_2)$ then we get the following equation:

$$((\omega_1 \oplus \omega_2) \rtimes u) \oplus (\omega_1 \oplus \omega_2) = u$$

Note that u contains no $\perp$. Additionally, according to conditions 6.2 and 6.3 we get the equation:

$$\pi((\omega_1 \oplus \omega_2) \rtimes u, \mathcal{K} \setminus (K_1 \cup K_2)) \bowtie \pi(\omega_1' \oplus \omega_2', K_1 \cup K_2)$$

As $(\mathcal{K} \setminus (K_1 \cup K_2)) \cup (K_1 \cup K_2) = \mathcal{K}$, we get the required result.

Additionally, we emphasise a specific feature of the transition relation composition when considered in the form '$\otimes_{\emptyset}^{\emptyset}$'. In particular, such a composition always results in a relation containing only self-transitions. This property is crucial for composition of semantics of memory-less handles and is used in the algorithm of bus instance construction.

Lemma 6.17 *Let $T_1, T_2 \subseteq Q \times Q$ arbitrary relations. Their composition $T_1 \otimes_{\emptyset}^{\emptyset} T_2$ satisfies the following condition:*

$$\langle q_u, q_v \rangle \in T_1 \otimes_{\emptyset}^{\emptyset} T_2 \Rightarrow q_u = q_v$$

Proof: Proof of this lemma follows directly from the Definition 6.16 by taking each of the sets K_1 and K_2 as $\emptyset$.

Finally, as all the preliminary notions have been introduced, we can follow in formal presenting the algorithm for bus instance construction. The algorithm constructs a bus instance for a given bus predicate β and a particular parameter interpretation $\mathcal{V}_p$. As the algorithm is considered to be general for all types of bus classes, we assume that the parameter interpretation is defined for all the necessary parameters. The fact that a memory-less bus class is considered is treated by implicit interpretation of the `capacity` parameter as zero. More precisely, we assume $\mathcal{V}_p(\texttt{capacity}) \overset{\mathrm{df}}{=} 0$ for any memory-less bus class.

Algorithm 6.18

```
procedure $GenerateModel(\beta :: bus\_predicate,\ \mathcal{V}_p :: parameter\_interpretation)$

  1. Construct the set $F$ of all memory locations contained in $\beta$:
     (a) Initiate $F := \emptyset$.
     (b) For each cooperation generator $\texttt{C}$ in $\beta$ do:
         i.   $X_1 := \{\texttt{k} \in Loc \parallel \exists \texttt{i} \in Var_{\text{in}}.\ \texttt{i}\uparrow\texttt{k} \in \beta \vee \texttt{i}\Uparrow\texttt{k} \in \beta\}$
         ii.  $X_2 := \{n \in \mathcal{N} \parallel n \in ran(\mathcal{V}_p(\texttt{rank}))\}$
         iii. $F := F \cup X_1 \cup X_2$

  2. Construct the state space $Q$:
     • $Q := \{q_0\}$, if $\mathcal{V}_p(\texttt{capacity}) = 0$,
     • $Q := \{q_i \parallel i \in \prod_{k \in F}(\{k\} \times \{0, ..., \mathcal{V}_p(\texttt{capacity})\})\}$, otherwise.

  3. Construct the transition relation $T$:
     (a) Initiate $T := \emptyset$.
     (b) For each cooperation pattern $\texttt{P}$ of the predicate $\beta$ where $\texttt{P} \equiv \texttt{C}$ or $\texttt{P} \equiv$
         $\texttt{C},\texttt{R}$ where $\texttt{C}$ is either $sec\texttt{I}/sec\texttt{O}$, $sec\texttt{I} \lhd / sec\texttt{O}$, $sec\texttt{I}/ \rhd sec\texttt{O}$ or $sec\texttt{I} \lhd$
         $/ \rhd sec\texttt{O}$ do:
         i.  Compute the set $Ints$ of interpretations of bus predicate variables
             included in $\texttt{P}$ by calling the procedure $ComputeInts(\texttt{P}, \mathcal{V}_p)$ (Al-
             gorithm 6.19).
         ii. For each $\mathcal{V}_v \in Ints$ compute the relation $T_{i\mathcal{V}_v} \subseteq Q \times Q$ relating
             all pairs of states which realise the input section identifiers. We
             write $q_1 \rightarrow_{i\mathcal{V}_v} q_2$ whenever $\langle q_1, q_2 \rangle \in T_{i\mathcal{V}_v}$.
             • If $sec\texttt{I}$ is of the form (up to the particular order of all the
               handles included):

               $$\begin{gathered} \texttt{i}_1\uparrow\texttt{k}_1, ..., \texttt{i}_{m_1}\uparrow\texttt{k}_{m_1}, \\ \texttt{i}_{m_1+1}\Uparrow\texttt{k}_{m_1+1}, ..., \texttt{i}_{m_2}\Uparrow\texttt{k}_{m_2}, \\ \texttt{i}_{m_2+1}\uparrow@, ..., \texttt{i}_{m_3}\uparrow@, \\ \texttt{i}_{m_3+1}\Uparrow@, ..., \texttt{i}_{m_4}\Uparrow@, \\ \texttt{i}_{m_4+1}, ..., \texttt{i}_{m_5} \end{gathered}$$

               where $m_1, ..., m_5 \in \mathcal{N}$ are indices denoting the borders of
               individual groups of handles satisfying $m_1 \leq m_2 \leq m_3 \leq$
               $m_4 \leq m_5$ and $m_4 > 1$ (we require at least one memory handle
               to be present).
```

$$
\begin{gathered}
T_1 := \{q_u \rightarrow_i q_v \parallel q_u, q_v \in Q \\
\wedge u \succ \textstyle\prod_{j \geq 1}^{m_1} \langle \mathtt{k}_j, x_j \rangle \wedge x_1, ..., x_{m_1} < \mathcal{V}_p(\mathtt{capacity}) \\
\wedge v = (\textstyle\prod_{j \geq 1}^{m_1} \langle \mathtt{k}_j, x_j \rangle \rtimes u) \oplus \textstyle\prod_{j \geq 1}^{m_1} \langle \mathtt{k}_j, x_j + 1 \rangle \}
\end{gathered}
$$

$$
\begin{gathered}
T_2 := \{q_u \rightarrow_i q_v \parallel q_u, q_v \in Q \\
\wedge u \succ \textstyle\prod_{j \geq m_1+1}^{m_2} \langle \mathtt{k}_j, x_j \rangle \\
\wedge v = (\textstyle\prod_{j \geq m_1+1}^{m_2} \langle \mathtt{k}_j, x_j \rangle \rtimes u) \oplus \textstyle\prod_{j \geq m_1+1}^{m_2} \langle \mathtt{k}_j, x'_j \rangle \\
\wedge \forall i, m_1 + 1 \leq i \leq m_2 . (x_i < \mathcal{V}_p(\mathtt{capacity}) \wedge x'_i = x_i + 1) \\
\vee (x_i = \mathcal{V}_p(\mathtt{capacity}) \wedge x'_i = x_i) \}
\end{gathered}
$$

$$
\begin{gathered}
T_{3_{\mathcal{V}_v}} := \{q_u \rightarrow_i q_v \parallel q_u, q_v \in Q \\
\wedge u \succ \textstyle\prod_{j \geq m_2+1}^{m_3} \langle \mathcal{V}_p(\mathtt{rank})(\mathcal{V}_v(\mathtt{i}_j)), x_j \rangle \\
\wedge x_{m_2+1}, ..., x_{m_3} < \mathcal{V}_p(\mathtt{capacity}) \\
\wedge v = (\textstyle\prod_{j \geq m_2+1}^{m_3} \langle \mathcal{V}_p(\mathtt{rank})(\mathcal{V}_v(\mathtt{i}_j)), x_j \rangle \rtimes u) \\
\oplus \textstyle\prod_{j \geq m_2+1}^{m_3} \langle \mathcal{V}_p(\mathtt{rank})(\mathcal{V}_v(\mathtt{i}_j)), x_j + 1 \rangle \}
\end{gathered}
$$

$$
\begin{gathered}
T_{4_{\mathcal{V}_v}} := \{q_u \rightarrow_i q_v \parallel q_u, q_v \in Q \\
\wedge u \succ \textstyle\prod_{j \geq m_3+1}^{m_4} \langle \mathcal{V}_p(\mathtt{rank})(\mathcal{V}_v(\mathtt{i}_j)), x_j \rangle \\
\wedge v = (\textstyle\prod_{j \geq m_3+1}^{m_4} \langle \mathcal{V}_p(\mathtt{rank})(\mathcal{V}_v(\mathtt{i}_j)), x_j \rangle \rtimes u) \\
\oplus \textstyle\prod_{j \geq m_3+1}^{m_4} \langle \mathcal{V}_p(\mathtt{rank})(\mathcal{V}_v(\mathtt{i}_j)), x'_j \rangle \\
\wedge \forall i, m_3 + 1 \leq i \leq m_4 . (x_i < \mathcal{V}_p(\mathtt{capacity}) \wedge x'_i = x_i + 1) \\
\vee (x_i = \mathcal{V}_p(\mathtt{capacity}) \wedge x'_i = x_i) \}
\end{gathered}
$$

$$T_{i_{\mathcal{V}_v}} := (T_1 \otimes_{K_2}^{K_1} T_2) \otimes_{K_3 \cup K_4}^{K_1 \cup K_2} (T_{3_{\mathcal{V}_v}} \otimes_{K_4}^{K_3} T_{4_{\mathcal{V}_v}})$$

where

$$K_1 \overset{\text{df}}{=} \{k_1, ..., k_{m_1}\} \quad K_2 \overset{\text{df}}{=} \{k_{m_1+1}, ..., k_{m_2}\}$$

$$K_3 \overset{\text{df}}{=} \{\mathcal{V}_p(\mathtt{rank})(\mathcal{V}_v(x)) \parallel x \in \{\mathtt{i}_{m_2+1}, ..., \mathtt{i}_{m_3}\}\}$$

$$K_4 \overset{\text{df}}{=} \{\mathcal{V}_p(\mathtt{rank})(\mathcal{V}_v(x)) \parallel x \in \{\mathtt{i}_{m_3+1}, ..., \mathtt{i}_{m_4}\}\}$$

Compute the set of locations used in input section $K_{i_{\mathcal{V}_v}}$:

$$K_{i_{\mathcal{V}_v}} \overset{\text{df}}{=} \bigcup_{i \in \{1,...,4\}} K_i$$

- *If* $sec\mathtt{I}$ *is of arbitrary possible form different from the previous one then*

$$T_{i\mathcal{V}_v} := \{q_u \to q_v \parallel q_u, q_v \in Q\}$$

Compute the set of locations used in input section $K_{i_{\mathcal{V}_v}}$*:*

$$K_{i_{\mathcal{V}_v}} \overset{\text{df}}{=} \emptyset$$

iii. *For each* $\mathcal{V}_v \in Ints$ *compute the relation* $T_{o\mathcal{V}_v} \subseteq Q \times Q$ *relating all pairs of states which realise the* output *section identifiers. We write* $q_1 \to_{o_{\mathcal{V}_v}} q_2$ *whenever* $\langle q_1, q_2\rangle \in T_{o\mathcal{V}_v}$.

- *If* $sec\mathtt{O}$ *is of the form (up to the particular order of all the handles included):*

$$\begin{gathered}
\mathtt{o}_1{\uparrow}\mathtt{k}_1, ..., \mathtt{o}_{n_1}{\uparrow}\mathtt{k}_{n_1},\\
\{\mathtt{o}^1_{n_1+1}, ..., \mathtt{o}^{l_1}_{n_1+1}\}{\uparrow}\mathtt{k}_{n_1+1}, ..., \{\mathtt{o}^1_{n_2}, ..., \mathtt{o}^{l_{n_2-n_1}}_{n_2}\}{\uparrow}\mathtt{k}_{n_2},\\
\mathtt{o}_{n_2+1}{\downarrow}\mathtt{k}_{n_2+1}, ..., \mathtt{o}_{n_3}{\downarrow}\mathtt{k}_{n_3},\\
\{\mathtt{o}^1_{n_3+1}, ..., \mathtt{o}^{l'_1}_{n_3+1}\}{\downarrow}\mathtt{k}_{n_3+1}, ..., \{\mathtt{o}^1_{n_4}, ..., \mathtt{o}^{l'_{n_4-n_3}}_{n_4}\}{\downarrow}\mathtt{k}_{n_4},\\
\mathtt{o}_{n_4+1}{\uparrow}@, ..., \mathtt{o}_{n_5}{\uparrow}@,\\
\mathtt{o}_{n_5+1}{\downarrow}@, ..., \mathtt{o}_{n_6}{\downarrow}@,\\
\mathtt{o}_{n_6+1}, ..., \mathtt{o}_{n_7},\\
?\mathtt{k}_{n_7+1}, ..., ?\mathtt{k}_{n_8}
\end{gathered}$$

where

- $n_1, ..., n_8 \in \mathcal{N}$ *indices denoting the borders of individual groups of handles satisfying* $n_1 \leq n_2 \leq n_3 \leq n_4 \leq n_5 \leq n_6 \leq n_7 \leq n_8$ *and* $n_6 > 1 \vee n_8 > n_7$ *(we require at least one memory handle to be present),*
- $\forall i \in \{1, ..., n_2 - n_1\}. l_i \in \mathcal{N}$ *denotes cardinality of the respective variable set in a multi-handle having the index* $l_i + n_1$*,*
- $\forall j \in \{1, ..., n_4 - n_3\}. l'_j \in \mathcal{N}$ *each denotes cardinality of the respective variable set in a multi-handle having the index* $l'_j + n_3$*,*
- *all the output variables which occur in the format stated above are assumed to be mutually different.*

Then construct the following relations:

$$\begin{gathered}
T'_1 := \{q_u \to_{o_{\mathcal{V}_v}} q_v \parallel q_u, q_v \in Q\\
\wedge u \succ \textstyle\prod_{j\geq 1}^{n_2}\langle \mathtt{k}_j, x_j\rangle \wedge x_1, ..., x_{n_2} > 0\\
\wedge v = (\textstyle\prod_{j\geq 1}^{n_2}\langle \mathtt{k}_j, x_j\rangle \rtimes u) \oplus \textstyle\prod_{j\geq 1}^{n_2}\langle \mathtt{k}_j, x_j\rangle\}
\end{gathered}$$

$$\begin{aligned}T'_2 := \{q_u \rightarrow_{o\mathcal{V}_v} q_v \parallel q_u, q_v \in Q \\ \wedge u \succ \textstyle\prod_{j\geq n_2+1}^{n_4} \langle \mathtt{k}_j, x_j\rangle \wedge x_{n_2+1}, ..., x_{n_4} > 0 \\ \wedge v = (\textstyle\prod_{j\geq n_2+1}^{n_4} \langle \mathtt{k}_j, x_j\rangle \rtimes u) \oplus \textstyle\prod_{j\geq n_2+1}^{n_4} \langle \mathtt{k}_j, x_j - 1\rangle\}\end{aligned}$$

$$\begin{aligned}T'_{3_{\mathcal{V}_v}} := \{q_u \rightarrow_{o\mathcal{V}_v} q_v \parallel q_u, q_v \in Q \\ \wedge u \succ \textstyle\prod_{j\geq n_4+1}^{n_5} \langle \mathcal{V}_p(\mathtt{rank})(\mathcal{V}_v(\mathtt{o}_j))), x_j\rangle \wedge x_{n_4+1}, ..., x_{n_5} > 0 \\ \wedge v = (\textstyle\prod_{j\geq n_4+1}^{n_5} \langle \mathcal{V}_p(\mathtt{rank})(\mathcal{V}_v(\mathtt{o}_j)), x_j\rangle \rtimes u) \\ \oplus \textstyle\prod_{j\geq n_4+1}^{n_5} \langle \mathcal{V}_p(\mathtt{rank})(\mathcal{V}_v(\mathtt{o}_j)), x_j\rangle\}\end{aligned}$$

$$\begin{aligned}T'_4 := \{q_u \rightarrow_{o\mathcal{V}_v} q_v \parallel q_u, q_v \in Q \\ \wedge u \succ \textstyle\prod_{j\geq n_5+1}^{n_6} \langle \mathcal{V}_p(\mathtt{rank})(\mathcal{V}_v(\mathtt{o}_j)), x_j\rangle \wedge x_{n_5+1}, ..., x_{n_6} > 0 \\ \wedge v = (\textstyle\prod_{j\geq n_5+1}^{n_6} \langle \mathcal{V}_p(\mathtt{rank})(\mathcal{V}_v(\mathtt{o}_j)), x_j\rangle \rtimes u) \\ \oplus \textstyle\prod_{j\geq n_5+1}^{n_6} \langle \mathcal{V}_p(\mathtt{rank})(\mathcal{V}_v(\mathtt{o}_j)), x_j - 1\rangle\}\end{aligned}$$

$$T_{o\mathcal{V}_v} := (T'_1 \otimes_{K_2}^{K_1} T'_2) \otimes_{K_3\cup K_4}^{K_1\cup K_2} (T'_{3_{\mathcal{V}_v}} \otimes_{K_4}^{K_3} T'_{4_{\mathcal{V}_v}})$$

where

$$\begin{aligned}K_1 \overset{\mathrm{df}}{=} \{k_1, ..., k_{n_2}\} \quad K_2 \overset{\mathrm{df}}{=} \{k_{n_2+1}, ..., k'_{n_4}\} \\ K_3 \overset{\mathrm{df}}{=} \{\mathcal{V}_p(\mathtt{rank})(\mathcal{V}_v(x)) \parallel x \in \{\mathtt{o}_{n_4+1}, ..., \mathtt{o}_{n_5}\}\} \\ K_4 \overset{\mathrm{df}}{=} \{\mathcal{V}_p(\mathtt{rank})(\mathcal{V}_v(x)) \parallel x \in \{\mathtt{o}_{n_5+1}, ..., \mathtt{o}_{n_6}\}\}\end{aligned}$$

Compute the set of locations used in output section $K_{o\mathcal{V}_v}$:

$$K_{o\mathcal{V}_v} \overset{\mathrm{df}}{=} \{k_{n_7+1}, ..., k_{n_8}\} \cup \bigcup_{i\in\{1,...,4\}} K_i$$

Remove all transitions the source states of which do not satisfy all the tests $?\mathtt{k}_{n_7+1}, ..., ?\mathtt{k}$, if there are any included in `P`*:*

$$T_{o\mathcal{V}_v} := \{\langle q_u, q_v\rangle \in T_{o\mathcal{V}_v} \| u \succ \prod_{j\geq n_7+1}^{n_8} \langle \mathtt{k}_j, x_j\rangle \wedge x_{n_7+1}, ..., x_{n_8} = 0\}$$

- *If sec*`O` *is of the form* `all`$\uparrow$`k` *then*

$$\begin{aligned}T_{o\mathcal{V}_v} := \{q_u \rightarrow_{o\mathcal{V}_v} q_v \parallel q_u, q_v \in Q \\ \wedge u \succ \langle \mathtt{k}, x\rangle \wedge x > 0 \\ \wedge v = (\langle \mathtt{k}, x\rangle \rtimes u) \oplus \langle \mathtt{k}, x\rangle\}\end{aligned}$$

Compute the set of locations used in output section $K_{o\mathcal{V}_v}$:

$$K_{o\mathcal{V}_v} \stackrel{\mathrm{df}}{=} \{\mathtt{k}\}$$

- *If sec*O *is of the form* all↓k *then*

$$T_{o\mathcal{V}_v} := \{q_u \rightarrow_{o\mathcal{V}_v} q_v \parallel q_u, q_v \in Q$$
$$\wedge u \succ \langle \mathtt{k}, x\rangle \wedge x > 0$$
$$\wedge v = (\langle \mathtt{k}, x\rangle \rtimes u) \oplus \langle \mathtt{k}, x-1\rangle\}$$

$$K_{o\mathcal{V}_v} \stackrel{\mathrm{df}}{=} \{\mathtt{k}\}$$

- *If sec*O *is of an arbitrary possible form which is different from the previous forms then*

$$T_{o\mathcal{V}_v} := \{q_u \rightarrow_{o\mathcal{V}_v} q_v \parallel q_u, q_v \in Q\}$$

$$K_{o\mathcal{V}_v} \stackrel{\mathrm{df}}{=} \emptyset$$

iv. *Construct the local transition relation $T_{\mathtt{P}}$:*

- *If* $\mathtt{C} \equiv sec\mathtt{I}/\,\mathrm{sec}\,\mathtt{O}$ *then*

$$T_{\mathtt{P}} := \{q_u \overset{\mathcal{V}_v(sec\mathtt{I})/\mathcal{V}_v(sec\mathtt{O})}{\longrightarrow} q_v \parallel \langle q_u, q_v\rangle \in T_{i\mathcal{V}_v} \otimes^{K_{i\mathcal{V}_v}}_{K_{o\mathcal{V}_v}} T_{o\mathcal{V}_v}$$
$$\wedge \mathcal{V}_v \in Ints\}$$

- *If* $\mathtt{C} \equiv sec\mathtt{I} \lhd /\,\mathrm{sec}\,\mathtt{I}$ *then*

$$T_{\mathtt{P}} := \{q_u \overset{W/\mathcal{V}_v(sec\mathtt{O})}{\longrightarrow} q_v \parallel \langle q_u, q_v\rangle \in T_{i\mathcal{V}_v} \otimes^{K_{i\mathcal{V}_v}}_{K_{o\mathcal{V}_v}} T_{o\mathcal{V}_v}$$
$$\wedge W \in 2^{\mathcal{V}_v(sec\mathtt{I})} \wedge W \neq \emptyset \wedge \mathcal{V}_v \in Ints\}$$

- *If* $\mathtt{C} \equiv sec\mathtt{I}/ \rhd \mathrm{sec}\,\mathtt{I}$ *then*

$$T_{\mathtt{P}} := \{q_u \overset{\mathcal{V}_v(sec\mathtt{I})/R}{\longrightarrow} q_v \parallel \langle q_u, q_v\rangle \in T_{i\mathcal{V}_v} \otimes^{K_{i\mathcal{V}_v}}_{K_{o\mathcal{V}_v}} T_{o\mathcal{V}_v}$$
$$\wedge R \in 2^{\mathcal{V}_v(sec\mathtt{O})} \wedge R \neq \emptyset \mathcal{V}_v \in Ints\}$$

- *If* $\mathtt{C} \equiv sec\mathtt{I} \lhd / \rhd \mathrm{sec}\,\mathtt{I}$ *then*

$$T_{\mathtt{P}} := \{q_u \overset{W/R}{\longrightarrow} q_v \parallel \langle q_u, q_v\rangle \in T_{i\mathcal{V}_v} \otimes^{K_{i\mathcal{V}_v}}_{K_{o\mathcal{V}_v}} T_{o\mathcal{V}_v}$$
$$\wedge W \in 2^{\mathcal{V}_v(sec\mathtt{I})} \wedge R \in 2^{\mathcal{V}_v(sec\mathtt{O})}$$
$$\wedge W, R \neq \emptyset \wedge \mathcal{V}_v \in Ints\}$$

v. *Increment the global transition relation* T*:*

$$T := T \cup T_{\mathtt{P}}$$

4. *Determine the initial state* q_0*:*

   - $q_0 \in Q \wedge \|Q\| = 1$, *if* $\mathcal{V}_p(\mathtt{capacity}) = 0$,
   - $q_0 := q_u$ *where* $u = \prod_{k \in F} \langle k, 0 \rangle$, *otherwise.*

Algorithm 6.19

`procedure` $ComputeInts(\mathtt{P}\texttt{::}$*cooperation_pattern*, $\mathcal{V}_p\texttt{::}$*parameter_interpretation*$)$

1. *Assume* $\mathtt{P} \equiv \mathtt{C}$ *or* $\mathtt{P} \equiv \mathtt{C}, \mathtt{R}$.

2. *Compute the set* $Ints_i$ *of interpretations of bus predicate variables included in* $sec\mathtt{I}$*:*

   - *If* $sec\mathtt{I}$ *is of the form (up to the particular order of all the handles included):*

$$\begin{gathered}
\mathtt{i}_1 \uparrow \mathtt{k}_1, ..., \mathtt{i}_{m_1} \uparrow \mathtt{k}_{m_1}, \\
\mathtt{i}_{m_1+1} \Uparrow \mathtt{k}_{m_1+1}, ..., \mathtt{i}_{m_2} \Uparrow \mathtt{k}_{m_2}, \\
\mathtt{i}_{m_2+1} \uparrow @, ..., \mathtt{i}_{m_3} \uparrow @, \\
\mathtt{i}_{m_3+1} \Uparrow @, ..., \mathtt{i}_{m_4} \Uparrow @, \\
\mathtt{i}_{m_4+1}, ..., \mathtt{i}_{m_5}
\end{gathered}$$

where $m_1, ..., m_5 \in \mathcal{N}$ *are indices denoting the borders of individual groups of handles satisfying* $m_1 \leq m_2 \leq m_3 \leq m_4 \leq m_5$ *and* $m_4 > 1$ *(we require at least one memory handle to be present), then:*

$$Ints_i := \{\mathcal{V}_v \parallel \rho_1(\mathcal{V}_v, \{\mathtt{i}_1, ..., \mathtt{i}_{m_5}\}) \wedge \neg\rho_2(\mathcal{V}_v, \{\mathtt{i}_{m_2+1}, ..., \mathtt{i}_{m_4}\})\}$$

where

$$\rho_1(\mathcal{V}_v, \{\mathtt{i}_1, ..., \mathtt{i}_m\}) :=$$

$$\begin{gathered}
\exists \{i_1, ..., i_m\} \subseteq \mathcal{V}_p(\mathtt{In}).\, \|\{i_1, ..., i_m\}\| = m \\
\wedge \bigwedge_{k \geq 1}^{m} \mathcal{V}_v(\mathtt{i}_k) = i_k \\
\wedge \forall x \in Var_{\mathrm{in}} \cup Var_{\mathrm{out}}.\, \{x\} \cap \{\mathtt{i}_1, ..., \mathtt{i}_m\} = \emptyset \Rightarrow \mathcal{V}_v(x) = \bot
\end{gathered}$$

$$\rho_2(\mathcal{V}_v, \{\mathtt{i}_k, ..., \mathtt{i}_l\}) :=$$

$$\bigvee_{k_1, k_2 \in \{k, ..., l\}} k_1 \neq k_2 \Rightarrow \mathcal{V}_p(\mathtt{rank})(\mathcal{V}_v(\mathtt{i}_{k_1})) = \mathcal{V}_p(\mathtt{rank})(\mathcal{V}_v(\mathtt{i}_{k_2}))$$

- *If* $sec\mathtt{I}$ *is of either form different from the previous one then*

$$Ints_i := \{\mathcal{V}_v \parallel \forall x \in Var_{in} \cup Var_{out}.\, \mathcal{V}_v(x) = \bot\}$$

3. *Compute the set* $Ints_o$ *of interpretations of bus predicate variables included in* $sec\mathtt{O}$*:*

- *If* $sec\mathtt{O}$ *is of the form (up to the particular order of all the handles included):*

$$\begin{gathered}\mathtt{o}_1\Uparrow\mathtt{k}_1, ..., \mathtt{o}_{n_1}\Uparrow\mathtt{k}_{n_1},\\ \{\mathtt{o}^1_{n_1+1}, ..., \mathtt{o}^{l_1}_{n_1+1}\}\Uparrow\mathtt{k}_{n_1+1}, ..., \{\mathtt{o}^1_{n_2}, ..., \mathtt{o}^{l_{n_2-n_1}}_{n_2}\}\Uparrow\mathtt{k}_{n_2},\\ \mathtt{o}_{n_2+1}\Downarrow\mathtt{k}_{n_2+1}, ..., \mathtt{o}_{n_3}\Downarrow\mathtt{k}_{n_3},\\ \{\mathtt{o}^1_{n_3+1}, ..., \mathtt{o}^{l'_1}_{n_3+1}\}\Downarrow\mathtt{k}_{n_3+1}, ..., \{\mathtt{o}^1_{n_4}, ..., \mathtt{o}^{l'_{n_4-n_3}}_{n_4}\}\Downarrow\mathtt{k}_{n_4},\\ \mathtt{o}_{n_4+1}\Uparrow@, ..., \mathtt{o}_{n_5}\Uparrow@,\\ \mathtt{o}_{n_5+1}\Downarrow@, ..., \mathtt{o}_{n_6}\Downarrow@,\\ \mathtt{o}_{n_6+1}, ..., \mathtt{o}_{n_7},\\ ?\mathtt{k}_{n_7+1}, ..., ?\mathtt{k}_{n_8}\end{gathered}$$

where

- $n_1, ..., n_8 \in \mathcal{N}$ *indices denoting the borders of individual groups of handles satisfying* $n_1 \leq n_2 \leq n_3 \leq n_4 \leq n_5 \leq n_6 \leq n_7 \leq n_8$ *and* $n_7 > 1$ *(we require at least one bus output variable to be present),*
- $\forall i \in \{1, ..., n_2 - n_1\}.\, l_i \in \mathcal{N}$ *denotes cardinality of the respective variable set in a multi-handle having the index* $l_i + n_1$*,*
- $\forall j \in \{1, ..., n_4 - n_3\}.\, l'_j \in \mathcal{N}$ *each denotes cardinality of the respective variable set in a multi-handle having the index* $l'_j + n_3$*,*
- *all the output variables which occur in the format stated above are assumed to be mutually different;*

then

$$\begin{gathered}Ints_o := \{\mathcal{V}_v \parallel \rho_3(\mathcal{V}_v, \{\mathtt{o}_1, .., \mathtt{o}_{n_7}\} \cup \textstyle\bigcup_{i=n_1+1}^{n_2}\{\mathtt{o}^1_i, .., \mathtt{o}^{l_{i-n_1}}_i\}\\ \cup \textstyle\bigcup_{i=n_3+1}^{n_4}\{\mathtt{o}^1_i, .., \mathtt{o}^{l'_{i-n_3}}_i\}) \wedge \neg\rho_4(\mathcal{V}_v, \{\mathtt{o}_{n_4+1}, .., \mathtt{o}_{n_6}\})\}\end{gathered}$$

where

$\rho_3(\mathcal{V}_v, \{\mathtt{o}_1, ..., \mathtt{o}_n\}) :=$

$$\begin{gathered}\exists\{o_1, ..., o_n\} \subseteq \mathcal{V}_p(\mathtt{Out}).\, \|\{o_1, ..., o_n\}\| = n\\ \wedge \textstyle\bigwedge_{k\geq 1}^{n} \mathcal{V}_v(\mathtt{o}_k) = o_k\\ \wedge\forall x \in Var_{\text{in}} \cup Var_{\text{out}}.\, \{x\} \cap \{\mathtt{o}_1, ..., \mathtt{o}_n\} = \emptyset \Rightarrow \mathcal{V}_v(x) = \bot\end{gathered}$$

$$\rho_4(\mathcal{V}_v, \{\mathtt{o}_k, ..., \mathtt{o}_l\}) :=$$

$$\bigvee_{k_1, k_2 \in \{k, \dots, l\}} k_1 \neq k_2 \Rightarrow \mathcal{V}_p(\mathtt{rank})(\mathcal{V}_v(\mathtt{o}_{k_1})) = \mathcal{V}_p(\mathtt{rank})(\mathcal{V}_v(\mathtt{o}_{k_2}))$$

- *If sec*$\mathtt{O}$ *is of the either form different from the above treated forms then*

$$Ints_o := \{\mathcal{V}_v \parallel \forall x \in Var_{in} \cup Var_{out}. \mathcal{V}_v(x) = \bot\}$$

4. *Compute the set* $Ints$ *of all the variable interpretations specified by* $\mathtt{P}$ *with respect to the given parameter interpretation* $\mathcal{V}_p$*:*

   - *If* $\mathtt{P} \equiv \mathtt{C}, \mathtt{R}$ *then*

$$Ints := \{\mathcal{V}_v \parallel \mathcal{V}_{vi} \in Ints_i, \mathcal{V}_{vo} \in Ints_o. \mathcal{V}_v = \mathcal{V}_{vi} \cup \mathcal{V}_{vo} \\ \wedge \rho_5(\mathcal{V}_v, \mathtt{C}) \wedge \rho_6(\mathcal{V}_v, \mathtt{R})\}$$

   where

$$\rho_5(\mathcal{V}_v, \mathtt{C}) := \bigvee_{\substack{x_1, x_2 \in vars(\mathtt{C}) \\ x_1 \neq x_2}} \mathcal{V}_p(\mathtt{rank})(\mathcal{V}_v(x_1)) = \mathcal{V}_p(\mathtt{rank})(\mathcal{V}_v(x_2))$$

   where $vars(\mathtt{C})$ *is a set of all rank-bounded memory handles included in* $\mathtt{C}$.

$$\rho_6(\mathcal{V}_v, \mathtt{R}) := \mathtt{R}[\mathcal{V}_p(\mathtt{rank})(\mathcal{V}_v(\mathtt{x}))/\mathtt{rank(x)}]$$

   - *If* $\mathtt{P} \equiv \mathtt{C}$ *then*

$$Ints := \{\mathcal{V}_v \parallel \mathcal{V}_{vi} \in Ints_i, \mathcal{V}_{vo} \in Ints_o. \mathcal{V}_v = \mathcal{V}_{vi} \cup \mathcal{V}_{vo} \wedge \rho_5(\mathcal{V}_v, \mathtt{C})\}$$

To argue that both of the algorithms above are correctly defined in the sense that for given values of the respective input parameters each of the algorithms stops the computation with the expected result, we have to go through the algorithms step by step. During this analysis we also sketch briefly a way of how these algorithms can be implemented. These sketches imply the rough approximation of time complexity of the algorithms. Concerning the complexity issues, we are not going into details here as we present both algorithms primary in order to formally define the process of bus class instantiation.

Correctness of Variable Interpretation Computation

We start with Algorithm 6.19. Let $\mathtt{P}$ a cooperation pattern and $\mathcal{V}_p$ a bus class parameter interpretation. We analyse the execution of the procedure $ComputeInts(\mathtt{P}, \mathcal{V}_p)$.

The computation in the step (2) accepts all the possible forms of the cooperation pattern input section. In the nontrivial situation when $sec\mathtt{I}$ is of the general form, we need to analyse the construction of the set $Ints_i$. Note that the set of all possible interpretations of variables included in an arbitrary cooperation pattern is always finite due to the two facts — at first, the number of variables in an arbitrary cooperation pattern is always finite, at second, the parameters `In` and `Out` are always interpreted as finite sets. Hence we have to argue that computation of validity of the two predicates ρ_1 and ρ_2 finishes in every possible case.

Both predicates ρ_1 and ρ_2 can be comprehended as requirements on the form of the possible variable interpretations. Each generated interpretation is determined by its range defined as a subset of $\mathcal{V}_p(\mathtt{In})$ which has a given finite cardinality. For the computation of $Ints_i$, we can consider these predicates to be applied already during generation of the interpretations so that each generated interpretation is required to satisfy it. The construction procedure iteratively computes all the interpretations which satisfy the mutual rank divergence stated by the predicate ρ_2 and which are additionally defined on the domain $\{\mathtt{i}_1, ..., \mathtt{i}_m\}$ with the range equal to an arbitrary subset of $\mathcal{V}_p(\mathtt{In})$ of the required cardinality m. The last conjunction of the predicate ρ_1 has only formal sense and is realised by considering the generated interpretations to be implicitly partial functions. Such a setting is important for the correct computation of bus instance in Algorithm 6.18. To conclude the analysis of the step (2) we have to discuss a one last issue. As the set $\mathcal{V}_p(\mathtt{In})$ and cardinality of the required subsets are finite, and hence $Ints_i$ is finite too, the construction procedure in the step (2) constructs a finite number of interpretations and stops.

The step (3) (computation of $Ints_o$) follows the same ideas and problems as the previous step. The only difference is that instead of the predicates ρ_1 and ρ_2, the counterparts of these predicates adapted to the output handles are used (ρ_3 and ρ_4, respectively). Note that the use of the predicate ρ_3 is more complicated here then the use of the predicate ρ_1 in construction of $Ints_i$. The reason for that is treating of the multi-handles included in the current cooperation pattern. Recall the syntactic requirement (1) of Definition 6.5 stating that all the variables occurring in any cooperation pattern must be mutually different. Such a requirement comprises also multi-handles. In particular, all multi-handles in the particular cooperation pattern output section must be interpreted as mutually disjoint sets of ports, and moreover, they must be disjoint from the set of all the other ports generated by the output section of the pattern. Hence the predicate ρ_3 is applied to a union of all output variables which occur in the current cooperation pattern.

Now only a one more computation step — the final step (4) regarding the computation of $Ints$ — remains for the analysis. If `P` contains no rank constraint then $Ints$ is defined as the set of interpretations each of which is defined as a union of an arbitrary interpretation in $Ints_i$ and an arbitrary interpretation in $Ints_o$ which additionally satisfies the predicate ρ_5. This predicate ensures exclusion of all interpretations which assign the same rank to any two different memory handles. This way, the requirement stated in the note 6.6 is achieved. Note that if there is no rank-bounded handle in the particular cooperation pattern then the predicate ρ_5 is simply true because of the disjunction over an empty index set. Additionally, it is worth noting that a domain of each interpretation in $Ints_i$ is disjunct with a domain of any interpretation in $Ints_o$. In the similar sense also the ranges of interpretations in $Ints_i$ and $Ints_o$ are mutually disjunct. Hence a union of variable interpretations (partial functions) is a correctly defined variable interpretation (partial function). Computation of $Ints$ can be realised directly by unification of the respective interpretations resulting with $\|Ints_i\| \cdot \|Ints_o\|$ many interpretations contained in $Ints$.

If `P` additionally contains a rank constraint then the computation is more complicated. Here only interpretations satisfying the conjunction of predicates ρ_5 and ρ_6 have to be included in $Ints$. Hence the unification of interpretations in $Ints_i$ and $Ints_o$ must be restricted only to these interpretations. The predicate ρ_6 is defined as a finite quantifier-free formula constructed directly from the rank constraint by replacing of all the included variables by the respective values given by the particular interpretation. As the link ranking parameter of the bus class is always interpreted by a finite ranking function, computation of validity of such a formula finishes for every parameter and variable interpretation. It is worth noting that due to the syntactic requirement 5 the ranking function in the predicate ρ_6 is never applied to an argument '$\perp$'.

To sum up, for any given parameter interpretation $\mathcal{V}_p$ and any cooperation pattern `P` the procedure $ComputeInts$ finishes with the resulting set $Ints$ containing all interpretations of variables included in `P` induced by the parameter interpretation $\mathcal{V}_p$. The time complexity of the computation sketched above is characterised by a polynomial function w.r.t. the size of the input structures. In particular, if $n_i = \|\mathcal{V}_p(\texttt{In})\|$, $n_o = \|\mathcal{V}_p(\texttt{Out})\|$, m_i, m_o denote each the number of input (resp. output) variables in `P`, k_i, k_o are the number of rank-bounded input handles and rank-bounded output variables respectively, and r is the number of rank comparisons in the rank constraint `R`, then the complexity is $\mathcal{O}(n_i^{m_i} \cdot (k_i+1)^2 + {n_o}^{m_o} \cdot (k_o+1)^2 + {n_i}^{m_i} \cdot {n_o}^{m_o} \cdot (r + k_i \cdot k_o + 1))$ where each summand correspond to the respective computation step described above (in the respective order).

Correctness of Bus Instance Construction

Next we concentrate on analysis of the Algorithm 6.18. Let β a bus predicate and $\mathcal{V}_p$ a bus class parameters interpretation. We analyse the execution of the procedure $GenerateModel(\beta, \mathcal{V}_p)$.

The step (1) is responsible for computation of the set of all memory locations introduced in cooperation patterns included in β. As the set of all memory locations referred in an arbitrary bus predicate is always finite and so is a range of an arbitrary ranking function given by any parameter interpretation, the computation is finished in every case and returns the required set.

The state space construction in the step (2) depends on the fact if the bus class is memory-less or not. In the negative case, the global state space is constructed taking all memory locations and their uniform capacity given by the `capacity` parameter interpretation into account. As both the capacity and number of memory locations is required to be finite, the state space is also finite.

The most intricated step of the computation is the step (3) which is responsible for generation of the transition relation on the state space constructed in the previous step. The computation of the step (3) runs in a loop executed for each cooperation pattern `P` included in β. The result of each iteration of the loop is construction of the particular fragment of the transition relation. This fragment, denoted $T_{\mathtt{P}}$, comprises the transitions induced by the actual cooperation pattern `P`. We analyse each of the respective computation steps of the loop individually.

In the first step of the loop, denoted *(3b i.)*, the Algorithm 6.19 is run. As we have discussed above, the respective computation stops for every input parameters and gives the expected result. In the step *(3b ii.)*, the input part $T_{i\mathcal{V}_v}$ of the transition relation fragment $T_{\mathtt{P}}$ is constructed for each $\mathcal{V}_v \in Ints$. $T_{i\mathcal{V}_v}$ can be comprehended as a transition relation with unspecified labels. Its main purpose is to relate states according to the semantics of the combination of the included input handles. Construction of $T_{i\mathcal{V}_v}$ depends on the form of the input section of the cooperation generator included in `P`. At first we discuss the computation in the case when the input section is of the general form including arbitrary number (denoted m_1) of memory-less input handles, arbitrary number (denoted $m_2 - m_1$) of memory $\uparrow$-input handles, arbitrary number (denoted $m_3 - m_2$) of memory $\Uparrow$-input handles, arbitrary number (denoted $m_4 - m_3$) of rank-bounded memory $\uparrow$-input handles, and finally arbitrary number (denoted $m_5 - m_4$) of rank-bounded memory $\Uparrow$-input handles. Especially, at least one memory handle is expected here as the strict inequality $m_1 < m_5$ is required to be satisfied. In this situation, a specific transition relation is constructed for each of the above mentioned groups of input handles. All of these transition relations have the same form as $T_{i\mathcal{V}_v}$, hence they can be intuitively comprehended as parts of $T_{i\mathcal{V}_v}$. The relations T_1 and T_2 are independent of the variable interpretation. T_1 realises composition of semantics of all the memory $\uparrow$-input handles and fits them into the overall state space. Such a composition corresponds to a specific product of several automatons as the one depicted in Figure 6.1a. The specificity of the product is given by concerning the behaviour of all the respective input handles to be performed simultaneously. Similarly, T_2 realises composition of semantics of all the memory $\Uparrow$-input handles.

The relations $T_{3_{\mathcal{V}_v}}$ and $T_{4_{\mathcal{V}_v}}$ are defined for each interpretation separately which goes with the fact that the memory location induced by rank-bounded memory handles is given by evaluating the respective rank on an interpretation of the particular input variable. In other respects the relations are constructed similarly to relations T_1 and T_2, respectively. Computation of each of the four relations above must always stop because of finiteness of the state space Q.

After all the relations above are constructed, their composition with respect to the operator $'\otimes'$ is realised. At first, T_1 is composed with T_2 w.r.t. the respective mutually disjoint sets of memory locations. At next, $T_{3_{\mathcal{V}_v}}$ and $T_{4_{\mathcal{V}_v}}$ are composed w.r.t. the respective sets of locations. Finally, composition of the relations resulting from the previous two compositions is realised.

For the input section of any possible form different from the one above (i.e., the form which contains no memory handles — '`all`' or $'-'$), we construct the relation which covers the entire state space and includes reflexive transitions (self-transitions). This goes with the intuition that memory-less input handles do not depend on actual configuration of memory locations and hence can be performed anytime.

The output section is treated in the step *(3b iii.)*. The computation is realised similarly as in the case of the input section which has been discussed above. The difference is in particular declarations of the relations which here reflect the semantics of respective output handles and in the fact that, additionally, the other forms of output section have to be treated. First kind of these forms considers multi-handles. Each multi-handle refers a one particular memory location of *Loc*, hence the generated transitions are declared to change only the respective state component. Another kind of forms considers test handles which contain no output variable. With respect to the intended semantics of these handles — testing of emptiness of the referred memory location, these handles are reflected by removing of all previously generated transition which do not satisfy such a requirement. All the other possible forms of an output section, in particular '`all`' or $'-'$, do not use any memory location and are treated in the same way like their respective input counterparts mentioned above.

In the step *(3b iv.)* construction of the transition relation fragment $T_{\mathtt{B}}$ is realised. Here transitions of previously constructed parts are composed for all interpretations in $Ints$ and extended with labels determined by evaluating the interpretation for all the input and output variables included in the respective sections. Moreover, if the cooperation pattern contains some unfolding operator then every transition labelled by each respective subset of evaluation of the particular section is added.

In the step *(3b v.)* the global transition relation is extended with the fragment just acquired. Note that the entire loop which has been just analysed must always stop because the number of cooperation patterns in a bus predicate is always finite.

The final step (4) treats determination of the initial state. This is realised with respect to the fact if the bus specification is memory-less or not. In the positive case, there is only one state in the state space and thus it is determined as the initial state. In the negative case, the state with the descriptor which corresponds to the situation when all the memory locations are empty is considered as initial.

As all the relations computed in the algorithm are finite due to finiteness of the state space and the number of variable interpretations, computation of Algorithm 6.18 must stop for any finite bus class and parameter interpretation and return the expected model. The returned model can have an empty transition relation. For example, any instance of a bus predicate `'i↑k/o↓k'` has the empty transition relation. This is due to the impossible combination of an input and an output memory handle. More precisely, there is no variable interpretation which would allow some transition to be constructed in the step *(3b iv.)* of the Algorithm 6.18.

Concerning the complexity issues, the algorithm is, roughly saying, exponential in space. The reason for that is construction of the state space which grows exponentially w.r.t. the number of memory locations contained in the bus predicate and the given capacity of the memory. The algorithm, as presented above, is certainly not the most effective one regarding the time complexity. E.g., computation of the relations T_1 and T_2 can be realised only once for each cooperation pattern, because it does not depend on variable interpretation. However, we have presented a preliminary algorithm which satisfactorily constructs a model of a bus class specification. In our future work, we aim to implement the algorithm, hence some optimisations will have to be considered. However, for the purpose of construction of buses used in VCN architectural descriptions in which the number of inputs, outputs, and also the number of memory locations and the memory capacity are typically low, we believe that this algorithm is satisfactory.

6.2.2 Model-Theoretic Issues

In the previous section the algorithm for construction of a cooperation machine which represents a model of a particular bus class specification for given parameter values was presented. Moreover, it was argued that the algorithm stops the computation for any given startup parameters and gives either the resulting cooperation machine or a state transition system with an empty transition relation. We call the former kind of model a *well-founded* model. The latter kind of model is referred to as an *empty model*. In this section we discuss for which startup conditions the algorithm produces either well-founded models or empty models.

First of all we define the notion of a *realisable bus class*. Intuitively, a realisable bus class is a bus class for which there exists at least one parameter interpretation for which the respective model is well-founded.

Definition 6.20 *Let $\mathcal{B}(\texttt{In}, \texttt{Out}, [\texttt{rank}, \texttt{capacity}]) = \{\varphi, \beta\}$ a bus class with φ a parameter constraint and β a bus predicate. Further let $\mathcal{V}_p$ a parameter interpretation which satisfies the parameter constraint φ.*

We say that the model $\mathcal{B}(\mathcal{V}_p(\texttt{In}), \mathcal{V}_p(\texttt{Out}), [\mathcal{V}_p(\texttt{rank}), \mathcal{V}_p(\texttt{capacity})]) = cm_{\mathcal{B}_{\mathcal{V}_p}} \overset{\text{df}}{=} \langle Q, 2^{\mathcal{V}_p(\texttt{In})} \times 2^{\mathcal{V}_p(\texttt{Out})}, T, q_0 \rangle$ of the bus class $\mathcal{B}$ is empty*, if $T = \emptyset$. Otherwise we say the model is* well-founded.

We say that the bus class $\mathcal{B}$ is realisable *if and only if there exists a parameter interpretation $\mathcal{V}_p$ which satisfies φ and has a well-founded model.*

The preliminary assumption for realisability of a given bus class is mutual conformance of the bus predicate and the parameter constraint in the bus specification. In particular, we require that the bus predicate does not contradict the parameter constraint and vice-versa. Therefore it is reasonable to consider only such consistent bus classes. The following definitions declare the notion of a consistent bus class formally.

Definition 6.21 *Let* P *a cooperation pattern of some bus class $\mathcal{B}$ and $\mathcal{V}_p$ a parameter interpretation of $\mathcal{B}$. Further let $\mathcal{V}_v$ a variable interpretation.*

We say that a variable interpretation $\mathcal{V}_v$ is allowable by P *if and only if $\mathcal{V}_v \in Ints$ where $Ints$ is computed by executing the Algorithm 6.19 for* P *and $\mathcal{V}_p$.*

Definition 6.22 *Let $\mathcal{B}(\texttt{In}, \texttt{Out}, [\texttt{rank}, \texttt{capacity}]) = \{\varphi, \beta\}$ a bus class with φ a parameter constraint and β a bus predicate. We say that $\mathcal{B}$ is* consistent *if and only if there exists a parameter interpretation $\mathcal{V}_p$ which satisfies φ and induces for each cooperation pattern* P *of β (in terms of Algorithm 6.19) at least one variable interpretation $\mathcal{V}_v$ which is allowable by* P.

Finally, we present a theorem which states the necessary conditions under which existence of a *well-founded* model is guaranteed.

Theorem 6.23 *Let $\mathcal{B}(\texttt{In}, \texttt{Out}, [\texttt{rank}, \texttt{capacity}]) = \{\varphi, \beta\}$ a consistent bus class and let $\mathcal{V}_p$ the parameter interpretation of $\mathcal{B}$ which satisfies φ and which induces for each cooperation pattern* P *in β a variable interpretation $\mathcal{V}_v$ that is allowable by* P. *Then the model $\mathcal{B}(\mathcal{V}_p(\texttt{In}), \mathcal{V}_p(\texttt{Out}), [\mathcal{V}_p(\texttt{rank}), \mathcal{V}_p(\texttt{capacity})]$ of $\mathcal{B}$ constructed by Algorithm 6.18 is well-founded.*

Proof: The statement follows from the fact that a consistent bus class specification guarantees existence of a parameter interpretation which induces an allowable variable interpretation for each particular cooperation pattern of the specification. According to the step *(3 iv.)* of Algorithm 6.18, each allowable variable interpretation implies the particular transition relation fragment to be nonempty. Thus the respective model generated by Algorithm 6.18 must be well-founded.

Corollary 6.24 *Each consistent bus class is realisable.*

By the corollary mentioned above, we completed the discussion of those model-theoretic issues which exclusively focus on searching for necessary conditions that guarantee existence of a well-founded model for a given bus class and its parameter interpretation. To conclude, the important property which a bus class must satisfy in order to have some well-founded model is that it must be consistent. For implementation of a compiler for the bus specification language, this is a key aspect which should be taken into account. Such a compiler should be capable of checking the consistency of the specification before construction of the respective cooperation machine.

6.2.3 Expressiveness

In this section we characterise individual types of bus classes from the semantic point of view and analyse their relative expressiveness.

First of all recall the relationships among the respective languages showed in Section 6.2 of the previous chapter, realised from the syntactic point of view. All these relationships are summarised in Figure 6.5. Each of the edges in the diagram means that the lower language is a proper sublanguage of the upper language.

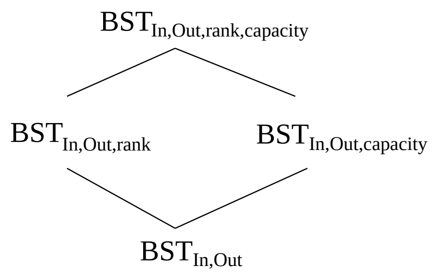


Figure 6.5: Syntactic relationships among bus specification languages

Here we study relationships among the bus specification languages from the semantic point of view. In particular, we compare relative expressiveness of all the languages with respect to what kind of cooperation machines can be specified in them up to isomorphism of respective state-transition graphs.

Before we compare the expressive power of individual bus class types, let us characterise respective semantic domains. As it has been defined in Section 6.1, the semantic domain of buses is the set CMS of cooperation machines which have the property of being finite state finitely branching labelled transition systems in which each transition label is defined as a pair of finite sets of ports (a cooperation). To identify the subset of CMS which characterises the language of each particular bus class type, we have to analyse the computation of the Algorithm 6.18.

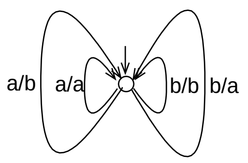
At first consider the language of memory-less rank-free bus classes $\mathrm{BST}_{\mathtt{In},\mathtt{Out}}$. The form of cooperation patterns allowed in this type of bus classes is reduced to generators in which no reference to a memory location is contained. Moreover, no rank constraint is included. As the set of memory locations F from the step (1) of Algorithm 6.18 is empty, the state space constructed in the step (2) contains only a single state. In the step (3) a finite number of self-transitions is constructed for this single state. Hence, the respective subset of CMS is characterised by some set of single-state cooperation machines. We denote this subset $\mathrm{CMS}_{\mathtt{In},\mathtt{Out}}$. Labels of the transitions are determined by interpretation of the input and output variables contained in cooperation generators. Thus each cooperation pattern generates a number of transitions which are given by all the possible interpretations of the variables with respect to the pattern structure. A typical example of this kind of specification is demonstrated in Figure 6.6.

$\mathcal{B}(\mathtt{In},\mathtt{Out}) := \{$

$\quad\mathtt{In},\mathtt{Out} \neq \emptyset$

$\quad\mathtt{i/o}$

$\}$



$\mathcal{B}(\{\mathtt{a?},\mathtt{b?}\},\{\mathtt{a!},\mathtt{b!}\})$

Figure 6.6: An example of a rank-free memory-less bus class and its model

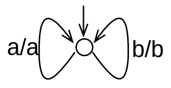
At second, let us analyse the language of ranked memory-less bus classes $\mathrm{BST}_{\mathtt{In},\mathtt{Out},\mathtt{rank}}$. Here the allowed cooperation patterns have the same kind of cooperation generators as the previous language, but they can be additionally extended with a rank constraint. Thus the construction of the cooperation machine follows the same guidelines as in the previous case with the only exception of the step (3) where a ranked cooperation pattern is treated differently. Here the number of transitions generated according to the respective pattern is potentially reduced with respect to the interpretation of the rank constraint. Hence rank constraints allow finer setting of the form of resulting cooperation machines. A typical example of this type of specification is depicted in Figure 6.7. We denote the subset of cooperation machines characterising this type of bus classes as $\mathrm{CMS}_{\mathtt{In},\mathtt{Out},\mathtt{rank}}$.

$\mathcal{B}(\mathtt{In}, \mathtt{Out}, \mathtt{rank}) := \{$

$\quad \mathtt{In}, \mathtt{Out}, \mathtt{rank} \neq \emptyset$

$\quad \mathtt{i/o} \wedge \mathtt{rank(i)} = \mathtt{rank(o)}$

$\}$



$\mathcal{B}(\{\mathtt{a?}, \mathtt{b?}\}, \{\mathtt{a!}, \mathtt{b!}\},$
$\{\mathtt{a?} \mapsto 1, \mathtt{b?} \mapsto 2, \mathtt{a!} \mapsto 1, \mathtt{b!} \mapsto 2\})$

Figure 6.7: An example of a ranked memory-less bus class and its model

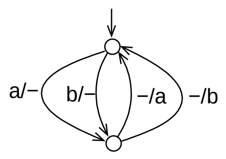
The third analysis deals with the language $\mathrm{BST}_{\mathtt{In},\mathtt{Out},\mathrm{capacity}}$ of rank-free bus classes equipped with memory. In the step (1) of the bus instance computation from classes of this type there is typically a nonempty set of memory locations generated. It implies that the state space computed in the step (2) can be nontrivial. In the step (3), transitions are generated according to the cooperation patterns and placed into the state space. Note that here cooperation patterns contain no rank constraints. The generated cooperation machine represents particular composition of semantics of all individual memory handles occurring in the specification. We denote the set of such cooperation machines which characterise this type of specifications as $\mathrm{CMS}_{\mathtt{In},\mathtt{Out},\mathtt{capacity}}$. An example of a specification of this kind is given in Figure 6.8.

$\mathcal{B}(\mathtt{In}, \mathtt{Out}, \mathtt{capacity}) := \{$

$\quad \mathtt{In}, \mathtt{Out} \neq \emptyset$

$\quad \mathtt{i{\uparrow}k/-}$
$\quad \mathtt{-/o{\downarrow}k}$

$\}$



$\mathcal{B}(\{\mathtt{a?}, \mathtt{b?}\}, \{\mathtt{a!}, \mathtt{b!}\})$

Figure 6.8: An example of a rank-free bus class with memory and its model

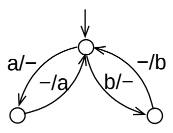
The last language which remains for analysis is $\mathrm{BST}_{\mathtt{In},\mathtt{Out},\mathtt{rank},\mathrm{capacity}}$. This language considers all the features of bus class specification. It differs from the previous language in the added possibility of extending cooperation patterns with rank constraints. Thus construction of a model of a bus class of this type follows the same guidelines as in the previous case. However, in the step (3), if a rank constraint is included in some cooperation pattern then the computation of the respective transition relation fragment is different. This difference is similar to the difference between rank-free memory-less and ranked memory-less languages. Here rank constraints allow finer setting of using of some memory locations by particular ranking of bus inputs and outputs which refer to them. An example of a typical specification of this type is depicted in Figure 6.9. Note that just cooperation machines of this kind are inexpressible in any of the others bus class languages. The set of cooperation machines which characterises this bus class type is denoted $\mathrm{CMS}_{\mathtt{In},\mathtt{Out},\mathtt{rank},\mathtt{capacity}}$.

$\mathcal{B}(\mathtt{In}, \mathtt{Out}, \mathtt{rank}, \mathtt{capacity}) :=$
$\{$

$\mathtt{In}, \mathtt{Out}, \mathtt{rank} \neq \emptyset$

$\mathtt{i}{\uparrow}@/-$
$-/\mathtt{o}{\downarrow}@$

$\}$



$\mathcal{B}(\{\mathtt{a}?, \mathtt{b}?\}, \{\mathtt{a}!, \mathtt{b}!\},$
$\{\mathtt{a}? \mapsto 1, \mathtt{b}? \mapsto 2, \mathtt{a}! \mapsto 1, \mathtt{b}! \mapsto 2\})$

Figure 6.9: An example of a ranked bus class with memory and its model

Now we follow with capturing the comparison of expressiveness of bus specification languages formally. First of all we analyse whether each particular bus specification language is expressible in its superior languages. Note that the semantics given by the Algorithm 6.18 is not modular due to the fact that each cooperation pattern is considered as a whole, i.e., its semantics is not directly constructed from semantics of its subparts. Hence the fact that a language is a sublanguage of the other language in our case does not automatically implies that the former language is expressible in the latter language. The following lemma analyses these expressibility relationship for the family of bus specification languages.

Lemma 6.25 *The following expressiveness relationships among the particular bus specification languages hold:*

1. $\mathrm{BST}_{\mathtt{In},\mathtt{Out}} \leq \mathrm{BST}_{\mathtt{In},\mathtt{Out},\mathtt{rank}}$
2. $\mathrm{BST}_{\mathtt{In},\mathtt{Out}} \leq \mathrm{BST}_{\mathtt{In},\mathtt{Out},\mathrm{capacity}}$
3. $\mathrm{BST}_{\mathtt{In},\mathtt{Out},\mathtt{rank}} \leq \mathrm{BST}_{\mathtt{In},\mathtt{Out},\mathtt{rank},\mathrm{capacity}}$
4. $\mathrm{BST}_{\mathtt{In},\mathtt{Out},\mathrm{capacity}} \leq \mathrm{BST}_{\mathtt{In},\mathtt{Out},\mathtt{rank},\mathrm{capacity}}$

Proof: We have to prove that each bus predicate encoded in the language on the left side can be encoded as a predicate of the language on the right side, and moreover, that both predicates specify the same bus instance up to isomorphism of the particular cooperation machines. Formally, for any bus predicate β_1 of the former language and any parameter interpretation $\mathcal{V}_v$ there must exist a bus predicate β_2 of the latter language which generates for the interpretation $\mathcal{V}_v$ the same model up to isomorphism.

1. At first we have to consider the language $\mathrm{BST}_{\mathtt{In,Out,rank}}$ which is an extension of the language $\mathrm{BST}_{\mathtt{In,Out}}$ such that the rank constraints are added to the syntax of the original language. According to Definition 6.7, the only syntactic difference of the two languages is in the form of cooperation patterns. In the latter language, patterns can either contain a rank constraint or not. Hence if we take a bus predicate β_1 of $\mathrm{BST}_{\mathtt{In,Out}}$, we can encode it directly in $\mathrm{BST}_{\mathtt{In,Out,rank}}$ as a predicate β_2 such that $\beta_2 \equiv \beta_1$. Now if we fix an arbitrary parameter interpretation $\mathcal{V}_p$, the computation in Algorithm 6.18 must follow in both cases the same steps. Especially in the step (4) of the nested call of Algorithm 6.19, the second possibility is always applied. Thus results of both computations must be the same cooperation machine. Additionally, the parameter interpretation $\mathcal{V}_p$ does not interpret the `rank` parameter. As the rank interpretation is never applied during the computation of β_2, there is no need to extend $\mathcal{V}_p$. Thus we have successfully finished the proof.

2. In this case, we need to encode a memory-less predicate β_1 (a predicate with no memory handles) into a language which permits use of memory handles. The encoding is realised by syntactically same predicate β_2, $\beta_2 \equiv \beta_1$. The proof follows similar guidelines as the previous one. However, here we have to carefully observe if the computation of β_2 matches the computation of β_1 in the steps (2) and (3) of the Algorithm 6.18. As we have a memory-less bus class, the interpretation of `capacity` parameter is set to 0. In this way, any possible parameter interpretation of the memory-less specification can be freely extended. Hence the computation in the step (2) must be the same. Computations do not event differ in the step (3), as the form of the cooperation patterns remains the same. Hence we have the result.

3. This situation concerns the extension of a ranked memory-free language to the full language. Also here according to Definition 6.7 the encoding β_2 of a predicate β_1 into the full language does not require any syntactic changes. By the similar steps as in the previous case we achieve the two isomorphic cooperation machines, as the particular computations in Algorithm 6.18 does not differ.

4. This situation is proved analogously to the first case.

Next we follow with analysis of inclusion relations among the sets of cooperation machines which characterise the respective bus specification languages. Before we start, let us emphasise that the symbol '$\subset$' denotes the strict subset relationship.

Definition 6.26 *Define the set of cooperation machines* $\mathrm{CMS}_{\mathtt{In,Out}} \subseteq \mathrm{CMS}$ *characterising the semantic domain of the language* $\mathrm{BST}_{\mathtt{In,Out}}$ *as the set of all models* *of any bus class $\mathcal{B} \in \mathrm{BST}_{\mathtt{In},\mathtt{Out}}$ generated for any parameter interpretation $\mathcal{V}_p$ by the Algorithm 6.18.*

Analogously define the sets of cooperation machines $\mathrm{CMS}_{\mathtt{In},\mathtt{Out},\mathtt{rank}}$, $\mathrm{CMS}_{\mathtt{In},\mathtt{Out},\mathtt{capacity}}$, and $\mathrm{CMS}_{\mathtt{In},\mathtt{Out},\mathtt{rank},\mathtt{capacity}}$ characterising the semantic domains of respective families of bus specification languages.

Lemma 6.27 *The following strict subset relationships hold:*

1. $\mathrm{CMS}_{\mathtt{In},\mathtt{Out}} \subset \mathrm{CMS}_{\mathtt{In},\mathtt{Out},\mathtt{rank}}$
2. $\mathrm{CMS}_{\mathtt{In},\mathtt{Out}} \subset \mathrm{CMS}_{\mathtt{In},\mathtt{Out},\mathtt{capacity}}$
3. $\mathrm{CMS}_{\mathtt{In},\mathtt{Out},\mathtt{rank}} \subset \mathrm{CMS}_{\mathtt{In},\mathtt{Out},\mathtt{rank},\mathtt{capacity}}$
4. $\mathrm{CMS}_{\mathtt{In},\mathtt{Out},\mathtt{capacity}} \subset \mathrm{CMS}_{\mathtt{In},\mathtt{Out},\mathtt{rank},\mathtt{capacity}}$

Proof: At first note that from Lemma 6.25 the non-strict versions of the set inclusions above directly follows. Thus it suffices to prove that each of these relationships is strict.

1. The first strict set inclusion follows from the claim that the cooperation machine from Figure 6.7 cannot be encoded in the language $\mathrm{BST}_{\mathtt{In},\mathtt{Out}}$. We follow the proof of this claim by contradiction. Suppose the cooperation machine in Figure 6.7 can be specified, up to isomorphism, in $\mathrm{BST}_{\mathtt{In},\mathtt{Out}}$. Denote this cooperation machine B. As all the transition labels of B contain a single event in both sections, all the generators in the specification must be of the form `i/o`. Moreover, the sets of inputs and outputs of B are the following:

   $$In(B) = \{\mathtt{a}?, \mathtt{b}?\} \qquad Out(B) = \{\mathtt{a}!, \mathtt{b}!\}$$

   Hence such a specification leads to the one depicted in Figure 6.6. Note that this specifications is a minimal representative of the supposed specification up to the particular setting of the bus constraint. However, the respective cooperation machine is evidently not isomorphic to B. Thus the claim holds.

2. Similarly to the previous case, this inequality follows from the claim that the cooperation machine from Figure 6.8 cannot be expressed in $\mathrm{BST}_{\mathtt{In},\mathtt{Out}}$. If we assume the converse of this claim, then we are forced to model behaviour of a buffer with a transition system which contains only a single state, which is impossible. Hence the claim holds.

3. The proof of this inequality follows the same steps as the previous one. The only distinction is that link ranking is considered here. But as link ranking is an orthogonal feature w.r.t. to the presence of memory, it does not influent the proof. Hence the inequality holds.

4. The fourth inequality follows from the claim that the cooperation machine depicted in Figure 6.9 cannot be specified in $\mathrm{BST}_{\mathtt{In,Out,capacity}}$. Denote this cooperation machine B and assume the converse of the claim. As each of the transition labels of B contains a single event in either the input or the output section, all the generators in the supposed specification must be of the respective forms. It leads us to the one of the specifications depicted in Figure 6.8, Figure 6.10, or all their consistent variations which are given by replacing the $\uparrow$ operator with $\Uparrow$ or/and the operator $\downarrow$ with $\uparrow$. Note that these specifications are minimal representatives of the supposed specification up to the particular setting of the bus constraint and up to the order of cooperation patterns. Additionally, note that the sets of inputs and outputs of B are the following:

$$In(B) = \{\mathtt{a?}, \mathtt{b?}\} \qquad Out(B) = \{\mathtt{a!}, \mathtt{b!}\}$$

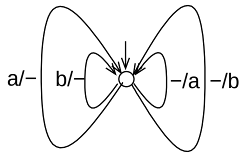
Thus the supposed specification should have the model corresponding to the one of the cooperation machines depicted in the above listed figures or its variant w.r.t. all the possible representatives. Intuitively, no of these models is isomorphic to B. Hence the claim holds.

$\mathcal{B}(\mathtt{In}, \mathtt{Out}, \mathtt{capacity}) := \{$

  $\mathtt{In}, \mathtt{Out} \neq \emptyset$

  $\mathtt{i/-}$
  $\mathtt{-/o}$

$\}$



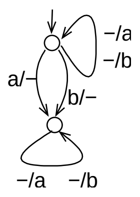
$\mathcal{B}(\{\mathtt{a?}, \mathtt{b?}\}, \{\mathtt{a!}, \mathtt{b!}\})$

$\mathcal{B}(\mathtt{In}, \mathtt{Out}, \mathtt{capacity}) := \{$

  $\mathtt{In}, \mathtt{Out} \neq \emptyset$

  $\mathtt{i{\uparrow}k/-}$
  $\mathtt{-/o}$

$\}$



$\mathcal{B}(\{\mathtt{a?}, \mathtt{b?}\}, \{\mathtt{a!}, \mathtt{b!}\})$

Figure 6.10: Some representatives of the supposed specification

Corollary 6.28 *All the expressiveness relationships stated in Lemma 6.25 are strict.*

The final corollary confirms that all the relationships depicted in Figure 6.5 holds also from the semantic point of view, and moreover, they are strict.

In the similar way as we have compared the semantic domains of bus class specification languages, we can compare the domain $\mathrm{CMS}_{\mathtt{In},\mathtt{Out},\mathtt{rank},\mathtt{capacity}}$ of full bus class specifications with the domain of all cooperation machines CMS.

Lemma 6.29 *The inclusion* $\mathrm{CMS}_{\mathtt{In},\mathtt{Out},\mathtt{rank},\mathtt{capacity}} \subset \mathrm{CMS}$ *is strict.*

Proof: The fact that each cooperation machine of $\mathrm{CMS}_{\mathtt{In},\mathtt{Out},\mathtt{rank},\mathtt{capacity}}$ is included in CMS follows directly from the Definition 6.5. Thus, only the strictness of this inclusion remains to be proved.

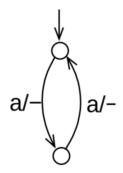


Figure 6.11: A cooperation machine not specifiable by BST

Assume the cooperation machine depicted in Figure 6.11. We show that it is impossible to encode such a cooperation machine in the language $\mathrm{BST}_{\mathtt{In},\mathtt{Out},\mathtt{rank},\mathtt{capacity}}$. Assume that the converse holds. Here follows the observation that the considered two-state cooperation machine contains a transition labelled $'\mathtt{a}/-'$ leading to the initial state from the other state. According to Algorithm 6.18, such a transition can be generated only from a cooperation generator of the form $\mathtt{i}[\uparrow, \Uparrow][\mathtt{k}, @]/-$ where the square bracket notation encodes either symbol in the respective brackets. However, according to Algorithm 6.18 the semantics of any of such generators comprises a transition leading to a state q_{xy} where x and y are determined with respect to the following forms:

- If the generator has one of the forms determined by the scheme $\mathtt{i}\uparrow[\mathtt{k}, @]/-$ then $y < \mathcal{V}_p(\mathtt{capacity})$ and x is either $\mathtt{k}$ or $\mathcal{V}_p(\mathtt{rank})(V_v(\mathtt{i}))$ where $\mathcal{V}_p$ is a parameter interpretation and $\mathcal{V}_v$ some allowed variable interpretation induced by $\mathcal{V}_p$. Note that from definition of interpretation of bus classes with memory it follows $\mathcal{V}_p(\mathtt{capacity}) > 0$. As the machine has only two states, the parameter interpretation must satisfy $\mathcal{V}_p(\mathtt{capacity}) = 1$. Hence we get $y = 0$. Now as the initial state is q_x0 because of the step (4) of the Algorithm 6.18, we have a contradiction.

- If the generator is of the form given by the scheme $\mathtt{i}\Uparrow[\mathtt{k}, @]/-$ then $y = \mathcal{V}_p(\mathtt{capacity})$ and x is either $\mathtt{k}$ or $\mathcal{V}_p(\mathtt{rank})(V_v(\mathtt{i}))$ where $\mathcal{V}_p$ is a parameter interpretation and $\mathcal{V}_v$ some allowed variable interpretation induced by $\mathcal{V}_p$. With respect to the definition of the relation T_1' in the Algorithm 6.18, the supposed transition must be a self-transition. But the considered cooperation machine has no self-transitions. Thus we have encountered a contradiction.

The previous result implies that the bus class specification language is incomplete in the sense of the capability to encode every finite cooperation machine. As the purpose of bus classes is compact parameterised description of typical communication media, we believe that the language is still expressible enough to describe the standard kinds of communication media. Roughly saying, the above mentioned expressiveness limitation is a penalty of the abstractness of bus class specifications given by the variety of considered parameters.

To conclude, let us recall that we had to compare all the bus class types from the semantic viewpoint. We measured the expressiveness of the languages with respect to what class of objects they allow to specify. There are other ways of comprehending the language expressiveness [AFV01]. We discuss the most significant measure — the universal computational power (the power of Turing machines). As our languages come under finite transition systems, they all do not have such universal power. If we permit the state space to be infinite, e.g., if we set the capacity of memory to be unbounded, then the bus specification language $\mathrm{BST}_{\mathtt{In},\mathtt{Out},\mathtt{capacity}}$, and hence also its superior language $\mathrm{BST}_{\mathtt{In},\mathtt{Out},\mathtt{rank},\mathtt{capacity}}$, will have the power of unbounded channels. A language which has the universal power must be capable of encoding of a universal two counter machine [Vaa93]. However, it will be still impossible to directly encode a universal two-counter machine into such an extended bus specification language. The problem is that it is impossible to capture the control mechanism of the machine, as the bus specification language contains no primitives for encoding of the recursive counter behaviour. However, if we add a finite state machine to control the unbounded cooperation machine, then we can intuitively get the universal power. As models of bus specifications are aimed to represent coordination models and to be used in context of some behavioural model, their unboundedness can lead to universal power of the entire specification language.

Chapter 7

VCN: Behavioural Model

The work presented in this chapter is an extended version of the results published in [Saf03] and [Saf04].

In Chapter 3 and Chapter 4, the visual notation for structural description of system architecture has been introduced, and its formal representation has been given in the form of structural terms in Chapter 5. In Chapter 6, a behavioural semantics for the coordination model of buses has been given. In this chapter, a complete behavioural model of VCN architectures is established. More specifically, to each structural term a so-called *behavioural term* is assigned providing an operational semantics for a particular VCN architecture. The behavioural model allows formal analysis of VCN architectures, i.e., it enables architectural compatibility checking. This phenomenon is introduced in the next chapter, where a specific equivalence checking framework is developed for behavioural terms.

7.1 Principles of Behavioural Model

The behavioural model of VCN is based on action-based state transition operational semantics. In this section, its basic principles are briefly introduced.

The most characteristic property of VCN structure is separation of coordination and computational layers of the design. Each network consists of some components and buses. From the behavioural point of view, components are elements encapsulating computational behaviour, while buses are elements which encapsulate coordinative behaviour. Putting both these classes of different behavioural elements together the behavioural model of entire network is established. There is a notable difference concerning the hierarchical aspects of these two classes of behavioural elements. Components allow hierarchical refinement of the design because they can encapsulate lower level coordination layers (networks), whereas buses are indivisible atomic elements.

With respect to the hierarchical structure of VCN components, the behavioural model has various levels of abstraction. At the bottom most level, semantics of leaves is defined. Lifting up one step in the hierarchy, the semantics of components is defined. At this point, the white box view of a component is transformed to the black box view. Finally the semantics of buses and the black box component semantics are put together to define semantics of entire networks.

7.1.1 Computational Layer

In Section 5.1.4, leaves have been introduced as cornerstones of the VCN computational layer. VCN leaves represent atomic computational components. They are defined as atoms in the VCN hierarchy. Each leaf is characterised by a set of all its observable events. Formally, as it has been defined in the previous chapter, there is a finite alphabet $\alpha(S) \subset ev_{\mathcal{P}}$ of events assigned to each leaf $S \in \mathrm{Leaves}$. In Section 7.2 we establish operational semantics which extends leaves with behavioural model in terms of transition systems. In this thesis, we define an operational semantics for leaves which is asynchronous in the way how the model interact with its environment. However, in [Saf02] and [SS05] we have also elaborated on the topic of synchronous semantics. The two possible models of computation can be characterised by the following aspects:

- asynchronous model

  In this setting, the chosen class of transition systems allows modelling of components with strict atomic interaction behaviour with interleaving semantics of concurrent component execution. Intuitively, each observable event of any model component occurs as a single discrete event of the entire model. The interaction with the environment is the following. The model waits (possibly infinitely long time) until the environment emits an event. If the event is emitted, the model immediately realises the computation step and changes its state. On the contrary, whenever the model is going to perform an action which emits an event, it waits until the environment is capable of receiving it. After the event is successfully emitted, the model continues its computation and changes immediately its state. This model of interaction works also among the particular components inside the model provided that each particular cooperation among any components of the model occurs as a single atomic internal event of the entire model.

- synchronous model

  Here the respective class of transition systems satisfies the character of non-strict non-atomic interaction behaviour. In this model, observable events of components in the system occur in discrete time instants. The interaction with the environment is such that the model reacts immediately to events emitted by the environment. The computation itself is considered to take zero time. This assumption is also called a zero-delay abstraction or synchronous hypothesis. Moreover, in the same sense like the model reacts to events of the environment the environment is supposed to react to events of the model, and also this is the way of interaction of particular components inside the model. There is no atomicity of particular cooperation among components in the model.

7.1.2 Coordination Layer

Considering the coordination layer, in Section 5.1.4 there have been introduced buses as atomic elements of the VCN coordination layer. Buses are closely related to other elements — links and ports. In contrast to these elements which passively represent static topologies of bus and component mutual interconnections, buses play an active role in the coordination layer. More precisely, buses represent behavioural aspects of particular coordination mechanisms. In the next section, we give buses operational semantics to capture their active role in general way.

Hierarchy of Networks

After choosing one of the two mentioned semantic models for component computation, the semantics of both VCN layers can be put together. More precisely, both the semantics of buses and leaves are combined to infer the behaviour of the entire network. The respective network transition system reflects the semantic model chosen for computation layer. Then by constructing a higher-level component (a component of a higher-level network) by network encapsulation, the behaviour of the network is lifted to define white box semantics of the higher-level component. In the scope of the higher-level network, black box semantics of the component is considered, which means that the component behaviour is abstracted with respect to the particular component interface (black box). Iterating this intuition, semantics of the entire system is inferred traversing the hierarchy from leaves to the top-most network.

7.2 Behavioural Model of Networks

In the asynchronous behavioural model, the semantic domain for VCN leaves and networks is defined in terms of traditional labelled transition systems with a single discrete event making a label of each transition. Formally, we take labelled transition systems of Definition 2.1 $\langle Q, Act, T, q_0 \rangle$

in which the alphabet Act is given as some finite subset of the set of annotated events $Act \subset \mathcal{E}^\sharp$. Event annotation is important for transition systems of networks to distinguish homonymous events of different components in a network.

To give a VCN diagram its behavioural model we need to construct a labelled transition system for the respective structural term. Note that structural terms are static in a sense that they do not consider the actual state (*configuration*) of a respective leaf or network. To assign the dynamic behavioural semantics to the static structural terms, we follow the traditional process algebraic approach of developing a structural operational semantics and introduce the agent language of *behavioural terms* to capture specification of the behavioural model. For each kind of structural terms we define a specific kind of behavioural terms which represent configurations of the respective structural term. Behavioural terms then represent directly states of the labelled transition system which defines the behavioural model of a particular structural term. This way we assign behavioural models to VCN diagrams.

In this section we define a term algebra of behavioural terms for each category of structural terms and give the respective transition system specifications.

Notation 7.1 *Formally we denote the language of (asynchronous) behavioural terms* $\mathbf{T}^a_{bhv}$ *as a union of the language* $\mathbf{LT}^a_{bhv}$ *of leaf behavioural terms and the language* $\mathbf{NT}^a_{bhv}$ *of network behavioural terms,* $\mathbf{T}^a_{bhv} \overset{\mathrm{df}}{=} \mathbf{LT}^a_{bhv} \cup \mathbf{NT}^a_{bhv}$.

Assigning of the behavioural model to structural terms is realised formally as a mapping denoted Φ^a, having the signature $\Phi^a : \mathbf{T}_{st} \to \mathbf{T}^a_{bhv}$, and defined in such a way that to every structural term there is assigned a behavioural term containing the initial state of the particular transition system (initial configuration).

7.2.1 Leaf Configurations

As leaves are cornerstones of VCN computation layer, we set up the language of behavioural terms for leaves to be abstract enough to encode a wide range of formalisms for behavioural description of computation with strict model of interaction (w.r.t. environment).

Definition 7.2 *Let $\mathcal{V}$ countable set of leaf term variables. Define the set of* asynchronous leaf behavioural terms (leaf configurations) $\mathbf{LT}^a_{bhv}$ *as the least set of finite guarded terms having the following form:*

- $\mathtt{A} ::= \tau \parallel \mathtt{w!} \parallel \mathtt{r?}$ *is* atomic statement
  where $w! \in \mathcal{E}_{\mathcal{W}}$ *and* $r? \in \mathcal{E}_{\mathcal{R}}$

- $\mathtt{S} ::= \mathtt{nil} \,\|\, \mathtt{A};\ \mathtt{S} \,\|\, \mathtt{S} + \mathtt{S} \,\|\, \mathtt{X} \,\|\, \mathrm{fix}(\mathtt{X} \overset{\mathrm{df}}{=} \mathtt{S})$ *is* asynchronous leaf behavioural term *where* $\mathtt{X} \in \mathcal{V}$.

We say that a term $\mathtt{S}$ *is* guarded *iff in each its subterm of the form* $\mathrm{fix}(\mathtt{X} \overset{\mathrm{df}}{=} \mathtt{S}')$ *each occurrence of* $\mathtt{X}$ *in the term* $\mathtt{S}'$ *has the context* $\mathtt{A};\ \mathtt{X}$ *for some atomic statement* $\mathtt{A}$.

A term $\mathtt{S}$ *is* closed *iff each occurrence of a variable* $\mathtt{X} \in \mathcal{V}$ *in any subterm* $\mathtt{S}'$ *of* $\mathtt{S}$ *has the context* $\mathrm{fix}(\mathtt{X} \overset{\mathrm{df}}{=} \mathtt{S}')$.

$$\frac{\mathtt{S}[\mathrm{fix}(\mathtt{X} \overset{\mathrm{df}}{=} \mathtt{S})/\mathtt{X}] \xrightarrow{e}_S \mathtt{S}'}{\mathrm{fix}(\mathtt{X} \overset{\mathrm{df}}{=} \mathtt{S}) \xrightarrow{e}_S \mathtt{S}'} \qquad \frac{}{\mathtt{w}!;\ \mathtt{S} \xrightarrow{w!}_S \mathtt{S}}$$

$$\frac{\mathtt{S}_1 \xrightarrow{e}_S \mathtt{S}_1'}{\mathtt{S}_1 + \mathtt{S}_2 \xrightarrow{e}_S \mathtt{S}_1'} \qquad \frac{}{\mathtt{r}?;\ \mathtt{S} \xrightarrow{r?}_S \mathtt{S}}$$

$$\frac{\mathtt{S}_2 \xrightarrow{e}_S \mathtt{S}_2'}{\mathtt{S}_1 + \mathtt{S}_2 \xrightarrow{e}_S \mathtt{S}_2'} \qquad \frac{}{\tau;\ \mathtt{S} \xrightarrow{\tau}_S \mathtt{S}}$$

Table 7.1: Operational semantics of asynchronous leaf behavioural terms

From the same reasons as in CCS and in other De Simone languages, the requirement of guardedness ensures recursive definitions to have unique solution. The rules in Table 7.1 define the operational semantics for the language $\mathbf{LT}^a_{bhv}$ of asynchronous leaf behavioural terms.

By Definition 7.2 we have defined an abstract language for description of VCN behavioural layer. We believe our definition is abstract enough to encode a satisfactorily large variety of computational models where the level of abstraction which is typical for architectural specification. Each potential candidate for computational model incorporable into VCN is arbitrary language semantically compatible with this setting of $\mathbf{LT}^a_{bhv}$. For details on specific properties of such a compatibility, we refer the reader to Section 7.2.6.

In the following definition, the set of all observable events which happen during computation of some asynchronous leaf term is defined. This set will be used further in this section for capturing of leaf alphabets.

Definition 7.3 *Let* $\mathtt{S} \in \mathbf{LT}^a_{bhv}$. *Define the set of* observable events *of* $\mathtt{S}$, *denoted* $events(\mathtt{S})$, *by induction:*

- $events(\mathtt{nil}) \overset{\mathrm{df}}{=} \emptyset$
- $events(\mathtt{w}!;\ \mathtt{S}) \overset{\mathrm{df}}{=} \{w!\} \cup events(\mathtt{S})$
- $events(\mathtt{r}?;\ \mathtt{S}) \overset{\mathrm{df}}{=} \{r?\} \cup events(\mathtt{S})$
- $events(\mathtt{S} + \mathtt{S}') \overset{\mathrm{df}}{=} events(\mathtt{S}) \cup events(\mathtt{S}')$

- $events(\text{fix}(\mathtt{X} \overset{\text{df}}{=} \mathtt{S})) \overset{\text{df}}{=} events(\mathtt{S})$

Note that the τ event is not included in the set of observable events of a leaf in the definition above. This is a natural requirement which goes with the intention to keep internal actions invisible by the environment.

Finally, we have to show how the operational semantics defined above is related with the notion of leaf structural terms. More precisely, it remains to be specified what kind of leaf configurations can be assigned to leaf structural terms to define their asynchronous behavioural model. In particular, a closed term representing the initial configuration of the respective leaf computation is assigned to a leaf structural term. This initial configuration represents the behavioural model — a particular leaf transition system given by the operational semantics defined above. Formally, we realise this assignment by the semantic mapping Φ^a defined for leaves as the function

$$\Phi^a : \text{Leaves} \rightarrow \mathbf{LT}^a_{bhv}$$

such that for each $S \in \text{Leaves}$ we assign $\Phi^a(S) = \mathtt{S}$ if and only if

- $\mathtt{S} \in \mathbf{LT}^a_{bhv}$ is a closed term,
- $\alpha(S) = events(\mathtt{S})$.

Notation 7.4 *By convention, the initial behavioural term of the leaf S is denoted* $\mathtt{S}$.

7.2.2 Component Configurations

In the previous chapter we have defined the auxiliary kind of structural terms, so-called component terms. Their main purpose is to glue together two subsequent coordination layers in the VCN hierarchy. From the behavioural point of view, the purpose of the component element is transformation of the component white box view into the black box view. As it has been stated in Section 4.3, this transformation is determined by the component gate and the component interface. Both these constructs are encapsulated in the component structural term together with the component body (a leaf or a network term).

To capture operational semantics of the component black box view, we introduce the auxiliary notion of (asynchronous) *component behavioural terms* (component configurations). The set of these terms is denoted $\mathbf{CT}^a_{\text{bhv}}$. Component terms have a specific role in the hierarchy of behavioural terms. They serve as basic elements for construction of network behavioural terms. Similarly as in the case of component structural terms, the set of component behavioural terms is not included in the overall set of behavioural terms $\mathbf{T}^a_{bhv}$. Intuitively, a component behavioural term is a three-tuple containing a behavioural term of the component body (a leaf or network configuration), a component interface, and a gate. The former element represents dynamic information and the other two elements carry static information about the component structure. Because the structural information is important in order to establish a transition relation among component configurations, it is included there in spite of its static nature (neither the gate nor the interface change during the component computation).

Definition 7.5 *For each component $C = \langle S, I, G\rangle \in \mathbf{CT}_{\mathrm{st}}$ and each behavioural term $\mathtt{S} \in \mathbf{T}^a_{bhv}$ define the* component behavioural term $\langle \mathtt{S}, I, G\rangle \in \mathbf{CT}^a_{\mathrm{bhv}}$. *Further define* component semantic mapping $\Phi^a_C : \mathbf{CT}_{\mathrm{st}} \to \mathbf{CT}^a_{\mathrm{bhv}}$ *as a function satisfying:*

$$\Phi^a_C(\langle S, I, G\rangle) \overset{\mathrm{df}}{=} \langle \mathtt{S}, I, G\rangle$$

where $\mathtt{S} = \Phi^a(S)$.

Notation 7.6 *The members of $\mathbf{CT}^a_{\mathrm{bhv}}$ will be denoted $c_1, c_2, ...$ Moreover, the set of configurations of a particular component C is denoted $Q(C)$, $Q(C) \subset \mathbf{CT}^a_{\mathrm{bhv}}$.*

7.2.3 Network Configurations

To represent the behavioural model for the set of network structural terms NT_{st}, we introduce the language of network configurations. Intuitively, a network configuration is determined by a vector of current component configurations and a vector of current bus states. Additionally, the structural information comprised in the link relation is also considered as a part of each network configuration to simplify inference of network transition systems.

Definition 7.7 *Define the language of* network behavioural terms (network configurations) *as the least set of terms having the following syntax:*

$$\mathtt{N} := \langle\langle c_1, ..., c_n\rangle, \langle q_1, ..., q_m\rangle, L\rangle$$

where

- $n, m \in \mathcal{N}$
- $c_1, .., c_n \in \mathbf{CT}^a_{\mathrm{bhv}}$
- $q_1 \in Q(B_1), .., q_m \in Q(B_m)$ *for some* $B_1, ..., B_m \in \mathrm{Buses}$
- $L \subseteq (\bigcup_{i=1}^m In(B_i) \cup \bigcup_{i=1}^m Out(B_i)) \times \{B_1, ..., B_m\}$ *a link relation*

Define the semantic mapping Φ^a for each network term $N = \langle\langle C_1, .., C_n\rangle, \langle B_1, .., B_m\rangle, L, Lrank\rangle \in \mathbf{NT}_{\mathrm{st}}$ in the following way:

$$\Phi^a(N) \overset{\mathrm{df}}{=} \langle\langle\Phi^a_C(C_1), ..., \Phi^a_C(C_n)\rangle, \langle\Phi_B(B_1), ..., \Phi_B(B_m)\rangle, L\rangle$$

Note 7.8 *Similarly as in the case of component configurations, we include the static information concerning the link relation in network behavioural terms. This information is important for construction of the transition relation over network configurations. However, in cases when the information about links is clear from the context or is not important at all, we will write simply $\langle\langle c_1, ..., c_n\rangle, \langle q_1, ..., q_m\rangle\rangle$ instead of $\langle\langle c_1, ..., c_n\rangle, \langle q_1, ..., q_m\rangle, L\rangle$.*

It is worth discussing the fact that in contrary to link relation we have not considered link ranks as a part of a network configuration. However, omitting of this structural information is absolutely correct, because presence of link ranks is necessary only for generation of bus instances. For building of the network behavioural model we use just these pregenerated bus instances (finite coordination machines).

The following notation is established due to formal capturing of specific forms of network configurations. More precisely, before we introduce SOS rules for network transition systems, we say how we formally write a network configuration which differs from another one in some particular component states or in a bus state.

Notation 7.9 *Let $\mathtt{N} = \langle\langle c_1, .., c_n\rangle, \langle q_1, ..., q_m\rangle\rangle$ a network configuration. By the notation $\mathtt{N}[c_i := c']$ denote the network configuration which differs from $\mathtt{N}$ only in its ith component configuration, so that the component configuration c_i is replaced with c'_i. Formally, $\mathtt{N}[c_i := c'] \overset{\mathrm{df}}{=} \langle\langle c_1, .., c_{i-1}, c', c_{i+1}, .., c_n\rangle, \langle q_1, .., q_m\rangle\rangle$.*

Similarly, by the notation $\mathtt{N}[q_j := q']$ denote the network configuration which differs from $\mathtt{N}$ only in its jth bus state, so that the bus state q_j is replaced with q'. Formally, $\mathtt{N}[q_j := q'] \overset{\mathrm{df}}{=} \langle\vec{c}, \langle q_1, .., q_{j-1}, q', q_{j+1}, .., q_m\rangle\rangle$.

For some $\theta \subseteq \{1, ..., n\}$ denote $\mathtt{N}[\bigwedge_{i\in\theta} c_i := c'_i]$ the network configuration which differs from $\mathtt{N}$ in all component positions included in θ, so that each component configuration c_i for some $i \in \theta$ is replaced with a new configuration c'_i. Moreover, the network configuration which differs from $\mathtt{N}$ in all component positions included in θ and additionally in jth state of the bus B_j which has moved to state q' is denoted $\mathtt{N}[\bigwedge_{i\in\theta} c_i := c'_i,\ q_j := q']$.

By convention, the initial behavioural term of the network N will be denoted $\mathtt{N}$.

7.2.4 Structural Operational Semantics

In this section, the transition system specification of the VCN asynchronous behavioural model is given. More specifically, two layers of transition systems are defined by a set of SOS rules specifying the respective transition relation — a transition system over component configurations and a transition system over network configurations. The former treats lifting of leaf and network transitions to the component (black box) level. The latter combines such local component transitions to form a global network transition (which can be further lifted to the higher component level w.r.t. the VCN hierarchy).

To establish some of the higher level transitions from a transition leading from a particular lower level configuration, we sometimes need to look for information about what labels appear on other transitions enabled from that lower level configuration. To this end, we define for each configuration t of arbitrary kind a set $xev(t)$ of all events appearing on transitions which evolve from t (the acronym xev abbreviates "next events").

Definition 7.10 *For an arbitrary behavioural term t of any kind define $xev(t)$ the set of all events which can occur on all single transitions starting from t, $xev(t) \overset{\mathrm{df}}{=} \{e \parallel \exists t'.\, t \overset{e}{\rightarrow} t'\}$ where $'\rightarrow'$ represents the relevant transition relation for the particular class of behavioural terms.*

Now we present the individual SOS inference rules. We start with rules for component configurations. The first rule defines the operational semantics of the bottom most components. Let $C = \langle S, I, G\rangle \in \mathbf{CT}_{\mathrm{st}}$ a component structural term, and $S \in \mathrm{Leaves}$ a leaf term. The operational semantics of C is defined by the transition relation $'\rightarrow_C'$ over the relevant component behavioural terms which is given by the inference rule:

$$(1)\quad \frac{\mathtt{S} \overset{e}{\rightarrow}_S \mathtt{S}'}{\langle \mathtt{S}, I, G\rangle \overset{gate_G(e)}{\longrightarrow}_C \langle \mathtt{S}', I, G\rangle}$$

Note 7.11 *The rule* (1) *is correct with respect to the gate semantics, because in the case of a leaf the type of all the gate mappings is implicitly $\perp$ and their character is one-to-one (see Notation 5.6).*

Next rule lifts leaf internal transitions to component internal transitions.

$$(2)\quad \frac{\mathtt{S} \overset{\tau}{\rightarrow}_S \mathtt{S}'}{\langle \mathtt{S}, I, G\rangle \overset{\tau}{\longrightarrow}_C \langle \mathtt{S}', I, G\rangle}$$

Let now $C = \langle N, I, G\rangle \in \mathbf{CT}_{\mathrm{st}}$ be a component structural term built over a network term $N = \langle\langle C_1, C_2, ..., C_n\rangle, \langle B_1, ..., B_m\rangle, L, Lrank\rangle \in \mathbf{T}_{st}$. The operational semantics of C is more complicated in this case because the various types of gates have to be considered. More specifically, a gate has a crucial influence on how the respective network transitions can be lifted to higher level component transitions. Formally, this situation is treated by the following four rules:

$$(3)\quad \frac{\mathtt{N} \overset{e^{\sharp i}}{\rightarrow}_N \mathtt{N}'}{\langle \mathtt{N}, I, G\rangle \overset{gate_G(e^{\sharp i})}{\longrightarrow}_C \langle \mathtt{N}', I, G\rangle} \quad \left[\ type(gate_G(e^{\sharp i})) \in \{\bot, +\}\ \right]$$

This rule (3) treats the case when the gate is of the trivial or the alternative type. In this situation there is a single network transition in the premise of the rule which is lifted to the component transition with the event relabelled with respect to the relevant gate specification.

$$(4)\quad \frac{\forall e^{\sharp i} \in dom(g).\, \mathtt{N} \overset{e^{\sharp i}}{\rightarrow}_N \mathtt{N}'_i}{\langle \mathtt{N}, I, G\rangle \overset{p}{\longrightarrow}_C \langle \mathtt{N}[\bigwedge_{i\in\theta} c_i := c'_i], I, G\rangle} \quad \left[\begin{array}{c} g \in map_G,\ ran(g) = \{p\} \\ type_G(g) = \times \\ \theta \overset{\text{df}}{=} \{i \parallel \exists e \in \mathcal{E}. e^{\sharp i} \in dom(g)\} \end{array}\right]$$

The rule (4) deals with the situation when the component contains a synchronous gate. In this case each instance of this rule contains finite number of network transitions (just those the labels of which are in the domain of the gate). The universal quantifier in the premise ensures that the component transition is realised only in the case when *all* the components in the network which are connected to the gate G can evolve from their actual configurations under the respective gated event. This behaviour conforms to the specification of synchronous gate described in Chapter 4. The resulting higher-level component transition then controls the change of the attached network configuration. More particularly, the respective network configuration included in the source component configuration changes all its component configurations related with the gate. This way, all transitions of the respective lower-level components are synchronised in a single atomic higher-level component transition.

(5)

$$\frac{\forall e^{\sharp i} \in \mathcal{M}_g.\, \mathtt{N} \overset{e^{\sharp i}}{\rightarrow}_N \mathtt{N}'_i}{\langle \mathtt{N}, I, G\rangle \overset{p}{\longrightarrow}_C \langle \mathtt{N}[\bigwedge_{i\in\theta} c_i := c'_i], I, G\rangle} \quad \left[\begin{array}{c} g \in map_G,\ ran(g) = \{p\} \\ type_G(g) = \cup \\ \mathcal{M}_g \overset{\text{df}}{=} \{e^{\sharp i} \in dom(g) \parallel e^{\sharp i} \in xev(\mathtt{N})\} \\ \theta \overset{\text{df}}{=} \{i \parallel e^{\sharp i} \in \mathcal{M}_g\} \end{array}\right]$$

The rule (5) treats the situation when the gate is of the universal type. The principle is analogous to the previous rule. The only difference is that here the synchronisation controlled by the universal gate G comprises the maximal set, denoted $\mathcal{M}_g$, of gated component actions which are enabled in the current network configuration $\mathtt{N}$. The maximality is given directly by the definition of $\mathcal{M}_g$ in the third side condition. This way, properties of the universal gate stated in Chapter 4 are satisfied.

The following rule is the last rule which is responsible for lifting of network transitions to higher-level component transitions. In particular, lifting of network internal τ-transitions to component τ-transitions is realised by this rule.

$$(6)\quad \frac{\mathtt{N} \overset{\tau}{\to}_N \mathtt{N}'}{\langle \mathtt{N}, I, G\rangle \overset{\tau}{\to}_C \langle \mathtt{N}', I, G\rangle}$$

The next four rules specify transition systems over network configurations. In other words, these rules capture the behavioural model of a network. More specifically, transitions of components in the network are taken as premises and are combined to form the resulting transition of the entire network.

The first rule of this group is responsible for interleaving of component actions which are not involved in any gate or bus synchronisation in a network. More precisely, each action which occurs on a free port of some component is lifted to a single action of the entire network.

$$(7)\quad \frac{c_i \overset{e}{\to}_C c'}{\mathtt{N} \overset{e^{\sharp i}}{\to}_N \mathtt{N}[c_i := c']} \quad \left[\begin{array}{l} \mathtt{N} = \langle\langle c_1, .., c_i, .., c_n\rangle, \vec{q}, L\rangle \\ e^{\sharp i} \in \mathcal{E}(freeports(C_i, L)) \end{array}\right]$$

As internal τ-transitions may happen at arbitrary level of VCN hierarchy, they must be captured also at the component level. By the following rule, each internal transition of a component is lifted to an internal transition of the network in which the respective component is included.

$$(8)\quad \frac{c_i \overset{\tau}{\to}_C c'}{\mathtt{N} \overset{\tau}{\to}_N \mathtt{N}[c_i := c']} \quad \left[\ \mathtt{N} = \langle\langle c_1, .., c_i, .., c_n\rangle, \vec{q}, L\rangle\ \right]$$

Finally we focus on synchronisation of components in a network. More specifically, we define a synchronisation rule which takes as a premise a particular cooperation of some bus in a network and a particular set of component transitions chosen w.r.t. a network link relation. Moreover, such a rule results in a single τ-transition at the network level denoting the internal synchronisation of relevant component transitions. Each such a synchronisation is considered as an atomic transition at the level of network transitions and labelled similarly as in CCS by the internal τ-event.

To establish the respective SOS rule for such synchronisations, at first we have to treat the semantics of cooperations because cooperations initiate and control synchronisation of components in a network. Note that there can be many cooperations which can be realised in a particular network configuration. Thus we need to choose one of them. Properly speaking, we need some classification of cooperations relative to a particular network configuration. As we have stated in Chapter 4, only cooperations which contain events of transitions evolving from current component configurations can be considered. Moreover, from such cooperations only those which contain maximal number of component transitions in the specific network configuration can be realised. To capture both the former and the latter phenomenon, we introduce the notion of so-called *enabled cooperation* and *maximal enabled cooperation*.

Definition 7.12 *Let* $\mathtt{N} = \langle\langle c_1, ..., c_n\rangle, \langle q_1, ..., q_m\rangle\rangle$ *a network configuration. Further let* B_j *for some* $j \in \{1, ..., m\}$ *some bus of* N *and* q_j *the current state of* $cm(B_j)$. *Define the* set of cooperations enabled for q_j in $\mathtt{N}$, *written* $enabled(\mathtt{N}, q_j)$, *as the set of cooperations* $enabled(\mathtt{N}, q_j) \subset \mathrm{Coops}$ *satisfying:*

- $\langle W/R\rangle \in enabled(\mathtt{N}, q_j)$ *if and only if both of the following conditions hold:*

  1. *There exists a transition* $q_j \overset{W/R}{\rightarrow} q'$ *for some* $q' \in Q(B_j)$.
  2. *Whenever* $e^{\sharp_i} \in \mathcal{E}^\sharp(W)$ *or* $e^{\sharp_i} \in \mathcal{E}^\sharp(R)$ *for some* $i \in \{1, ..., n\}$ *then* $e^{\sharp_i} \in xev(c_i)$.

We say that $\langle W/R\rangle$ *is* maximal enabled cooperation for q_j in $\mathtt{N}$ *and write* $maxenabled(\langle W/R\rangle, \mathtt{N}, q_j)$ *if and only if* $\langle W/R\rangle \in enabled(\mathtt{N}, q_j)$ *and for each* $\langle W'/R'\rangle \in enabled(\mathtt{N}, q_j)$ *the following condition holds:*

$$(W' \subseteq W \wedge R' \subseteq R) \vee (W' \cap W = \emptyset \wedge R' \cap R = \emptyset)$$

Note 7.13 *Note that for any network configuration* $\mathtt{N}$ *and any of its bus states* q_j *it holds that* $\langle\emptyset/\emptyset\rangle \in enabled(\mathtt{N}, q_j)$ *just if there exists an internal bus transition* $q_j \overset{\emptyset/\emptyset}{\rightarrow} q'$ *for some* $q' \in Q(B_j)$. *In other words, if an internal cooperation occurs in the label of some transition leading from a particular state of a cooperation machine then it is automatically enabled for that state.*

Using the notion of maximal enabledness, we can now establish the most complicated rule of VCN asynchronous behavioural model specification.

$$(9)\quad \frac{q_j \overset{W/R}{\rightarrow}_{B_j} q'_j \quad \forall e^{\sharp_i} \in \mathcal{M}_{B_j} .\, c_i \overset{e}{\rightarrow}_C c'_i}{\mathtt{N} \overset{\tau}{\rightarrow}_N \mathtt{N}[\bigwedge_{i\in\theta} c_i := c'_i, q_j := q'_j]} \quad \left[\begin{array}{c} \mathtt{N} = \langle\langle c_1, .., c_n\rangle, \langle q_1, .., q_j, .., q_m\rangle\rangle \\ maxenabled(\langle W/R\rangle, \mathtt{N}, q_j) \\ \mathcal{M}_{B_j} \overset{\mathrm{df}}{=} \mathcal{E}(W) \cup \mathcal{E}(R) \\ \theta \overset{\mathrm{df}}{=} \{i \parallel e^{\sharp_i} \in \mathcal{M}_{B_j}\} \end{array}\right]$$

This rule implements the intuition about the meaning of cooperations which has been described in Chapter 4. The first premise of the rule denotes the coordination machine transition which is supposed to control the respective atomic coordination action determined by the particular cooperation. The universally predicated premise on the right represents the particular finite number of premises all of which determine the respective transitions of components participating in the coordination action. In the side condition of the rule, the assumption of maximal enabledness is imposed on the respective cooperation. This way, the choice of the right cooperation from all the cooperations enabled in the particular bus state is achieved. More precisely, from the set of all the cooperations enabled in the particular bus state a tighter set containing only maximal cooperations is considered. From that set one particular cooperation is then chosen nondeterministically (all "maximal" instances of the rule are valid and hence any of them can be applied).

Note 7.14 *It is worth discussing how possible internal (empty) cooperations are treated by the previous rule. More specifically, we focus on the preemptive power of this kind of cooperations. If in a particular bus state some non-internal cooperations are enabled and so is also a $\langle\emptyset/\emptyset\rangle$-cooperation, then the expected meaning of this situation is internally non-deterministic choice of evolving the internal cooperation or one of the non-internal cooperations.*

Formally, according to the definition of maximal enabledness, the internal $\langle\emptyset/\emptyset\rangle$-cooperation is a maximal enabled cooperation only if there is no other cooperation enabled in the respective bus state. We consider this feature of internal cooperation exclusion because of making the VCN behavioural model (i.e., the SOS rules) human readable. In such a setting, to capture the expected preemptive power of internal cooperations, we introduce an explicit SOS rule which separately treats the situation when some bus is changing its state by performing an internal action.

$$(10)\quad \frac{q_j \overset{\emptyset/\emptyset}{\rightarrow}_{B_j} q_j'}{\mathtt{N} \overset{\tau}{\rightarrow}_N \mathtt{N}[q_j := q_j']}$$

The intuitive purpose of this rule is to transform occurrence of an empty cooperation $\langle\emptyset/\emptyset\rangle$ in some bus B_j to an internal network level action τ. However, in the case when $enabled(\mathtt{N}, q_j) = \{\langle\emptyset/\emptyset\rangle\}$ for some $\mathtt{N}$ and q_j, this rule coincides with the respective instance of the previous rule.

To sum up, this section gives ten rules specifying the VCN asynchronous behavioural model. These rules comprise the specification of component transition systems as well as network transition systems. Both kind of transition systems interleave in layers traversing the VCN hierarchy from the bottom-most components to the top-most network of a particular VCN model. That way, the behavioural model reflects the recursive definition of VCN structural terms (Definition 5.27).

7.2.5 Properties of the Behavioural Model

Herein we analyse some properties of the rules declared above. In particular, we focus on the rule format with respect to congruence property of weak bisimulation equivalence, as we use this kind of equivalence in the next chapter for architectural compatibility analysis of asynchronous VCN specifications.

It is worth noting that the format of the rules (4) and (5) does not fit the requirements of any congruence format for weak bisimulation. The reason for that is full use of so-called implicit variable copying [UP97]. In particular, let us consider for example the rule (4). The variable $\mathtt{N}$ representing a network behavioural term is contained in every premise generated by the universal quantifier which ranges over the domain of the respective gate. Moreover, this variable is also contained in the source and in the target of the rule. Hence the variable $\mathtt{N}$ is being fully implicitly copied, which implies that the weak bisimulation equivalence might be not compositional with respect to the asynchronous semantics of VCN. In the next chapter we show by a counter example that a situation in which the congruence property is violated exactly exists for both rules. The question is, if such a complicated rule format is necessary for implementing semantics of synchronous and universal gates. However, if we relaxed the implicit copying and defined any rule for this kinds of gates in the style of the rule (9) which does not suffer from the mentioned problem and let the gates to be sensed already at the network level, we would violate the required feature of gates to be applied at the level of component encapsulation. Next, if we defined the rules to infer semantics of component encapsulation directly from semantics of components (both the conclusion and premises would be component configuration transitions), then the basic assumption of the general panth rule format which requires that the source contains only one function symbol would be violated [AFV01]. In particular, in such a case the source would contain the operator of network composition nested inside the component encapsulation operator. We did not find any other possibility how to implement the gate semantics without the use of full copying, thus we believe it is an unavoidable property which is characteristic for this kind of component encapsulation.

Concerning formats of the other rules of network and component transition system specification, it is worth noting that representation of the side condition (i.e., the assumption of maximal enabledness) of the rule (9) has to necessarily include negative premises, while all the other rules avoid of negative premises. As we suppose the cooperation machines to be finite, i.e., finitely branching, each instance of the rule (9) must be finite. The reason for that is finiteness of the set $enabled(\mathtt{N}, q_j)$ (Definition 7.12) for any network configuration and any state of the considered cooperation machine. Hence the side condition expands into a union of finite sets of negative transitions where each set represents the condition providing that the respective enabled superset of the considered cooperation is not enabled. Formally, if we mark H the set of premises of a particular instance of the rule (9), then we get:

$$\begin{aligned} H \equiv \{q_j \overset{W/R}{\rightarrow} q_j'\} \cup \textstyle\bigcup_{p^{\sharp i} \in W \cup R} \mathtt{c_i} \overset{\mathtt{p}}{\rightarrow}_{C_i} \mathtt{c_i'} \\ \cup \bigcup_{\substack{\langle W'/R'\rangle \supset \langle W/R\rangle \\ \langle W'/R'\rangle \in enabled(\mathtt{N}, q_j)}} \{\mathtt{c_i} \overset{p}{\nrightarrow}_{C_i} \| p^{\sharp i} \in (W' \setminus W) \cup (R' \setminus R)\} \end{aligned} \tag{7.1}$$

The form of the premise H implies that the rule (9) fits the ntyft/ntyxt format [AFV01] — at first, the rule contains no predicates, at second, all positive premises have a single variable in their right-hand sides, at third, the source contains only one function symbol, and finally, right-hand sides of all positive premises contain mutually distinct variables. Note that the rule does not fit the constraints imposed by GSOS format which is a strict subformat of ntyft/ntyxt. The reason for that lies in the fact that the second union in the expression (7.1) can contain more than one negative transitions which evolve from the same particular component configuration. An example of a network behavioural model which violates this property is demonstrated in Figure 7.1 and 7.2. The bus in this model allows multicasting of an event produced by the component $C1$ to the other two components $C2$ and $C3$. The component $C3$ allows receiving of the event from the bus in two ways (by either one of the two input ports in_a and in_b). The respective instance of the rule (9) for the situation where the network is in configuration $\mathtt{N}$, denoted by darkened states, leads to the following set of premises:

$$H \equiv \{q_0 \xrightarrow{\{out^{\sharp_1}\}/\{in^{\sharp_2}\}} q_0\} \cup \{\mathtt{c}_1 \xrightarrow{\mathtt{out!}}_{C_1} \mathtt{c}_1', \mathtt{c}_2 \xrightarrow{\mathtt{in?}}_{C_2} \mathtt{c}_2'\} \cup \{\mathtt{c}_3 \overset{\mathtt{in\_b?}}{\not\to}_{C_3}, \mathtt{c}_3 \overset{\mathtt{in\_a?}}{\not\to}_{C_3}\}$$

The reason why H has just this form follows from the fact that for the considered cooperation $\{out^{\sharp_1}\}/\{in^{\sharp_2}\}$ the set of all its sup-cooperations enabled in $\mathtt{N}$ is just the following set:

$$\{\langle\{out^{\sharp_1}\}/\{in^{\sharp_2}, in\_b^{\sharp_3}\}\rangle, \langle\{out^{\sharp_1}\}/\{in^{\sharp_2}, in\_a^{\sharp_3}\}\rangle\}$$

Note that H contains two different transitions evolving from $\mathtt{c}_3$, thus the GSOS format is violated.

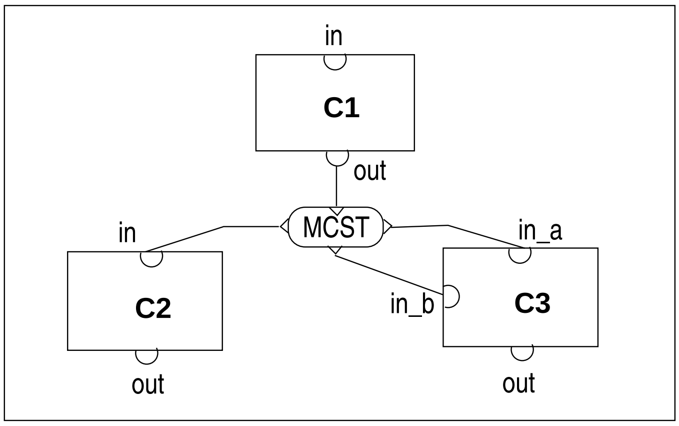


Figure 7.1: A network behavioural model which violates GSOS format

Note that the fundamental consistency requirement in definition of cooperation (Definition 5.19), stating that any two ports in each cooperation must be of different components, avoids of direct violation of the GSOS format. However, the counterexample above shows that despite the consistency requirement the GSOS format is not satisfied by the asynchronous behavioural model of VCN. Hence we cannot apply the results [Blo95] and [UP97] which imply congruence results for weak bisimulation by imposing other restrictions to GSOS format. However, as all the rules of the asynchronous behavioural model fall under the ntyft/ntyxt format, the asynchronous behavioural model of VCN is compositional with respect to strong bisimulation.

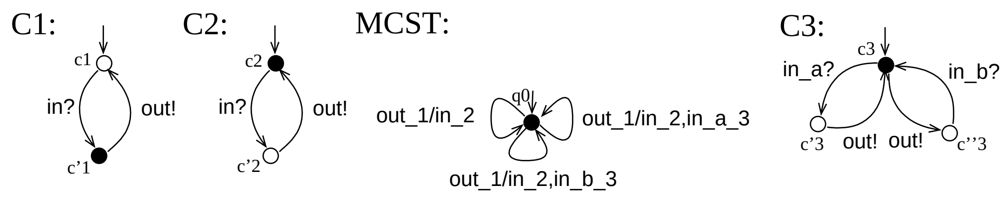


Figure 7.2: Behavioural model of components and the bus from Figure 7.1

7.2.6 Expressiveness

In this subsection we discuss some expressiveness issues of the transition system specification for the asynchronous behavioural model given above. We consider measuring of expressiveness in two aspects. The first aspect concerns the domain of structures which can be described by various languages up to strong bisimulation equivalence, whereas the second aspect asks which operations of one language are realisable in terms of the operations of other language up to strong bisimulation equivalence.

With respect to the rule (9), the structural operational semantics of networks is based on composition of two kinds of transition systems — cooperation machines and transition systems of leaf configurations, given by the rules in Table 7.1. Concerning the expressiveness issues, it is worth noting that combination of these two kinds of transition systems determines the expressive power of the entire asynchronous version of VCN. As the primary aim of VCN is description of finite state component-based architectures, and in such architectures features of connectors are typically of greater interest than features of components, we focus especially on analysis of expressive power of the coordination layer.

However, as the language of leaf configurations is a cornerstone of the VCN semantics, let us give some notes concerning the leaf configurations at first. The specification of leaf transition systems has been defined by SOS rules given in Section 7.2. All those rules in Table 7.1 have the structure which fits the requirements of the De Simone rule format with recursion. More precisely, all the rules have positive premises with different variables and the target has no multiple occurrences of variables and does not contain any of the source variables. Together with the recursion rule and the assumption of guarded recursion the language of leaf configurations can be classified as a finitary De Simone language (in terms of [AFV01]) with finite guarded recursion and hence its semantic domain corresponds to regular transition systems. This fact follows from [vG95]. This result also implies that the language of leaf configurations can be embedded into CCS [Mil89] or CSP [Hoa85] and is strictly less expressive than full versions of these process languages. Especially, in LT^a_{bhv} only finite recursion is allowed. From the point of view of the second expressiveness aspects mentioned above, LT^a_{bhv} does not contain any counterpart to parallel composition, renaming, and restriction operators of the mentioned process languages.

In consequence, we analyse the expressiveness of the coordination layer. First of all, concerning the first mentioned aspect of expressiveness, the language NT^a_{bhv} of network behavioural terms is expressible in the domain of regular transition systems. The reason for that is the fact that LT^a_{bhv} is a modular sublanguage of NT^a_{bhv} and coordination layer operators do not increase its expressiveness. This is due to finiteness aspects of all the component encapsulation and network composition rules, and the fact that also the cooperation machines used for the semantics of buses are considered to be finite (it does not matter which type of bus class language is chosen for specification of buses).

More interesting is to analyse the power of coordination layer with respect to the second expressiveness aspect. Let us consider the operator of network composition given by the rule (9) at first. Here the expressiveness of the network composition operator depends on the type of bus class specification language chosen for generating of buses. Let us denote each of the respective family of languages $\mathrm{NT}^a_{bhv}(\mathcal{B}_{\texttt{In},\texttt{Out}})$, $\mathrm{NT}^a_{bhv}(\mathcal{B}_{\texttt{In},\texttt{Out},\texttt{rank}})$, $\mathrm{NT}^a_{bhv}(\mathcal{B}_{\texttt{In},\texttt{Out},\texttt{capacity}})$, and $\mathrm{NT}^a_{bhv}(\mathcal{B}_{\texttt{In},\texttt{Out},\texttt{rank},\texttt{capacity}})$. If we take the family of languages $\mathrm{NT}^a_{bhv}(\mathcal{B}_{\texttt{In},\texttt{Out}})$, then we find out that the network composition operator of the sublanguage $\mathrm{NT}^a_{bhv}(\mathcal{B}_{\mathrm{HSK}})$ determined by the synchronous handshake bus class, defined in Chapter 6, corresponds to the parallel composition of CCS language. However, as CCS allows parallel composition to occur in the context of prefix operator (i.e., process expressions of the form $a.(P \mid Q)$), parallel composition of $\mathrm{NT}^a_{bhv}(\mathcal{B}_{\mathrm{HSK}})$ is less expressive in this aspect. Note that the inability to fork threads of parallel processes is the common restriction of static architectural languages, thus we find it natural. In this way, $\mathrm{NT}^a_{bhv}(\mathcal{B}_{\mathrm{HSK}})$ corresponds to GCCS — a graphical version of CCS which also implements such a restriction. On the contrary, a language of the family $\mathrm{NT}^a_{bhv}(\mathcal{B}_{\texttt{In},\texttt{Out}})$ which considers the synchronous atomic broadcasting bus class, denoted $\mathrm{NT}^a_{bhv}(\mathcal{B}_{\mathrm{BCST}})$, has more expressive network composition operator which cannot be realised in (asynchronous) CCS. Moreover, if we consider the language $\mathrm{NT}^a_{bhv}(\mathcal{B}_{\mathrm{MCST}})$ determined by atomic multicasting bus class, we get a network composition operator which is inexpressible even in SCCS and CSP. The reason for that is the principle of maximal enabledness which is not present in SCCS and CSP.

The family of languages $\mathrm{NT}^{a}_{bhv}(\mathcal{B}_{\texttt{In},\texttt{Out},\texttt{capacity}})$ determined by buses equipped with bounded memory does not increase expressiveness of the network composition operator with respect to any language considered above. This is due to the possibility of realising the bounded memory in terms of other primitives of the language. However, what comes here to concern is the feature of simultaneous reading from different memory locations at a single atomic computation step. Here a comparison with Linda-like coordination languages is relevant.

Concerning the language families $\mathrm{NT}^{a}_{bhv}(\mathcal{B}_{\texttt{In},\texttt{Out},\texttt{rank}})$ and $\mathrm{NT}^{a}_{bhv}(\mathcal{B}_{\texttt{In},\texttt{Out},\texttt{rank},\texttt{capacity}})$, i.e., by taking the link ranking into account, expressiveness is added to network composition in the way of what kind of cooperations can be defined. As link ranking increases expressiveness of the abstract bus specification language, the increase of expressiveness concerns the feature of greater control over addressing the ports of the cooperating components. For details see Chapter 6.

The operator of component encapsulation has two purposes. At first, it allows relabelling and hiding of events occurring inside the component body. In this sense its expressiveness is fully comparable with the relabelling operator of CCS and other process algebras. At second, it brings behavioural features to the network hierarchy. In particular, synchronous and universal gates allow to control coordination of lower level components in the network which is nested inside the component body. In this way, an event occurring outside of a component can atomically (in one step) cause cooperation in the subsystem inside the component body. This feature is not expressible by the event relabelling operators present in traditional process algebras such as CCS or CSP. In this aspect is VCN behavioural model also more expressive than the architectural languages AID and Wright, and to our best knowledge, we do not know about any architectural language which would contain such an operator.

7.3 Additional Notes

Because of the close relations between VCN and AID [RC03], we additionally compare the behavioural model of the two languages here.

In contrast to VCN, the formal semantics of AID components is defined directly in terms of labelled transition systems. Semantics of connectors is also defined as a state-transition system. Such a transition system differs from the VCN cooperation machine in the format of transition labels. Each AID transition is equipped with a trigger and an effect. The trigger is a set of actions which has the meaning of a condition imposed on the connector environment provided that each component which should participate in a particular synchronisation by evolving action a must be in the state in which the action a is enabled, and moreover, action a must be included in the trigger. The principle of matching a trigger with a set of enabled component actions is similar to VCN — the principle of maximal enabledness is applied. However, the synchronisation comprises only those actions which are included in the effect. This feature makes the AID behavioural model more expressive than VCN in this particular sense. I.e., by this inherently intricated coordination model AID allows modelling of synchronous lossy cooperations which cannot be expressed in VCN. Another application of this trigger/effect cooperation model is to enable generic definitions of AID buses. However, using the bus class specification language together with VCN link ranking which is not present in AID, the most of reasonable coordination models can be expressed in a generic way. Thus we avoided of such a complicated behavioural model as in AID.

In spite of higher expressiveness power of AID bus transitions, the AID behavioural model is defined by very complicated SOS rules (viz. Definition 3.6 in [RC03]) about which it is difficult to discuss if they satisfy the restrictions of some suitable congruence rule format. Especially, it is worth noting that in AID, the condition of so-called *independency* is introduced as a premise of the respective synchronisation rule, additionally to maximal enabledness. This condition requires at most one action of each participating component to be considered for synchronisation. Note that in VCN such a condition is implicitly satisfied by cooperations in labels of cooperation machines.

Another difference between AID and VCN is in the semantics of gates. In AID there is no counterpart to the notion of synchronous and universal gates.

Chapter 8

Architectural Interoperability Checking

This chapter contains the results which have been published in [Saf06].

The behavioural model defined in the previous chapter has been established to support bottom-up design methodology in VCN. In this chapter, a framework for automatic verification of behavioural correctness of VCN networks is developed with respect to that behavioural model. In this sense, issues regarding design-by-correctness introduced in Chapter 5 are further extended here.

To reason about correctness of VCN networks, we utilise the process-algebraic approach of behavioural equivalence based on bisimulation of behavioural models. The notion of bisimulation-based comparison of behavioural models of concurrent systems has been firstly introduced by Milner [Mil89]. Various kinds of bisimulation equivalences and simulation preorders have been studied during the last two decades [vG01]. An important kind of bisimulation equivalence is weak bisimulation. The most significant properties of this kind of behavioural equivalence are the following:

1. Weak bisimulation is not sensitive to internal behaviour of components which allows only the relevant behaviour to be considered for some particular kind of behavioural reasoning.

2. For finite state systems there are efficient algorithms with polynomial complexity [CS01b] which automatise the process of checking that two systems are equivalent w.r.t. weak bisimulation.

In order to utilise the notion of weak bisimulation for the setting of VCN, we have to deal with more intricate kind of transition labels than in traditional labelled transition systems. More specifically, for behavioural comparison of buses we customise the notion of weak bisimulation to capture cooperation machines which are cooperation-labelled transition systems. With respect to the fact that cooperations can be comprehended as sets of ports, we naturally consider two labels to be equivalent if the respective cooperations contain the same input and output ports. Additionally, we consider a transition labelled by the empty cooperation $\langle\emptyset/\emptyset\rangle$ as internal transition of cooperation machines. Hence weak bisimulation of cooperation machines, so-called *cooperation-labelled weak bisimulation*, is established w.r.t. this setting.

Concerning the congruence results, we have to deal with the fact that VCN terms differ from traditional process algebraic terms (e.g., agents of CCS or processes of CSP) in the way of how the static parallel composition operator is defined.

8.1 Behavioural Equivalence of Cooperation Machines

Cooperation machines are transition systems with cooperations appearing in transition labels. To capture the notion of behavioural equivalence of cooperation machines, we utilise the traditional (weak) bisimulation equivalence to the notion of cooperation-labelled weak bisimulation.

Notation 8.1 *Let $\gamma \in \mathrm{Coops}^*$ a sequence of cooperations. Denote $\hat{\gamma}$ the following sequences of cooperations:*

- $\hat{\gamma} \overset{\mathrm{df}}{=} \epsilon$, *if* $\gamma = \langle\emptyset/\emptyset\rangle$;
- $\hat{\gamma} \overset{\mathrm{df}}{=} \hat{\gamma}'$, *if* $\gamma = \langle\emptyset/\emptyset\rangle^* \cdot \gamma' \cdot \langle\emptyset/\emptyset\rangle^*$ *where* $\gamma' \neq \epsilon$.

Denote $\overset{\gamma}{\Rightarrow}_B$ the following sequence of succeeding transitions:

$$\overset{\gamma}{\Rightarrow}_B \overset{\mathrm{df}}{=} (\overset{\emptyset/\emptyset}{\rightarrow}_B)^* \overset{\gamma}{\rightarrow}_B (\overset{\emptyset/\emptyset}{\rightarrow}_B)^*$$

Definition 8.2 *Let B_1 and $B_2 \in \mathrm{Buses}$ with $Q(B_1)$ and $Q(B_2)$ sets of states of the respective cooperation machines.*

- *A relation $R \subseteq Q(B_1) \times Q(B_2)$ is a* (weak) cooperation-labelled bisimulation *if whenever $\langle b_1, b_2\rangle \in R$ then for each $\langle W/R\rangle \in coop(B_1) \cup coop(B_2)$ both of the following holds:*
  1. *If $b_1 \overset{\gamma}{\rightarrow}_{B_1} b_1'$ then $\exists b_2' \in Q(B_2).\, b_2 \overset{\hat{\gamma}}{\Rightarrow}_{B_2} b_2'$ and $\langle b_1', b_2'\rangle \in R$.*
  2. *If $b_2 \overset{\gamma}{\rightarrow}_{B_2} b_2'$ then $\exists b_1' \in Q(B_1).\, b_1 \overset{\hat{\gamma}}{\Rightarrow}_{B_1} b_1'$ and $\langle b_1', b_2'\rangle \in R$.*
- *We say b_1 and b_2 are* (weakly) bisimulation equivalent *and write $b_1 \approx^{cl} b_2$ if there exists a cooperation-labelled bisimulation R such that $\langle b_1, b_2\rangle \in R$.*

- *We say that* buses $B_1, B_2 \in \mathrm{Buses}$ are behaviourally equivalent *and write* $B_1 \cong^{cl} B_2$ *if and only if* $\Phi_B(B_1) \approx^{cl} \Phi_B(B_2)$.

8.2 Behavioural Equivalence of Structural Terms

To introduce the notion of behavioural equivalence of VCN behavioural terms, we employ the traditional weak bisimulation approach. The definition of weak bisimulation has been recalled in Chapter 2.

Definition 8.3 *Let* $S_1, S_2 \in \mathrm{Leaves}$ *leaves,* $C_1, C_2 \in \mathbf{CT}_{\mathrm{st}}$ *components, and* $N_1, N_2 \in \mathbf{T}_{\mathrm{st}}$ *networks.*

We say that leaves S_1 and S_2 are behaviourally equivalent, *and write* $S_1 \cong S_2$*, if and only if* $\Phi^a(S_1) \approx \Phi^a(S_2)$. *Analogously,* components $C_1, C_2 \in \mathbf{CT}_{\mathrm{st}}$ *are said to be* behaviourally equivalent, *written* $C_1 \cong C_2$*, and* networks $N_1, N_2 \in \mathbf{T}_{\mathrm{st}}$ *are said to be* behaviourally equivalent, *written* $N_1 \cong N_2$*, iff* $\Phi^a_C(C_1) \approx \Phi^a_C(C_2)$ *and* $\Phi^a(N_1) \approx \Phi^a(N_2)$*, respectively.*

In the following theorem we study the congruence property of bisimulation with respect to component encapsulation. In [AB05], the congruence property is the fundamental tool for checking of architectural interoperability.

Theorem 8.4 *Let* $I \in \mathrm{Infaces}$ *an interface, and* $G \in \mathrm{Gates}$ *a gate satisfying that each mapping in* map_G *has the type* $\perp$ *or* $+$*. The following implications hold:*

1. *For all leaves* $S_1, S_2 \in \mathrm{Leaves}$ *it holds that*

$$S_1 \cong S_2 \Rightarrow \langle S_1, I, G\rangle \cong \langle S_2, I, G\rangle$$

2. *For all networks* $N_1, N_2 \in \mathbf{T}_{\mathrm{st}}$ *it holds that*

$$N_1 \cong N_2 \Rightarrow \langle N_1, I, G\rangle \cong \langle N_2, I, G\rangle$$

Proof: *1.* We prove $Rel \overset{\mathrm{df}}{=} \{\langle\langle \mathtt{s}_1, I, G\rangle, \langle \mathtt{s}_2, I, G\rangle\rangle \parallel \mathtt{s}_1 \approx \mathtt{s}_2\}$ is a weak bisimulation.

Let $\langle \mathtt{s}_1, I, G\rangle \xrightarrow{e} \langle \mathtt{s}', I, G\rangle$ for some leaf configurations $\mathtt{s}_1, \mathtt{s}'$ of the leaf S_1 and some event $e \in \mathcal{E}$. From bisimulation equivalence of leaves S_1 and S_2 it follows that there must exist configurations $\mathtt{s}_2$ and $\mathtt{s}''$ of S_2 and a transition $\mathtt{s}_2 \overset{e'}{\Rightarrow} \mathtt{s}''$ such that $\mathtt{s}' \approx \mathtt{s}''$ where $gate_G(e') = e$. Note that G contains only one-to-one or asynchronous mappings. Thus $\langle \mathtt{s}', I, G\rangle \overset{e}{\Rightarrow} \langle \mathtt{s}'', I, G\rangle$ and $\langle\langle \mathtt{s}_2, I, G\rangle, \langle \mathtt{s}'', I, G\rangle\rangle \in Rel$. We have proved the first condition of weak bisimulation for the case of noninternal events. The second condition of bisimulation is symmetric and hence is the proof.

Let $\langle \mathtt{s}_1, I, G\rangle \xrightarrow{\tau} \langle \mathtt{s}', I, G\rangle$ for some leaf configurations $\mathtt{s}_1, \mathtt{s}'$ of the leaf S_1. Now there are two possibilities:

- $\mathtt{s}_1 \xrightarrow{\tau} \mathtt{s}'$

  In this case suppose $\mathtt{s}_2$ is a configuration of S_2 by the equivalence of S_1 and S_2 it follows that S_2 does not perform any transition from $\mathtt{s}_2$. Thus there is no transition from $\langle \mathtt{s}_2, I, G\rangle$ and we have $\mathtt{s}' \approx \mathtt{s}_2$. Hence $\langle\langle \mathtt{s}', I, G\rangle, \langle \mathtt{s}_2, I, G\rangle\rangle \in \mathit{Rel}$. The opposite simulation is symmetric.

- $\mathtt{s}_1 \xrightarrow{e} \mathtt{s}'$ where $e \in \mathcal{E}$ such that $\mathit{gate}_G(e)$ is not defined

  Here by the equivalence of S_1 and S_2 there exist configurations $\mathtt{s}_2$, $\mathtt{s}''$ and a transition $\mathtt{s}_2 \overset{\hat{e}}{\Rightarrow} \mathtt{s}''$ such that $\mathtt{s}' \approx \mathtt{s}''$. As gate_G is not defined for the event e, from the rule (1) it follows that $\langle \mathtt{s}_2, I, G\rangle \xrightarrow{\tau} \langle \mathtt{s}'', I, G\rangle$. Hence $\langle \mathtt{s}_2, I, G\rangle \overset{\hat{\epsilon}}{\Rightarrow} \langle \mathtt{s}'', I, G\rangle$ and $\langle\langle \mathtt{s}', I, G\rangle, \langle \mathtt{s}'', I, G\rangle\rangle \in \mathit{Rel}$. The opposite case can be proved symmetrically.

2. We have to prove that the following relation is weak bisimulation:

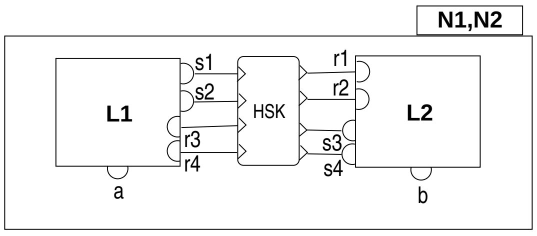
$$\mathit{Rel} \overset{\text{df}}{=} \{\langle\langle \mathtt{N}_1, I, G\rangle, \langle \mathtt{N}_2, I, G\rangle\rangle \parallel \mathtt{N}_1 \approx \mathtt{N}_2\}$$

Suppose $\langle \mathtt{N}_1, I, G\rangle \xrightarrow{e} \langle \mathtt{N}', I, G\rangle$ for some network configurations $\mathtt{N}_1, \mathtt{N}'$ of the network N_1 and some event $e \in \mathcal{E}$. All the gate mappings $g \in \mathit{map}_G$ satisfy the assumption $\mathit{type}_G(g) \in \{+, \perp\}$, hence no synchronisation can arise by employing the gate G. Let $g \in \mathit{map}_G$ a gate mapping satisfying $e \in \mathit{ran}(g)$. The supposed transition can be inferred only by the rule (2). Hence there exists i such that $\mathtt{N}_1 \xrightarrow{e'\sharp i} \mathtt{N}'$ where $\mathit{gate}_G(e'^{\sharp i}) = e$. Equivalence of N_1 and N_2 gives existence of configurations $\mathtt{N}_2$ and $\mathtt{N}''$ of N_2 and a transition $\mathtt{N}_2 \overset{\widehat{e'\sharp i}}{\Rightarrow} \mathtt{N}''$ such that $\mathtt{N}' \approx \mathtt{N}''$. Thus $\langle \mathtt{N}_2, I, G\rangle \overset{\hat{e}}{\Rightarrow} \langle \mathtt{N}'', I, G\rangle$ by the rule (2) being employed again and we have $\langle\langle \mathtt{N}', I, G\rangle, \langle \mathtt{N}'', I, G\rangle\rangle \in \mathit{Rel}$. The opposite simulation is symmetric. □

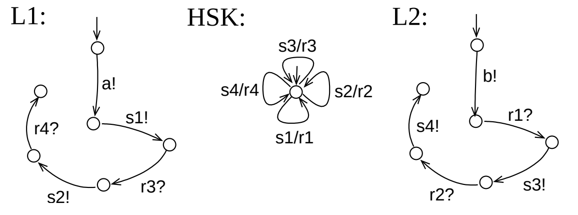
The assumption limiting the type of the gate function in the above mentioned theorem is crucial. In Figure 8.1 and Figure 8.2 there is an example which demonstrates that the claim of this theorem does not hold in general in cases when the gate contains a synchronous or universal gate mapping. In Figure 8.1 there are depicted two networks N_1 and N_2 which have the same structure (a) and differs only in definition of leaves (b, c). In spite of the difference in their leaf behaviour, both networks have the same semantics up-to weak bisimulation.

However, if we encapsulate each of the networks in a component with a synchronous gate G as is depicted in Figure 8.2 (a), the behaviour of the component created from N_1 is not equivalent with behaviour of the respective component created from N_2. More precisely, we have $N_1 \cong N_2$ and on the contrary, $\langle N_1, I, G\rangle \not\cong \langle N_2, I, G\rangle$. Hence the congruence property is violated for such network encapsulation.

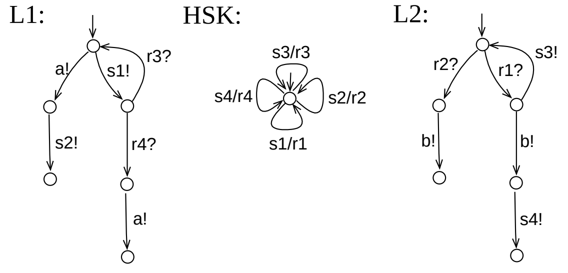
The following theorem focuses on the congruence property with respect to modular component and network interchange.



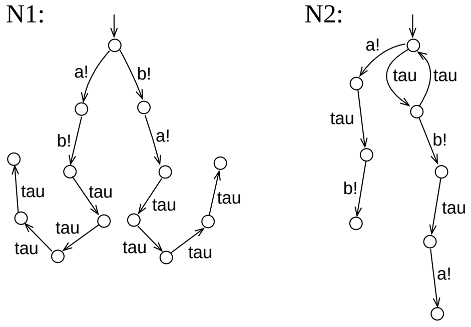
(a) shared topology of networks N_1 and N_2



(b) semantics of N_1's bus and leaves



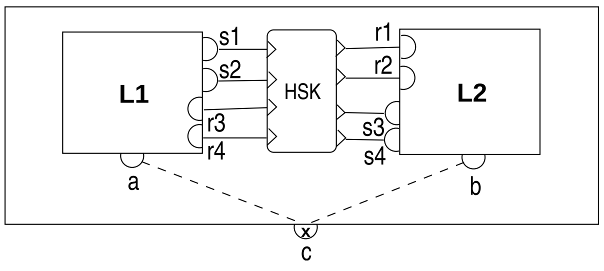
(c) semantics of N_2's bus and leaves



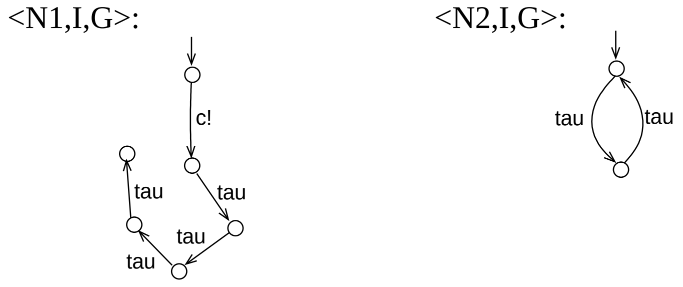
(d) semantics of N_1 and N_2 (both coincide)

Figure 8.1: Bisimilar nets which violate congruence for a $\times$-gate operator

Theorem 8.5 *Let* $N = \langle\langle C_1, .., C_i, .., C_n\rangle, \langle B_1, .., B_j, .., B_m\rangle, L, Lrank\rangle \in \mathbf{T}_{\rm st}$ *a network. The following implications hold:*



(a) applying $\times$-gate operator $G := \{L_1.a \times L_2.b \mapsto c\}$



(b) inequivalent semantics of N_1 and N_2 after application of G

Figure 8.2: Violation of congruence property for a $\times$-gate operator

- *For an arbitrary component $C' \in \mathbf{CT}_{\text{st}}$ it holds that*

$$C' \cong C_i \Rightarrow N \cong N[C_i := C']$$

- *For an arbitrary bus $B' \in \text{Buses}$ it holds that*

$$B' \cong^{cl} B_j \Rightarrow N \cong N[B_j := B']$$

Proof: Both cases are similar. The only difference is that in the case of buses the cooperation-labelled weak bisimulation is considered whereas in the case of components classic notion of weak bisimulation is applied. As it can be seen in Section 8.4.3 of this chapter, in particular in lemma 8.7, the classic weak bisimulation can be lifted to a special case of cooperation-labelled bisimulation. Therefore here it suffices to prove only the case of buses.

In particular, we prove that the relation

$$Rel \overset{\text{df}}{=} \{(\underbrace{\langle\langle \mathtt{c}_1, .., \mathtt{c}_n\rangle, \langle \mathtt{q}_1, .., \mathtt{q}_j^1, .., \mathtt{q}_m\rangle\rangle}_{\mathtt{N}_1}, \underbrace{\langle\langle \mathtt{c}_1, .., \mathtt{c}_n\rangle, \langle \mathtt{q}_1, .., \mathtt{q}_j^2, .., \mathtt{q}_m\rangle\rangle}_{\mathtt{N}_2}) \| \mathtt{q}_j^1 \approx^{cl} \mathtt{q}_j^2\}$$

is (weak) bisimulation.

1. At first suppose the transition $\mathtt{N}_1 \xrightarrow{e^{\sharp_i}} \mathtt{N}'_1$ for some configuration $\mathtt{N}'_1$ where $i \in \{1, ..., n\}$. Such a transition must be inferred according to the rule (7) and hence by the respective decomposition there exists a component configuration $\mathtt{c}'_\mathtt{i}$ such that $\mathtt{N}'_1 \equiv \mathtt{N}_1[c_i := c'_i]$ and $\mathtt{c_i} \xrightarrow{e}_{C_i} \mathtt{c}'_\mathtt{i}$. As $\mathtt{c_i}$ is included also in $\mathtt{N}_2$ it follows w.r.t. composition according to the rule (7) that $\mathtt{N}_2 \xrightarrow{e^{\sharp_i}} \mathtt{N}'_2$ and $\mathtt{N}'_2 \equiv \mathtt{N}_2[c_i := c'_i]$. As no cooperation occurs, all the buses remain in their current states, i.e., the bus B_j remains in the state $\mathtt{q}^1_j$. Hence $(\mathtt{N}'_1, \mathtt{N}'_2) \in Rel$. The respective opposite direction of the bisimulation proof is symmetric.
2. Now suppose the transition $\mathtt{N}_1 \xrightarrow{\tau} \mathtt{N}'_1$ for some configuration $\mathtt{N}'_1$. Such a transition must be inferred according to the rule (8) or (9). In the former situation the prove is similar to the previous case. In the latter situation, there must exist $k \in \{1, ..., m\}$ and $\theta \subseteq \{1, ..., n\}$ such that $\mathtt{N}'_1 \equiv \mathtt{N}_1[\bigwedge_{i\in\theta} c_i := c'_i, q_k := q'_k]$. Two cases have to be distinguished here:

- If $k = j$ then w.r.t. the rule (9) there is a transition $\mathtt{q}^1_j \xrightarrow{\gamma}_{B_j} \mathtt{q'}^1_j$ where $\gamma \doteq \{e^{\sharp_i} \,\|\, i \in \theta\}$. From $B_j \cong^{cl} B'$ follows there exists a state of the bus B' such that $\mathtt{q}^2_j \xrightarrow{\gamma}_{B_j} \mathtt{q'}^2_j$. As all the component configurations of $\mathtt{N}_1$ are included also in $\mathtt{N}_2$ and there is no more component configuration introduced in $\mathtt{N}_2$, the cooperation γ must be maximally enabled in $\mathtt{N}_2$. As additionally $\mathtt{q'}^1_j \approx^{cl} \mathtt{q'}^2_j$, it follows $(\mathtt{N}'_1, \mathtt{N}'_2) \in Rel$. The opposite situation is symmetric.

- If $k \neq j$ then by the arguments that all the components configurations involved in the cooperation are included in $\mathtt{N}_1$ and so is the bus state $\mathtt{q_k}$, we achieve the required result. Note that the B_j remains in the state $\mathtt{q}^1_j$. □

8.3 Saturated Cooperation Machines

Note that the cooperation-labelled transition relation of cooperation machines can be comprehended formally as an extension of the classical transition relation which is used for determining the operational semantics of VCN terms. By the following lemma this classical transition relation can be lifted to the format of cooperation-labelled transition relation. More precisely, we can take the operational semantics of a VCN term as a so-called saturated cooperation machine. That way we achieve uniform framework for behavioural analysis of VCN structural terms and their parts in terms of cooperation machines.

Definition 8.6 *The* cooperation machine $cm(B)$ *is called* saturated *if each transition of* $cm(B)$ *has the form* $\langle q, b, q' \rangle$ *satisfying just one of the following possibilities:*

- $b \equiv \langle \{w\}/\emptyset \rangle$ *for some* $w \in \mathcal{W}^\sharp$*;*
- $b \equiv \langle \emptyset/\{r\} \rangle$ *for some* $r \in \mathcal{R}^\sharp$*;*
- $b \equiv \langle \emptyset/\emptyset \rangle$*.*

Lemma 8.7 *Let* $t \in \mathbf{T}^a_{\mathrm{st}}$ *an arbitrary structural term. There exists a saturated cooperation machine, denoted* $cm(t)$*, which is isomorphic to the transition system* $\Phi^a(t)$ *up to the format of transition labels.*

Proof: With respect to the form of the term t, there are two cases which have to be treated separately.

- If $t \equiv \mathtt{S}$ is a leaf term $\mathtt{S} \in$ Leaves then we construct the $cm(t) \overset{\mathrm{df}}{=} \langle Q, T, q_0 \rangle$ cooperation machine in the following way. First of all, assume without loss of generality some annotation index $i \in \mathcal{N}$. Assign the initial state $q_0 \overset{\mathrm{df}}{=} \mathtt{S}$ and declare $q_0 \in Q$. The other states in Q are determined together with transitions of T by slightly modified rules of the leaf operational semantics defined in Table 7.1. The modified set of rules is the following:

$$\frac{\mathtt{S}[\mathrm{fix}(\mathtt{X} \overset{\mathrm{df}}{=} \mathtt{S})/\mathtt{X}] \overset{\gamma}{\to}_T \mathtt{S}'}{\mathrm{fix}(\mathtt{X} \overset{\mathrm{df}}{=} \mathtt{S}) \overset{\gamma}{\to}_T \mathtt{S}'} \qquad \frac{}{\mathtt{w!};\ \mathtt{S} \overset{\emptyset/\{w^{\sharp_i}!\}}{\to_T}{}_S\ \mathtt{S}}$$

$$\frac{\mathtt{S}_1 \overset{\gamma}{\to}_T \mathtt{S}_1'}{\mathtt{S}_1 + \mathtt{S}_2 \overset{\gamma}{\to}_T \mathtt{S}_1'} \qquad \frac{}{\mathtt{r?};\ \mathtt{S} \overset{\{r^{\sharp_i}?\}/\emptyset}{\to_T}{}_S\ \mathtt{S}}$$

$$\frac{\mathtt{S}_2 \overset{\gamma}{\to}_T \mathtt{S}_2'}{\mathtt{S}_1 + \mathtt{S}_2 \overset{\gamma}{\to}_T \mathtt{S}_2'} \qquad \frac{}{\tau;\ \mathtt{S} \overset{\emptyset/\emptyset}{\to}_T \mathtt{S}}$$

  Because of the format of labels in the transition relation defined by the rules above, the resulting cooperation machine is saturated. For the initial leaf configuration t comparing its two operational semantics, one given by transition system constructed by the rules of Table 7.1 and the other by the rules above, we can conclude that both transition systems are isomorphic up to format of transition labels. Hence the alternative semantics of t, $cm(t)$, is equivalent to $\Phi^a(t)$.

- When $t \equiv \mathtt{S}$ is a network configuration then the proof is achieved analogously to the previous case. Instead of the rules for leaf configurations the rules $(7 - 10)$ defined for network configurations in Section 7.2.4 are taken into account. Note that event annotation is inherent to the network transition relation $\to_N$. The rules $(7 - 10)$ are modified in the following way:

$$(7') \quad \frac{c_i \overset{\mathcal{E}(p)}{\rightarrow}_C c'}{\mathtt{N} \overset{\gamma}{\rightarrow}_L \mathtt{N}[c_i := c']} \quad \left[\begin{array}{c} \mathtt{N} = \langle\langle c_1, .., c_i, .., c_n\rangle, \vec{q}\rangle \\ p^{\sharp_i} \in freeports(C_i, L(N)) \\ \gamma \overset{\text{df}}{=} \begin{cases} \langle p^{\sharp_i}/\emptyset\rangle, \text{ if } e \in \mathcal{W}, \\ \langle \emptyset/p^{\sharp_i}\rangle, \text{ if } e \in \mathcal{R}. \end{cases} \end{array} \right]$$

$$(8') \quad \frac{c_i \overset{\tau}{\rightarrow}_C c'}{\mathtt{N} \overset{\emptyset/\emptyset}{\rightarrow}_N \mathtt{N}[c_i := c']} \quad [\ \mathtt{N} = \langle\langle c_1, .., c_i, .., c_n\rangle, \vec{q}\rangle\ ]$$

$$(9') \quad \frac{q_j \overset{W/R}{\rightarrow}_{B_j} q'_j \quad \forall e^{\sharp_i} \in \mathcal{M}_{B_j} . \, c_i \overset{e}{\rightarrow}_C c'_i}{\mathtt{N} \overset{\emptyset/\emptyset}{\rightarrow}_N \mathtt{N}[\bigwedge_{i\in\theta} c_i := c'_i, q_j := q'_j]} \quad \left[\begin{array}{c} \mathtt{N} = \langle\langle c_1, .., c_n\rangle, \langle q_1, .., q_j, .., q_m\rangle\rangle \\ maxenabled(\langle W/R\rangle, \mathtt{N}, q_j) \\ \mathcal{M}_{B_j} \overset{\text{df}}{=} \mathcal{E}(W) \cup \mathcal{E}(R) \\ \theta \overset{\text{df}}{=} \{i \parallel e^{\sharp_i} \in \mathcal{M}_{B_j}\} \end{array} \right]$$

$$(10') \quad \frac{q_j \overset{\emptyset/\emptyset}{\rightarrow}_{B_j} q'_j}{\mathtt{N} \overset{\emptyset/\emptyset}{\rightarrow}_N \mathtt{N}[q_j := q'_j]}$$

Note that the only changes in the rules comprise the labels of the transition relation being inferred. Moreover, modification of only these rules is sufficient. Rules of component configurations can be left without any modifications as we are interested only in transitions of network configurations. By the same reasons as in the previous case, $cm(t)$ is saturated and equivalent to $\Phi^a(t)$ considering isomorphism of the relevant transition systems up to the format of transition labels.

□

8.4 Architectural Interoperability Failure Freedom

8.4.1 Gate Interoperability

To achieve interoperability correctness of the behavioural model at the level of component black box view with respect to its white box view, we have to look into the way how the operational semantics of black box model is inferred from the white box model of a component. In particular, we have to focus on the rules $(1-5)$ of the operational semantics and investigate potential possibility of introducing any interoperability correctness violation.

Taking the rule (1) into account, we have to emphasise the fact that the only allowed form of gate mappings for leaf encapsulation is one-to-one mapping. Therefore, no interoperability failure can arise by invocation of this rule. Whenever the leaf performs some action e, its black box view performs the relevant action $gate_G(e)$ which denotes renaming of e or transforms e to the internal τ-action of the component black box view.

The situation of the rule (2) is trivial and no interoperability failure can arise here.

The situation of the rules $(3-5)$ treat the cases when the white box view has form of the network. In this case the black box comprises hierarchical embedding of that network to form a component which encapsulates its behaviour. In this situation, gate mappings can be of arbitrary type.

When the action of the encapsulated network is mapped to a one-to-one or asynchronous gate mapping, the relevant action of the component black box view is inferred by the rule (3). By the similar reasons as in the case of the rule (1), this situation does not introduce any interoperability failure. More precisely, in the case of one-to-one mapping the situation is just the same as in (1). In the case of asynchronous mapping, the network event which causes the relevant black box action is chosen nondeterministically from all events from the domain of the particular gate mapping which are enabled in the current network configuration.

The situation of the rule (4) is more complicated. It treats the case when the gate mapping is synchronous. In contrast to previous situations, here the potential violation of interoperability correctness can arise. Assume a simple example of a network depicted in Figure 8.3. The problem which arises here is deadlock which occurs just in the network configuration illustrated by the darkened states of the respective leaves. The reason for this deadlock situation is that the component REG_1 behaves incorrectly with respect to gate synchronisation with REG_2 on the $full$ signal. REG_1 engages in the $full$ signal in a consequence to reception of the set event, whereas the REG_2 component can engage in $full$ signal after it receives out from REG_1 by its set input event. Both the components are deadlock free if they are taken standalone, also the network of both components coordinated by the bus HSK is deadlock free. However, encapsulating such an interoperably correct network to the synchronous gate, the interoperability correctness is violated. Later on in this section we develop a general solution for checking such kind of correctness violation.

Finally, the rule (5) treats the case when the gate mapping is of universal type. The situation here is similar to the situation of the rule (3). The only difference is, that the nondeterministic choice of the component interaction with respect to the gate mapping comprises maximal number of components which can synchronise in a particular network configuration. No interoperability failure can arise here, because the universal gate mapping does not strictly force any synchronisation.

8.4.2 Network Interoperability

In the scope of network, mutual interoperability of components is controlled by behavioural model of buses which realise component interconnection. The only rules responsible for derivation of component cooperation behaviour are (9) and (10). There are several cases when cooperation of interoperably correct components can lead to interoperability failure of

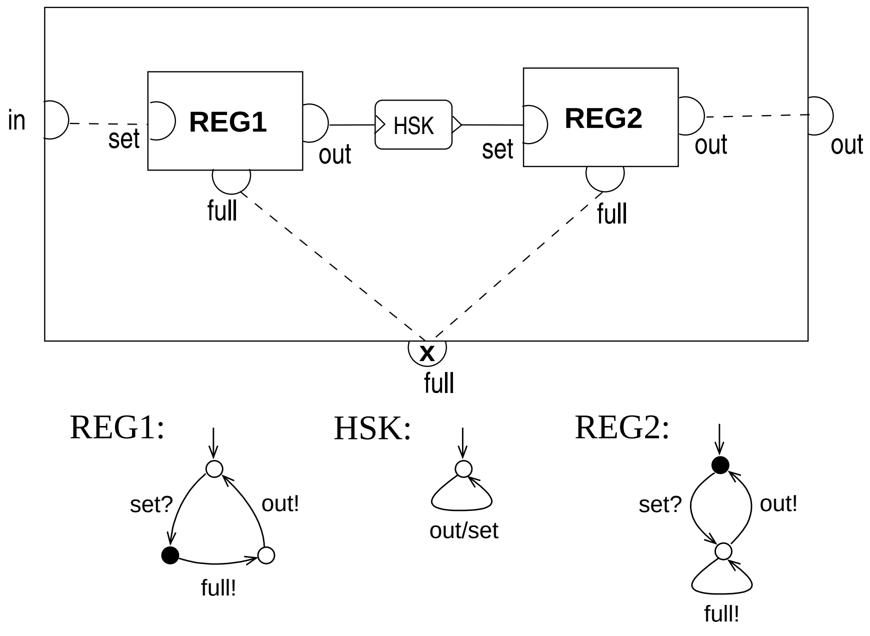


Figure 8.3: Violation of gate interoperability correctness by deadlock

the entire network topology composed of such components. These cases are characterised by the shape of the particular network topology.

To capture the network topology shape, we introduce the notion of network elements dependency graph (abbreviated dependency graph). It is defined as a bipartite graph in which the nodes are just buses and components of a particular network. Each edge in such a graph represents the fact that there is at least one link between a specific pair of a bus and a component. The information about multiple links between a component and a bus is abstracted and so is the information about link direction. Dependency graph of some network can contain cycles. Cycles are the main source of possible network interoperability failure, as it can be seen in Figure 8.4.

8.4.3 Formal Solution

Our formal solution for checking of interoperability correctness of VCN asynchronous behavioural model revisits and extends the approach previously presented and proved by Bernardo et.al. in [AB05]. Similarly to that work, we establish the solution of checking for arbitrary interoperability-critical property which is preserved by specific behavioural equivalence of VCN structural terms. This equivalence is just the behavioural equivalence of structural terms stratified w.r.t. the particular interoperability property.

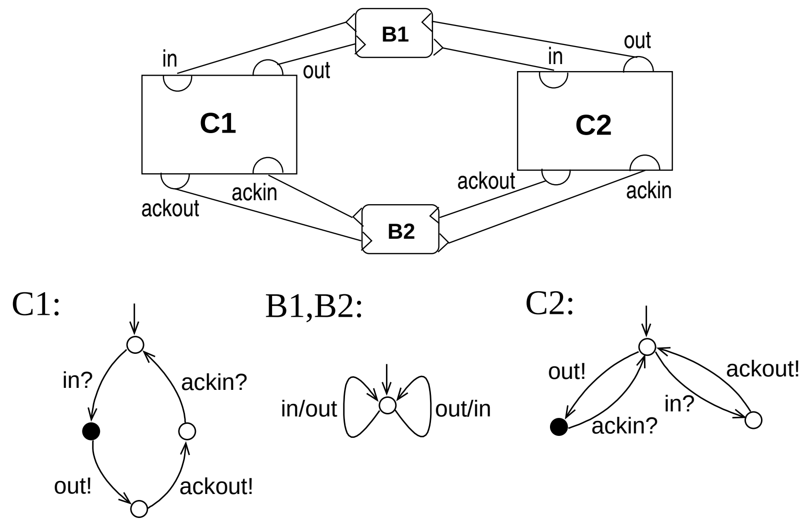


Figure 8.4: Violation of network interoperability correctness by deadlock

With respect to the inductive definition of structural terms the particular interoperability check traverses the hierarchy from leaves to the top-most network.

Note that the approach of Bernardo et.al. deals with design architectures composed of uniform components connected by links which can be of one-to-many character. Anyway, those links are static (stateless) connectors. There is no explicit notion of a connector like in Wright or in our approach. However, connectors can be there still modelled explicitly as components which are logically treated differently than common components. What is not possible there is modelling of connector dynamism concerning atomic many-to-many cooperations, as our approach allows by the behavioural model of buses. This is the reason why the results of [AB05] cannot be directly applied for developing the interoperability checking framework for the behavioural model of VCN (for technical details see Note 8.15). Therefore we follow the way of utilising and extending those results to fit the character of behavioural model of buses.

Notation 8.8 *For a component C_i and a bus B_j of some network $\langle\vec{C},\vec{B},L,Lranks\rangle \in \mathbf{T}_{\mathrm{st}}$ denote $Llinks(C_i,B_j)$ the set of all links between C_i and B_j:*

$$Llinks(C_i,B_j) \overset{\mathrm{df}}{=} \{l \in links(L,B_j) \,\|\, \exists p \in ports(I(C_i)).\, port(l) = p^{\sharp_i}\}$$

Definition 8.9 *Let $N = \langle\langle C_1,...,C_n\rangle,\langle B_1,...,B_m\rangle,L,Lrank\rangle$ a network. Define the* dependency graph of *N, denoted $\mathcal{G}(N)$, as the bipartite graph $\mathcal{G}(N) \overset{\mathrm{df}}{=}$*

$\langle \{C_1, ..., C_n\} \cup \{B_1, ..., B_m\}, E\rangle$ where E is defined in the following way:

$$E \overset{\text{df}}{=} \{\langle C_i, B_j\rangle \parallel Llinks(C_i, B_j) \neq \emptyset\}$$

Denote $nd_C(\mathcal{G}(N))$ the set of all component nodes of $\mathcal{G}(N)$, $nd_C(\mathcal{G}(N)) \overset{\text{df}}{=} \{C_1, ..., C_n\}$, and $nd_B(\mathcal{G}(N))$ the set of all its bus nodes, $nd_B(\mathcal{G}(N)) \overset{\text{df}}{=} \{B_1, ..., B_m\}$.

In the following part, we will assume φ a formula of modal μ-calculus expressing some interoperability safety property. An example of such a property can be the deadlock freedom expressed by the formula:

$$\nu Z.\langle\langle -\rangle\rangle \texttt{tt} \wedge [[-]]Z$$

Notation 8.10 *Denote $\approx_\varphi$ weak bisimulation stratified by the property φ. Such an equivalence relation is coarser than the original $\approx$ in that it additionally distinguishes states which satisfy φ and states which do not. The relevant behavioural equivalence of structural terms is denoted $\cong_\varphi$.*

Definition 8.11 *Let $C \in \mathbf{CT}_{\text{st}}$ a component, $T \in \mathbf{T}_{\text{st}}$ a network or a leaf, and $B \in \text{Buses}$ a bus. Further let φ be a property expressed in AFMC with weak box and weak diamond operators.*

We say C satisfies *φ if $\Phi_C(C) \models \varphi$, T* satisfies *φ if $\Phi^a(T) \models \varphi$, and B* satisfies *φ if $\Phi_B(B) \models \varphi$.*

We begin with the very basic kind of networks. In particular, we focus on two cases of network topologies in which the dependency graph is connected. The purely acyclic case is treated at first, the solution for a purely cyclic topology is discussed consequently. Finally the solution is extended to arbitrary topology.

In [BCD02] it has been proved that for checking of an acyclic component topology it suffices to check interaction compatibility of all pairs of mutually connected components. The notion of such compatibility is based on a weak bisimilarity of the two components in a pair. Abstraction of both components is taken comprising only the actions of mutual interaction, while all the other actions are hidden. To overcome the potential problem with internal nondeterminism of components, the compatibility checking is realised in such a way that one of the components is checked on weak bisimilarity against the behaviour of its parallel composition with the other component.

We show that the similar approach can be extended to the behavioural model of VCN. As the dependency graph of a VCN network is bipartite and the notion of network composition is more intricated than the parallel composition operator in pure process algebraic approaches, we have at first to establish the notion of compatibility between its arbitrary two adjacent nodes, which are always a bus and a component.

In order to establish the interoperability checking methodology, we need to analyse networks in the sense of inferring the validity of some property φ for the entire network from validity of φ computed for its non-trivial sub-parts, so-called *subnetworks*. Subnetwork is intuitively a network smaller than the mother network and consists of a subset of components, subset of buses, and subset of the link relation of the mother network. Subnetwork can also contain a bus which is connected to some component outside of the subnetwork. In this case all cooperations of such a bus are restricted only to events of components inside the subnetwork. For a link relation L of a particular subnetwork, such a restricted bus is called *L-projection* of the original bus. Formal definitions of the notion of bus projection and the notion of subnetwork are the following.

Definition 8.12 *Let $N = \langle C_1, ..., C_n \rangle, \langle B_1, ..., B_m \rangle, L, Lrank \rangle$ a network, and let B_j for some $j \in \{1, ..., m\}$ its bus, cm$(B_j) = \langle Q, T, q_0 \rangle$ its cooperation machine, and $L' \subseteq L$ a link relation. Define L'-*projection of B_j*, denoted $\pi(B_j, L')$, as the bus $B' \in \mathrm{Buses}$ with semantics defined by the cooperation machine cm$(B') \overset{\mathrm{df}}{=} \langle Q, T', q_0 \rangle$ where T' is defined in the following way:*

$$\langle q, \langle W'/R' \rangle, q' \rangle \in T' \overset{\mathrm{df}}{\Leftrightarrow} \langle q, \langle W/R \rangle, q' \rangle \in T$$

where

- $W' \overset{\mathrm{df}}{=} W \cap ports(B_j, L')$
- $R' \overset{\mathrm{df}}{=} R \cap ports(B_j, L')$

The set of cooperations coop(B') is defined as the set of all cooperations appearing in labels of T'.

Definition 8.13 *Let $N = \langle \langle C_1, ..., C_n \rangle, \langle B_1, ..., B_m \rangle, L, Lrank \rangle \in \mathbf{T}_{\mathrm{st}}$ a network. For some $n' \leq n$ and $m' \leq m$ define the* subnetwork N' of N *as an arbitrary network $N' = \langle \langle C'_1, ..., C'_{n'} \rangle, \langle B'_1, ..., B'_{m'} \rangle, L', Lrank' \rangle \in \mathbf{T}_{\mathrm{st}}$ satisfying:*

- $\{C'_1, ..., C'_{n'}\} \subseteq \{C_1, ..., C_n\}$
- $\{B'_1, ..., B'_{m'}\} \subseteq \{B_1, ..., B_m\}$
- $L' \overset{\mathrm{df}}{=} \{l \in L \| l \in Llinks(C, B) \wedge C \in \{C'_1, ..., C'_{n'}\} \wedge B \in \{B'_1, ..., B'_{m'}\}\}$
- *Each $B' \in \{B'_1, ..., B'_{m'}\}$ is defined as $\pi(B_j, L')$ for some $j \in \{1, ..., m\}$.*
- $Lrank' \overset{\mathrm{df}}{=} Lrank_{\|_{L'}}$ *is restriction of $Lrank$ to L'*

Further define for some $i \in \{1, ..., n\}$ star topology of B_j in N *as a subnetwork N' of N having the following form:*

$$N' \overset{\text{df}}{=} \langle\langle C'_1, ..., C'_{n'}\rangle, \langle B'\rangle, L', Lrank'\rangle \in \mathbf{T}_{\text{st}}$$

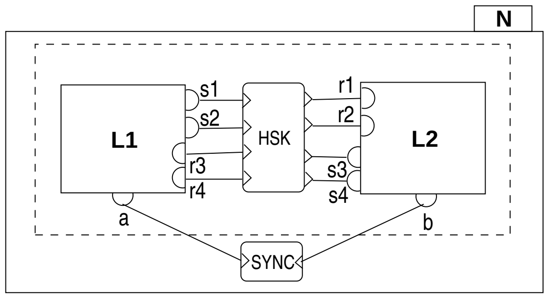
where

- *Each $C'_i \in \{C'_1, ..., C'_{n'}\}$ satisfies $Llinks(C'_i, B_j) \neq \emptyset$*
- *$L' \overset{\text{df}}{=} \{l \in L \parallel \exists i \in \{1, ..., n'\}. \, l \in Llinks(C'_i, B_j)\}$*
- *$Lrank' \overset{\text{df}}{=} Lrank_{\|_{L'}}$ is restriction of $Lrank$ to L'*
- *$B' \overset{\text{df}}{=} \pi(B_j, L')$*

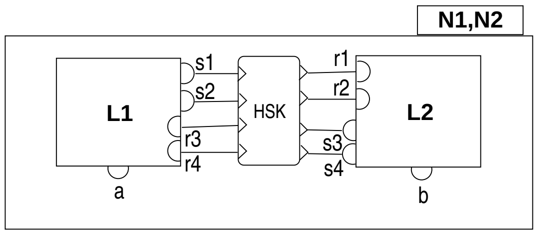
Note 8.14 *To avoid the problems which could be caused by changing the component annotation indeces when taking a subnetwork as a standalone network annotated by $\{1, ..., n'\}$ for some n', we rather assume the annotation of subnetwork components to be the same as in the original network. Hence if we have for example a subnetwork N' containing components C_2, C_4 of the original network $N = \langle\langle C_1, C_2, C_3, C_4\rangle, \vec{B}, L, Lrank\rangle$, we do not change the annotation of events in $bbox(C_2)$ and $bbox(C_4)$ when considering the subnetwork N' as a standalone network. Thus the set of annotation indeces of $N' = \langle\langle C_2, C_4\rangle, \vec{B}', L', Lrank'\rangle$ is $\{2, 4\} \subset \{1, ..., 4\}$.*

Note 8.15 *Because of the complexity of internetwork relationships (i.e., cyclic dependency of components), behavioural equivalence of two networks might not be in general a congruence w.r.t. subnetwork interchange as it is demonstrated in Figure 8.5 and Figure 8.6. In those figures, there are two networks N_1 and N_2 satisfying $N_1 \cong N_2$ taken as a subnetwork of a particular network N. If we denote M_1 the network N containing N_1 and M_1 the network N containing N_2 then we get $M_1 \ncong M_2$. The fact that behavioural equivalence is not a congruence w.r.t. subnetwork interchange leads us to development of specific techniques for interoperability checking of VCN networks. In other words, this is the main reason why the techniques of Bernardo et.al. cannot be employed in VCN.*

To analyse interoperability of buses and particular components in a network we need to look into the network internal behaviour. More specifically, cooperations cannot be hidden in such analysis. In particular, we need to observe cooperations occurring on the network link relation or on its specific subset. To capture this kind of observation of network behaviour with respect to some link relation L, we define L-observable model of the network. This operational model is based on the cooperation-labelled transition relation.

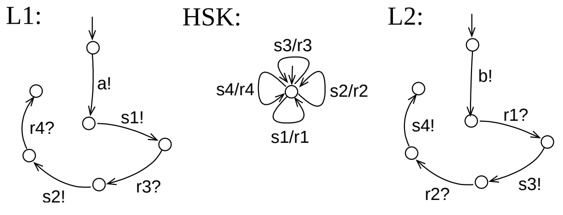


(a) a sub-topology of a network N (subnet denoted as a dashed box)

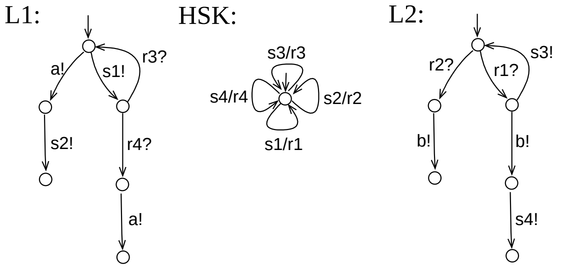


(b) shared topology of some two subnetworks of N — $N1$ and $N2$,

$N_1 \equiv N_2 \equiv \langle\langle L_1, L_2\rangle, \langle HSK\rangle, \{L_1.s_1! \mapsto HSK, L_1.s_2! \mapsto HSK, L_1.r_3? \mapsto HSK, L_1.r_4? \mapsto HSK,$
$L_2.r_1? \mapsto HSK, L_2.r_2? \mapsto HSK, L_2.s_3! \mapsto HSK, L_2.s_4! \mapsto HSK\}, \emptyset\rangle$

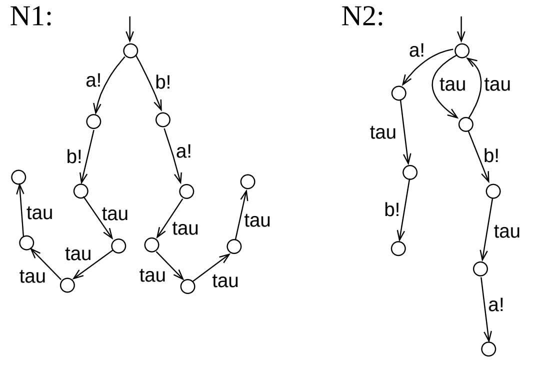


(c) semantics of N_1's bus and leaves

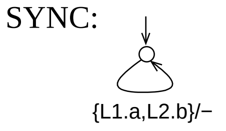


(d) semantics of N_2's bus and leaves

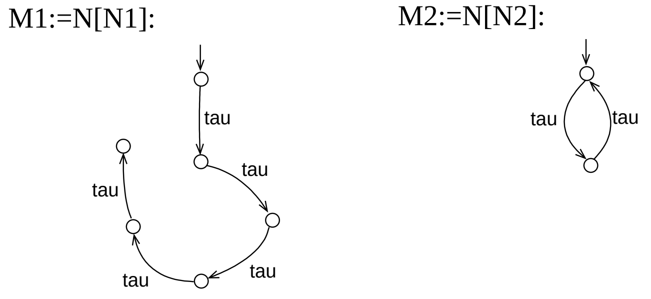
Figure 8.5: Example of two different subnetworks of a network N



(a) semantics of subnetworks N_1 and N_2 (considered as separate networks)



(c) semantics of the bus $SYNC$ included in the main network N



(c) inequivalent semantics of N with N_1 and N with N_2

Figure 8.6: Violation of congruence property w.r.t subnetwork interchange

Definition 8.16 *Let $N = \langle\langle C_1, ..., C_n\rangle, \langle B_1, ..., B_m\rangle, L, Lrank\rangle \in \mathbf{T}_{\mathrm{st}}$ a network and let $L' \subseteq L$ link relation. Define L'-*observation of network N *by a transition relation $\rightarrow_{L'}$ defined over network configurations by the rules $(7', 8', 10')$ from the proof of lemma 8.7 and the following rule:*

$$(9')\quad \frac{q_j \overset{W/R}{\rightarrow}_{B_j} q'_j \quad \forall e^{\sharp_i} \in \mathcal{M}_{B_j}.\, c_i \overset{e}{\rightarrow}_C c'_i}{\mathtt{N} \overset{W'/R'}{\rightarrow}_{L'} \mathtt{N}[\bigwedge_{i\in\theta} c_i := c'_i, q_j := q'_j]} \quad \left[\begin{array}{c} \mathtt{N} = \langle\langle c_1, .., c_n\rangle, \langle q_1, .., q_j, .., q_m\rangle\rangle \\ maxenabled(\langle W/R\rangle, \mathtt{N}, q_j) \\ \mathcal{M}_{B_j} \overset{\mathrm{df}}{=} \mathcal{E}(W) \cup \mathcal{E}(R) \\ \theta \overset{\mathrm{df}}{=} \{i \parallel e^{\sharp_i} \in \mathcal{M}_{B_j}\} \\ W' \overset{\mathrm{df}}{=} W \cap ports(L') \\ R' \overset{\mathrm{df}}{=} R \cap ports(L') \end{array}\right]$$

*The L'-*observation of network *N, denoted $\Phi^a_{L'}(N)$, is determined as in the case of the common behavioural model by the initial network configuration $\Phi^a(N)$, but the transition relation $\rightarrow_{L'}$ is taken instead of $\rightarrow_N$.*

Definition 8.17 *Let $N = \langle\langle C_1, ..., C_n\rangle, \langle B_1, ..., B_m\rangle, L, Lrank\rangle$ a network, and let $C_i = \langle T, I, G\rangle$ and B_j (for $i \in \{1, ..., n\}, j \in \{1, ..., m\}$) adjacent nodes of $\mathcal{G}(N)$. Further let $L' \subseteq L$ defined as a link relation $L' \overset{\text{df}}{=} \{l \in links(L, B_j) \parallel \exists p \in I(C_i). p^{\sharp_i} = port(l)\}$.*

We say C_i is compatible in *N* with *B_j* w.r.t. property *φ, written $C_i \bowtie^N_\varphi B_j$, if and only if the following equation holds:*

$$\Phi_B(B') \approx^{cl}_\varphi \Phi^a_{L'}(\langle\langle T, I', G'\rangle, \langle B'\rangle, L', Lrank'\rangle)$$

where

- $I' \overset{\text{df}}{=} I_{\parallel_P}$ *where* $P \overset{\text{df}}{=} \{p \in ports(I) \parallel \exists l \in L'. p^{\sharp_i} = port(l)\}$
- $G' \overset{\text{df}}{=} G_{\parallel_{I'}}$
- $B' \overset{\text{df}}{=} \pi(B_j, L')$
- $Lrank' \overset{\text{df}}{=} Lrank_{\parallel_{L'}}$

Lemma 8.18 *Let $N = \langle\langle C_1, ..., C_n\rangle, \langle B_1, ..., B_m\rangle, L, Lrank\rangle \in \mathbf{T}_{\text{st}}$ a network with $\mathcal{G}(N)$ acyclic, where each $C_i = \langle T_i, I_i, G_i\rangle$ for some $T_i \in \mathbf{T}_{\text{st}}$. Let B_j a bus for some $j \in \{1, ..., m\}$. For each star topology $\langle\langle C'_1, ..., C'_{n'}\rangle, \langle B'\rangle, L', Lrank'\rangle$ of B_j in N, if $C'_i \bowtie^N_\varphi B_j$ for all $i \in \{1, ..., n'\}$ then*

$$\Phi_B(B') \approx^{cl}_\varphi \Phi^a_{L'}(\langle\langle C''_1, ..., C''_{n'}\rangle, \langle B'\rangle, L', Lrank'\rangle) \tag{8.1}$$

where

- $B' \overset{\text{df}}{=} \pi(B_j, L')$
- *For each* $i \in \{1, ..., n'\}$ $C''_i \overset{\text{df}}{=} \langle T_i, I'_i, G'_i\rangle$
  - $I'_i \overset{\text{df}}{=} I_{i\parallel_P}$ *where* $P \overset{\text{df}}{=} \{p \in ports(I) \parallel \exists l \in L'. p^{\sharp_i} = port(l)\}$
  - $G'_i \overset{\text{df}}{=} G_{i\parallel_{I'_i}}$

Proof: Throughout the proof we use the following notation:

- $\rightarrow_{\pi_{[n]}}$ denotes the transition of the bus projection $\pi(B, L')$ where L' is the link relation L restricted to contain only all the links of each $C_i \in \{C_1, ..., C_n\}$ with B_j;
- $\rightarrow_{\pi_n}$ denotes the transition of the bus projection $\pi(B, L')$ where L' contains only the links between C_n and B_j;

- $\doteq$ denotes the technical fact that γ cannot be directly treated as a set of events, but the events must be formally divided into the output and input group to match the syntax of cooperations.

We follow the proof by induction on n'. Not to unnecessarily over-complicate the notation, n' is abbreviated n in the proof. In this way, all the primed symbols from the lemma are taken unprimed here, as only the star topology subnetwork is relevant for proving the claim of the lemma and therefore the symbols cannot be mixed with those in N.

- If $n = 1$ then the situation is degraded for some $i \in \{1, ..., n\}$ to $C_i \bowtie_{\varphi}^{N} B_j$ which is trivially satisfied by the assumption of compatibility of C_i and B_j.

- If $n > 1$ assume that the equation 8.1 holds for $n - 1$.

  We prove that the following relation is (weak) bisimulation.

$$Rel_n \overset{\mathrm{df}}{=} \{(q, \langle\langle \mathtt{c_1}, ..., \mathtt{c_n}\rangle, \langle q\rangle\rangle) \,\|\, \forall i \in \{1, ..., n\}.\, q \approx_{\varphi}^{cl} \langle\langle \mathtt{c_i}\rangle, \langle q\rangle\rangle\}$$

  If $\langle\langle \mathtt{c_1}, ..., \mathtt{c_{n-1}}, \mathtt{c_n}\rangle, \langle q\rangle\rangle \xrightarrow{\gamma} \mathtt{N}'$ then γ must be of one of the following forms:

  - $\gamma \doteq \{e^{\sharp_i} \,\|\, i \in \mathcal{I}\}$ where $\mathcal{I} \subseteq \{1, ..., n-1\}$ some nonempty index set

    As we take take the open behavioural model of the star topology, the only possible engaging in such a transition under γ implies that $\mathtt{c_i} \xrightarrow{e_i}_C \mathtt{c_i'}$ for all $e^{\sharp_i} \in \gamma$, where ${e_i}^{\sharp_i} = e^{\sharp_i}$ (e_i is unannotated $e^{\sharp_i}$). Note that $\mathtt{c_n}$ does nothing, so that $\mathtt{N}'[c_n := \mathtt{c_n}]$. Moreover, $q \xrightarrow{\gamma}_{\pi_{[n]}} q'$. From that follows $\langle\langle \mathtt{c_1}, ..., \mathtt{c_{n-1}}\rangle, \langle q\rangle\rangle \xrightarrow{\gamma} \mathtt{N}''$. As $(q, \langle\langle \mathtt{c_1}, ..., \mathtt{c_{n-1}}\rangle, \langle q\rangle\rangle) \in Rel_{n-1}$, by the induction hypothesis $q \overset{\gamma}{\Rightarrow}_{\pi_{[n-1]}} q'$ and $q' \approx_{\varphi}^{cl} \mathtt{N}''$. Compatibility of C_n and B_j gives $q' \approx_{\varphi}^{cl} q \approx_{\varphi}^{cl} \langle\langle \mathtt{c_n}\rangle, \langle q\rangle\rangle$. Hence, $(q', \mathtt{N}') \in Rel_n$.

  - $\gamma \doteq \{e^{\sharp_i} \,\|\, i \in \mathcal{I}\} \cup \{e^{\sharp_n}\}$ where $\mathcal{I} \subseteq \{1, ..., n-1\}$ some nonempty index set

    Here, similarly as in the previous case, performing of such transition means that $\mathtt{c_i} \xrightarrow{e_i}_C \mathtt{c_i'}$ must be fired for all $e^{\sharp_i} \in \gamma$, where ${e_i}^{\sharp_i} = e^{\sharp_i}$. Note that also $\mathtt{c_n} \xrightarrow{e_n}_C \mathtt{c_n'}$. This is only possible if the bus engages in the relevant transition, hence $q \xrightarrow{\gamma}_{\pi_{[n]}} q'$ too. From that follows $\langle\langle \mathtt{c_n}\rangle, \langle q\rangle\rangle \xrightarrow{e^{\sharp_n}} \langle\langle \mathtt{c_n'}\rangle, \langle q'\rangle\rangle$ and $\langle\langle \mathtt{c_1}, ..., \mathtt{c_{n-1}}\rangle, \langle q\rangle\rangle \xrightarrow{\gamma'} \mathtt{N}''$ where $\gamma' \doteq \gamma \setminus \{e^{\sharp_n}\}$. This action restriction occurs by hiding of the component C_n and removing all links between C_n and B_j.

As $C_n \bowtie_{\varphi}^{N} B_j$, we have $q \overset{e^{\sharp n}}{\Rightarrow}_{\pi_n} q'$ and $q' \approx_{\varphi}^{cl} \langle\langle \mathtt{c}'_{\mathtt{n}} \rangle, \langle q' \rangle\rangle$. Moreover $(q, \langle\langle \mathtt{c}_1, ..., \mathtt{c}_{\mathtt{n}-1} \rangle, \langle q \rangle\rangle) \in Rel_{n-1}$ and hence by the induction hypothesis $q \overset{\gamma'}{\Rightarrow}_{\pi_{[n-1]}} q'$ and $q' \approx_{\varphi}^{cl} \mathtt{N}''$. Thus $(q', \mathtt{N}') \in Rel_n$.

– $\gamma \equiv \langle \emptyset / \emptyset \rangle$
  In this case either $\mathtt{c_i} \overset{\tau}{\rightarrow}_C \mathtt{c}'_{\mathtt{i}}$ for some $i \in \{1, ..., n\}$ or $q \overset{\emptyset/\emptyset}{\rightarrow}_{\pi_{[n]}} q'$. The requirement of weak transition in B is satisfied trivially in both cases.

If $q \overset{\gamma}{\rightarrow}_{\pi_{[n]}} q'$ then we have to distinguish cases where component configuration $\mathtt{c_n}$ is involved in γ or not. Therefore we follow in the similar way like in the previous case.

– $\gamma \doteq \{e^{\sharp i} \,\|\, i \in \mathcal{I}\}$ where $\mathcal{I} \subseteq \{1, ..., n-1\}$ some nonempty index set
  We have by induction hypothesis $\langle\langle \mathtt{c}_1, ..., \mathtt{c_n} \rangle, \langle q \rangle\rangle \overset{\gamma}{\Rightarrow} \mathtt{N}'$ and $q' \approx_{\varphi}^{cl} \mathtt{N}'$. By the compatibility of C_n and B_j follows $\langle\langle \mathtt{c_n} \rangle, \langle q \rangle\rangle \approx_{\varphi}^{cl} q \approx_{\varphi}^{cl} q'$. Therefore $\langle\langle \mathtt{c}_1, ..., \mathtt{c}_{\mathtt{n}-1}, \mathtt{c_n} \rangle, \langle q \rangle\rangle \overset{\gamma}{\Rightarrow} \mathtt{N}''$ and $(q', \mathtt{N}'') \in Rel_n$.

– $\gamma \doteq \{e^{\sharp i} \,\|\, i \in \mathcal{I}\} \cup \{e^{\sharp n}\}$ where $\mathcal{I} \subseteq \{1, ..., n-1\}$ some nonempty index set
  Here by compatibility of C_n and B_j we have $\langle\langle \mathtt{c_n} \rangle, \langle q \rangle\rangle \overset{e^{\sharp n}}{\Rightarrow} \langle\langle \mathtt{c}'_{\mathtt{n}} \rangle, \langle q' \rangle\rangle$ and $q' \approx_{\varphi}^{cl} \langle\langle \mathtt{c}'_{\mathtt{n}} \rangle, \langle q' \rangle\rangle$. Let $\gamma' \doteq \gamma \setminus \{e^{\sharp n}\}$. By the induction hypothesis $q \overset{\gamma'}{\rightarrow}_{\pi_{[n-1]}} q'$ implies $\langle\langle \mathtt{c}_1, ..., \mathtt{c}_{\mathtt{n}-1} \rangle, \langle q, \rangle\rangle \overset{\gamma'}{\Rightarrow} \mathtt{N}'$ and $q' \approx_{\varphi}^{B} \mathtt{N}'$. As γ is a maximal enabled cooperation in that situation, the premise of the rule (5) is satisfied and hence $\langle\langle \mathtt{c}_1, ..., \mathtt{n-1}, \mathtt{n} \rangle, \langle q \rangle\rangle \overset{\gamma}{\Rightarrow} \mathtt{N}''$. Recall that $q' \approx_{\varphi}^{cl} \langle\langle \mathtt{c}'_{\mathtt{n}} \rangle, \langle q' \rangle\rangle$. Thus $(q, \mathtt{N}'') \in Rel_n$.

– $\gamma \equiv \langle \emptyset / \emptyset \rangle$
  In this case the claim holds trivially.

□

The key property which allows the previous lemma to hold is hiding of all component actions with the only exception of those connected to the bus making the centre of the star topology. Such an abstraction cannot be used when dealing with a cyclic network topology. An example of the interoperability correctness violation in the cyclic topology is depicted in Figure 8.4. Note that if we take only the star topology $\langle\langle C'_1, C'_2 \rangle, \langle \pi(B_1, L) \rangle, L \equiv \{\langle in^{\sharp 1}, B_1 \rangle, \langle out^{\sharp 1}, B_1 \rangle, \langle in^{\sharp 2}, B_1 \rangle, \langle out^{\sharp 2}, B_1 \rangle\}, \emptyset\rangle$ where the components C'_1, C'_2 are C_1, C_2 projected to links in L then such a subnetwork is deadlock free. The similar situation holds for the star topology of the bus B_2. However, taking the whole cyclic topology deadlock can arise, as is depicted by the darkened states.

Theorem 8.19 *Let $N = \langle\langle C_1, ..., C_n\rangle, \langle B_1, ..., B_m\rangle, L, Lrank\rangle \in \mathbf{T}_{\mathrm{st}}$ a network with $\mathcal{G}(N)$ cyclic, where each $C_i = \langle T_i, I_i, G_i\rangle$ is connected by some sequence of edges with each component of $\{C_1, .., C_{i-1}, C_{i+1}, C_n\}$ and each bus of $\{B_1, ..., B_m\}$. If there exists $i \in \{1, ..., n\}$ such that*

$$\Phi^a\langle\langle C_i'\rangle, \langle\rangle, \emptyset, \emptyset\rangle \approx^{cl}_{\varphi} \Phi^a_{L'}(\langle\langle C_1', ..., C_n'\rangle, \langle B_1, ..., B_m\rangle, L, Lrank\rangle)$$

where

- *$L' \overset{\mathrm{df}}{=} \bigcup_{j\in\{1,...,m\}} Llinks(C_i, B_j)$*
- *For each $i \in \{1, ..., n\}$ $C_i' \overset{\mathrm{df}}{=} \langle T_i, I_i', G_i'\rangle$*
  - *$I_i' \overset{\mathrm{df}}{=} I_{i\|_P}$ where $P \overset{\mathrm{df}}{=} \{p \in ports(I_i) \parallel \exists l \in L. p^{\sharp_i} = port(l)\}$*
  - *$G_i' \overset{\mathrm{df}}{=} G_{i\|_{I_i'}}$*

and for all $i \in \{1, ...n\}$ and $j \in \{1, ..., m\}$ C_i, B_j satisfy φ then N satisfies φ. Additionally, converse of this implication also holds.

Proof: The relation $\approx^{cl}_{\varphi}$ preserves the property φ. Hence, the claim holds.

□

Note 8.20 *It is worth noting that a stronger version of the previous theorem where the existential quantification is changed to the universal quantification also holds. The reason is that if for some component of the cycle the theorem was not satisfied then we would achieve a contradiction with the assumption that N satisfies φ. However, the existential condition stated in the theorem 8.19 is necessary and also sufficient for the cycle interoperability result.*

Finally, we extend our solution to analysis of arbitrary network topology.

Definition 8.21 *Let $N = \langle\vec{C}, \vec{B}, L, Lrank\rangle$ a network. Define* cycle covering strategy *σ for N by the following algorithm:*

1. *Initially let all components and buses of N unmarked.*
2. *If there is an unmarked bus B in some cycle of $\mathcal{G}(N)$, mark it and mark all components and buses which form a cycle with B in $\mathcal{G}(N)$.*

Definition 8.22 *Let* $N \overset{\text{df}}{=} \langle \vec{C}, \langle B_1, .., B_j, .., B_m \rangle, L, Lrank \rangle$ *a network. Define* cyclic N-neighbourhood of B_j*, denoted* $c\kappa(B_j, N)$*, as the set*

$$\{\Omega \parallel \Omega \text{ a maximal cyclic subgraph of } \sigma\text{-covering of } \mathcal{G}(\mathrm{N}) \text{ containing } \mathrm{B_j}\}$$

The set of all maximal cyclic subgraphs of $\mathcal{G}(N)$ *is denoted* $c\kappa(N)$ *and defined*

$$c\kappa(N) \overset{\text{df}}{=} \{c\kappa(B, N) \parallel B \in \{B_1, ..., B_m\}\}.$$

Maximality of the cycles detected by the σ-covering in the definition of $c\kappa(N)$ is important. It ensures that if we replace each cycle of $c\kappa(N)$ with an acyclic subnetwork which has behavioural model isomorphic with behaviour of the original cycle, the dependency graph of the resulting network is acyclic. In the following part, we focus on relations which can exist among such maximal cycles in the network. Moreover, we define the replacements which reduce the cyclic dependency graph to the acyclic graph corresponding to a network which is behaviourally isomorphic with the original network containing cycles. By that way, we establish an algorithm for checking architectural interoperability of networks having arbitrary topologies.

Lemma 8.23 *Let* N *a network. For any two cyclic subgraphs* $\Omega_1, \Omega_2 \in c\kappa(N)$ *such that* $\Omega_1 \neq \Omega_2$*. Just one of the following situations is the way how* Ω_1 *and* Ω_2 *are directly connected.*

1. *There is a bus* B *which belongs to both* Ω_1 *and* Ω_2*.*
2. *There is a component* C *which belongs to both* Ω_1 *and* Ω_2*.*
3. *There is a bus* B *in* Ω_1 *and a component* C *in* Ω_2 *and there exists some link between* B *and* C*.*
4. *There is a component* C *in* Ω_1 *and a bus* B *in* Ω_2 *and there exists some link between* C *and* B*.*

Definition 8.24 *Let* $N = \langle \langle C_1, ..., C_n \rangle, \langle B_1, ..., B_m \rangle, L, Lrank \rangle$ *a network and* $\Omega \in c\kappa(N)$ *some (maximal) cycle of* $\mathcal{G}(N)$*. Define* border *of* Ω*, denoted* $\beta(\Omega)$*, as a union of the set of components in* Ω *each of which has at least one port not connected to any bus in* Ω*, and the set of buses each of which has at least one cooperation containing a port not belonging to any component in* Ω*:*

$$\begin{aligned}
&\beta(\Omega) \overset{\text{df}}{=} \\
&\quad \{C \in nd_C(\Omega) \parallel \exists p \in ports(I(C)), \forall B \in nd_B(\Omega).\, p \notin ports(Llinks(B, C))\} \cup \\
&\quad \{B \in nd_B(\Omega) \parallel \exists \gamma \in coop(B), p^\sharp \in \gamma, \forall C \in nd_C(\Omega).\, p \notin ports(Llinks(B, C))\}
\end{aligned}$$

Definition 8.25 *Let* $N = \langle \langle C_1, ..., C_n \rangle, \langle B_1, ..., B_m \rangle, L, Lrank \rangle$ *a network and* $\Omega \in c\kappa(N)$ *some (maximal) cycle of* $\mathcal{G}(N)$*. Define* Ω*-subnetwork as a subnetwork* $\langle \langle C'_1, ..., C'_{n'} \rangle, \langle B'_1, ..., B'_{m'} \rangle, L', Lrank' \rangle \in \mathbf{T}_{\text{st}}$ *of* N *satisfying:*

- $\{C'_1, ..., C'_{n'}\} = nd_C(\Omega)$
- $\{B'_1, ..., B'_{m'}\} = \{\pi(B, L') \parallel B \in nd_B(\Omega)\}$

Note that the link relation L' and the corresponding ranking $Lrank'$ in the previous definition is uniquely determined by the vectors of components and buses in terms of definition 8.13.

In the following theorem, there is declared a set of conditions which are both necessary and sufficient to ensure an arbitrary network topology to satisfy some property φ.

Theorem 8.26 *Let $N = \langle\langle C_1, ..., C_n\rangle, \langle B_1, ..., B_m\rangle, L, Lrank\rangle$ a network with $\mathcal{G}(N)$ which is a connected graph of arbitrary shape, and let φ a property. N satisfies φ if and only if the following conditions hold:*

1. *Each bus $B_j \in \{B_1, ..., B_m\}$ satisfies φ.*
2. *Each component $C_i \in \{C_1, ..., C_n\}$ satisfies φ.*
3. *Each bus B which is not included in any cycle of $c\kappa(N)$ satisfies $C \bowtie_\varphi B$ for each C such that $Llinks(C, B) \neq \emptyset$.*
4. *Each bus $B \in \beta(\Omega)$ of an arbitrary cycle $\Omega \in c\kappa(N)$ satisfies $C \bowtie_\varphi B$ for each C such that $Llinks(C, B) \neq \emptyset$ and C is not included in Ω.*
5. *For each cycle $\Omega \in c\kappa(N)$ it holds that the Ω-subnetwork of N satisfies φ.*

To constructively prove this crucial theorem, we apply the idea of step-by-step reduction of the entire potentially cyclic topology to a smaller acyclic topology which has the behavioural model observationally equivalent to the original topology. This reduction is realised with respect to the relations among cycles in the topology declared in lemma 8.23. The intuition about such an observational behaviour preserving reduction of cyclic topologies to acyclic topologies is the following. Each cycle in the entire network is replaced with a star topology which has the behaviour model equivalent to the cyclic topology. This star topology is defined in such a way that all the relevant links leading from components and buses of the cycle to the buses and components outside the cycle are remapped to equivalent links leading from components and buses of the star topology. The important property that must be satisfied by such a replacement is mutual compatibility of those components and buses. Moreover, the star topology must itself satisfy compatibility of its components with the bus forming its centre. Then by replacing all the cyclic subnetworks with such compatible star topologies the acyclicity and intercompatibility of the entire network is achieved. The interoperability result is then established by extending the lemma 8.18 to a general acyclic topology.

In order to substitute a cyclic subnetwork with the respective acyclic subnetwork, at first we need to capture observational behaviour of the original cyclic subnetwork. As some buses of the cyclic subnetwork may be linked to an acyclic part of the entire network or to another cyclic subnetwork in terms of lemma 8.23, we have to include the relevant (open) parts of cooperations of such buses into the behavioural model of the desired acyclic subnetwork. We call such a kind of observation an open behavioural model.

Secondly we need to define the desired acyclic subnetwork in such a way that it is equivalent to the original cyclic subnetwork in terms of the open behavioural model and hence we achieve mutual interchangeability of the two subnetworks with respect to bisimulation of their open behavioural models. We define the desired acyclic subnetwork as a star topology consisting of just the components which are included in the border of the cycle of the original subnetwork. The centre of this star topology is defined by additionally introducing a bus B' which represents the open behavioural model of the relevant subnetwork of the original cycle. We call such a subnetwork an internal subnetwork of the cycle. The internal subnetwork is constructed from the original cyclic topology by removing all the bordering components. Hence its open behavioural model contains only the behaviour observed in the relevant parts of the cycle which is necessary for establishing the required bisimulation relationship between the original cyclic topology and the respective star topology. Transitions included in the internal subnetwork directly reflect cooperations of buses from the cycle border. More specifically, each cooperation in which some component of the cycle border or some component outside the entire cycle is involved, is directly reflected in the open behavioural model of the internal subnetwork. Therefore the bus B' directly reflects all the internal behaviour of the original cyclic topology which is observed at the level of the cycle border.

In contrary, the internal cycle behaviour which is not observed at the level of the cycle border is hidden in B' (i.e., represented by internal transitions). This comprises the following cooperations:

- Each cooperation in which only internal components of the cycle are involved is hidden (represented as an internal $\langle\emptyset/\emptyset\rangle$-transition of B').
- If a cooperation contains events of some components included in the cycle border and also some components which are not included there then we consider in B' the projection of such a cooperation that is restricted to events of bordering components.
- Transitions of internal components (components not included in the cycle border) are hidden (represented as internal $\langle\emptyset/\emptyset\rangle$-transitions of B').

Note that behaviour of bordering components is not reflected in B' because these components are contained themselves in the resulting star topology.

In the following paragraphs we introduce formally the notion of open network, the open behavioural model, and the L-observation of the open behavioural model. In consequence, we formulate the lemma capturing the cycle reduction. Next we show that the construction in this lemma satisfies the required bisimulation relationship. Finally we prove the theorem 8.26.

Definition 8.27 *Define the* open network N *as a network* $N = \langle\langle C_1, ..., C_n\rangle, \langle B_1, ..., B_m\rangle, L, Lrank\rangle \in \mathbf{T}_{\mathrm{st}}$ *in which there exists at least one port* $p^\sharp$ *included in a cooperation of* $coop(B_j)$ *for some bus* B_j *such that there is no link for this port in* L, $p^\sharp \notin ports(links(L, B_j))$.

Define behavioural model of open network N, *denoted* $\Phi^a_o(N)$, *by a transition relation* $\to_o$ *defined over network configurations by the rules* $(7', 8', 10')$ *from the proof of lemma 8.7 additionally extended with the following rule:*

$$(9^*)\quad \frac{q_j \overset{W/R}{\rightarrow}_{B_j} q'_j \quad \forall e^{\sharp_i} \in \mathcal{M}_{B_j}.\, c_i \overset{e}{\rightarrow}_C c'_i}{\mathtt{N} \overset{W'/R'}{\rightarrow}_o \mathtt{N}[\bigwedge_{i\in\theta} c_i := c'_i, q_j := q'_j]} \quad \left[\begin{array}{c} \mathtt{N} = \langle\langle c_1, .., c_n\rangle, \langle q_1, .., q_j, .., q_m\rangle\rangle \\ W' \overset{\mathrm{df}}{=} \{w^\sharp \in W \| w^\sharp \notin ports(links(L, B_j))\} \\ R' \overset{\mathrm{df}}{=} \{r^\sharp \in R \| r^\sharp \notin ports(links(L, B_j))\} \\ maxenabled(\langle W''/R''\rangle, \mathtt{N}, q_j) \\ \mathcal{M}_{B_j} \overset{\mathrm{df}}{=} \mathcal{E}(W'') \cup \mathcal{E}(R'') \\ \text{where } W'' \overset{\mathrm{df}}{=} W \setminus W' \text{ and } R'' \overset{\mathrm{df}}{=} R \setminus R' \\ \theta \overset{\mathrm{df}}{=} \{i \parallel e^{\sharp_i} \in \mathcal{M}_{B_j}\} \end{array}\right]$$

Let $L' \subseteq L$ *a link relation. Define* L'-observation of open network N, *denoted* $\Phi^a_{oL'}(N)$, *by a transition relation* $\to_{oL'}$ *given by the rules* $(7', 8')$ *and the following rule:*

$$(9^+)\quad \frac{q_j \overset{W/R}{\rightarrow}_{B_j} q'_j \quad \forall e^{\sharp_i} \in \mathcal{M}_{B_j}.\, c_i \overset{e}{\rightarrow}_C c'_i}{\mathtt{N} \overset{W'/R'}{\rightarrow}_{oL'} \mathtt{N}[\bigwedge_{i\in\theta} c_i := c'_i, q_j := q'_j]} \quad \left[\begin{array}{c} \mathtt{N} = \langle\langle c_1, .., c_n\rangle, \langle q_1, .., q_j, .., q_m\rangle\rangle \\ W' \overset{\mathrm{df}}{=} W_1 \cup W_2 \text{ where} \\ W_1 \overset{\mathrm{df}}{=} \{w^\sharp \in W \| w^\sharp \notin ports(links(L, B_j))\} \\ W_2 \overset{\mathrm{df}}{=} W \cap ports(L') \\ R' \overset{\mathrm{df}}{=} R_1 \cup R_2 \text{ where} \\ R_1 \overset{\mathrm{df}}{=} \{r^\sharp \in R \| r^\sharp \notin ports(links(L, B_j))\} \\ R_2 \overset{\mathrm{df}}{=} R \cap ports(L') \\ maxenabled(\langle W''/R''\rangle, \mathtt{N}, q_j) \\ \mathcal{M}_{B_j} \overset{\mathrm{df}}{=} \mathcal{E}(W'') \cup \mathcal{E}(R'') \\ \text{where } W'' \overset{\mathrm{df}}{=} W \setminus W_1 \text{ and } R'' \overset{\mathrm{df}}{=} R \setminus R_1 \\ \theta \overset{\mathrm{df}}{=} \{i \parallel e^{\sharp_i} \in \mathcal{M}_{B_j}\} \end{array}\right]$$

In the prove of the following lemma we use the L'-observation of the original cyclic network to show star compatibility of the resulting acyclic network.

Definition 8.28 *Let* $N = \langle\langle C_1, ..., C_n\rangle, \langle B_1, ..., B_m\rangle, L, Lrank\rangle \in \mathbf{T}_{\mathrm{st}}$ *a network. For some* $n_k \leq n$ *and* $m' \leq m$ *define the* open subnetwork N' of N *as a network* $N' = \langle\langle C_{n_1}, ..., C_{n_k}\rangle, \langle B_{m_1}, ..., B_{m_k}\rangle, L', Lrank'\rangle \in \mathbf{T}_{\mathrm{st}}$ *satisfying:*

- $\{C_{n_1}, ..., C_{n_k}\} \subseteq \{C_1, ..., C_n\}$
- $\{B_{m_1}, ..., B_{m_k}\} \subseteq \{B_1, ..., B_m\}$
- $L' \overset{\text{df}}{=} \{l \in L \parallel l \in Llinks(C, B) \wedge C \in \{C_{n_1}, ..., C_{n_k}\} \wedge B \in \{B_{m_1}, ..., B_{m_k}\}\}$
- $Lrank' \overset{\text{df}}{=} Lrank_{\parallel_{L'}}$ *is restriction of* $Lrank$ *to* L'

Note 8.29 *Similarly as in the case of a common subnetwork, we assume the annotation of components in the open subnetwork* N' *to be equal to the annotation of the relevant component in the original network* N. *Hence the set of annotation indeces of components in* N' *satisfies* $\{n_1, ..., n_k\} \subseteq \{1, ..., n\}$.

We follow with definition of the internal subnetwork. The main motivation for such a notion is capturing of the internal behaviour of a network having the cyclic shape. As we have explained above, the internal network is the cornerstone of transforming cyclic topologies to behaviourally equivalent acyclic topologies.

Definition 8.30 *Let* $N = \langle\langle C_1, ..., C_n\rangle, \langle B_1, ..., B_m\rangle, L, Lrank\rangle$ *a network with* $\mathcal{G}(N)$ *purely cyclic. Define* internal subnetwork *of* N, *denoted* $N_{\neg\beta}$, *as an open subnetwork* $N_{\neg\beta} \overset{\text{df}}{=} \langle\langle C'_1, ..., C'_{n'}\rangle, \langle B_1, ..., B_m\rangle, L', Lrank'\rangle$ *of* N *where*

- $\{C'_1, ..., C'_{n'}\} = \beta(\mathcal{G}) \cap nd_C(\mathcal{G})$
- $L' \overset{\text{df}}{=} L \setminus \bigcup_{B_j \in nd_B(\mathcal{G})} \bigcup_{C_i \notin \beta(\mathcal{G})} Llinks(C_i, B_j)$
- $Lrank \overset{\text{df}}{=} Lrank_{\parallel L'}$

Note 8.31 *Note that for a particular network* N *satisfying the assumption of the previous definition the internal subnetwork is defined uniquely. The reason for that is that the internal subnetwork must contain all the bordering components and all the buses of the respective cyclic network.*

The following lemma is used in the proof of the succeeding lemma and characterises the fact that if we take a cyclic network and remove a bordering component then such a modification preserves the open behavioural model of the respective internal network.

Lemma 8.32 *Let* $N = \langle\langle C_1, ..., C_n\rangle, \langle B_1, ..., B_m\rangle, L, Lrank\rangle$ *a network with* $\mathcal{G}(N)$ *purely cyclic. For each component* C_i *for some* $i \in \{1, ..., n\}$ *such that* $C_i \in \beta(\mathcal{G})$ *the following claim holds:*

$$\Phi_o^a(N_{\neg\beta}) \approx_\varphi^{cl} \Phi_o^a(N'_{\neg\beta})$$

where $N' \overset{\text{df}}{=} \langle\langle C_1, ..., C_{i-1}, C_{i+1}, ..., C_n\rangle, \langle B_1, ..., B_m\rangle, L', Lrank'\rangle$ *such that*

- $L' \overset{\text{df}}{=} L \setminus \bigcup_{j\geq 1}^{m} Llinks(C_i, B_j)$
- $Lrank' \overset{\text{df}}{=} Lrank_{||_{L'}}$

Proof: By definition of internal subnetwork, the component C_i is not included in both $N_{\neg\beta}$ and $N'_{\neg\beta}$. Hence both subnetworks have isomorphic open behavioural models. □

At next the crucial lemma the intuition of which has been given above is presented. This lemma realises the reduction of a cyclic subnetwork to an acyclic subnetwork preserving the behavioural model.

Lemma 8.33 *Let N a network of an arbitrary topology containing at least one cycle $\Omega \in c\kappa(N)$. Let N_Ω the open Ω-subnetwork of N having the form $N_\Omega = \langle\langle C_1, ..., C_n\rangle, \langle B_1, ..., B_m\rangle, L, Lrank\rangle$. Suppose M is an open network $M \overset{\text{df}}{=} \langle\vec{C}, \vec{B}, L', \emptyset\rangle$ where*

- *If $\beta(\Omega)$ contains no component then $\vec{C} = \emptyset$, otherwise $\vec{C} = \langle C_{n_1}, ..., C_{n_k}\rangle$ where $\{C_{n_1}, ..., C_{n_k}\} = \beta(\Omega) \cap nd_C(\Omega)$.*
- *$\vec{B} = \langle B'\rangle$ with behaviour defined by $cm(B') \overset{\text{df}}{=} \Phi_o^a(N_{\neg\beta})$ where $N_{\neg\beta}$ is the internal subnetwork of N_Ω.*
- *$L' \overset{\text{df}}{=} \{\langle p^\sharp, B'\rangle \mid\mid \langle p^\sharp, B\rangle \in Llinks(C, B), B \in nd_B(\Omega), C \in \beta(\Omega) \cap nd_C(\Omega)\}$*

Then the following properties hold:

1. *$\Phi_o^a(M) \sim^{cl} \Phi_o^a(N_\Omega)$*
2. *If N_Ω satisfies φ then $C_{n_i} \bowtie_\varphi^M B'$ for each $n_i \in \{n_1, ..., n_k\}$*

Proof: At first we prove (1). We begin with characterisation of the relation among configurations of $\Phi_o^a(N_{\neg\beta})$ and states of the cooperation machine $cm(B')$. Note that as $cm(B')$ is defined just as the open behavioural model of the internal subnetwork of N_Ω, the both cooperation machines are isomorphic. This is crucial argument which allows us to prove the required bisimulation stated in the claim (1).

Another important fact which enables the claim (1) to be satisfied concerns the relationship between the link relation of N_Ω and the link relation of M. Especially, each free port of every bordering component in N_Ω is also a free port of the respective component in M. Moreover, there is no other free port introduced in M. Additionally, each component which has a free port is included in the cycle border. Hence the following equation holds:

$$freeports(N_\Omega) = freeports(M)$$

Focusing on links between each bordering component and every bus in N_Ω, we have that those links are bijectively remapped to relevant links between each respective component of M and the bus B'. Links leading from internal components in N_Ω are the only links lost by the construction of M. This is an expected fact.

We prove that the following relation is (strong) bisimulation:

$$Rel \overset{\mathrm{df}}{=} \{(\underbrace{\langle\langle \mathtt{c}_{\mathtt{n}_1}, .., \mathtt{c}_{\mathtt{n}_k}\rangle, \langle \mathtt{q}\rangle\rangle}_{\mathtt{M}}, \underbrace{\langle\langle \mathtt{c}_1, .., \mathtt{c}_\mathtt{n}\rangle, \langle \mathtt{q}_1, .., \mathtt{q}_\mathtt{m}\rangle\rangle}_{\mathtt{N}}) \parallel \mathtt{N}_{\neg\beta} \equiv \mathtt{q}\}$$

where $\mathtt{N}_{\neg\beta} \equiv \langle\langle \mathtt{c}_{\mathtt{n}'_1}, .., \mathtt{c}_{\mathtt{n}'_{k'}}\rangle, \langle \mathtt{q}_1, .., \mathtt{q}_\mathtt{m}\rangle\rangle$ for $\{n'_1, ..., n'_{k'}\} \subseteq \{n_1, ..., n_k\}$ satisfying $\{C_{n'_1}, ..., C_{n'_{k'}}\} = \beta(\Omega) \cap nd_C(\Omega)$.

Suppose $\mathtt{M} \xrightarrow{\gamma} \mathtt{M}'$ for some configuration $\mathtt{M}'$. There are following possibilities of the form of γ:

- $\gamma \equiv \langle W/R\rangle$ where $W \neq \emptyset$ and $R \neq \emptyset$

In this situation the supposed transition must be inferred according to the rule (9*). With respect to the structure of the open network M, in particular the fact that there is only one bus B', it follows that this bus must be involved in the supposed transition. Hence there exist $W' \supseteq W$ and $R' \supseteq R$ such that $\mathtt{q} \xrightarrow{W'/R'}_{B'} \mathtt{q}'$ for some $\mathtt{q}, \mathtt{q}' \in Q(B')$. Note that the sets $W' \setminus W$ and $R' \setminus R$ are given by ports of the surrounding components involved in the cooperation implied by the supposed transition. Therefore for each $p^{\sharp n_i} \in (W' \setminus W) \cup (R' \setminus R)$ there must exist a transition in n_ith component such that $\mathtt{c}_{\mathtt{n}_\mathtt{i}} \overset{\mathcal{E}(p)}{\to} \mathtt{c}'_{\mathtt{n}_\mathtt{i}}$ for the respective component configuration included in $\mathtt{M}$. As M consists only of components which were originally included in N_Ω and the annotation of the components is preserved in M with respect to note 8.29, the configuration $\mathtt{c}_{\mathtt{n}_\mathtt{i}}$ is directly contained in $\mathtt{N}$, $\mathtt{c}_{\mathtt{n}_\mathtt{i}} = \mathtt{c}_\mathtt{j}$ where $j = n_i$.

In general, the cooperation $\langle W'/R'\rangle$ is union of two mutually disjoint cooperation parts $\langle W'/R'\rangle = \gamma_1 \cup \gamma_2$ such that $\gamma_1 \doteq \{p^{\sharp n_i} \in \langle W'/R'\rangle \parallel n_i \in \{n_1, ..., n_k\}\}$ and $\gamma_2 \overset{\mathrm{df}}{=} \langle W'/R'\rangle \setminus \gamma_1$. Note that $\gamma_1 \equiv \langle W' \setminus W/R' \setminus R\rangle$ and $\gamma_2 \equiv \langle W/R\rangle$. Moreover, the cooperation part γ_2 can be further divided into another two cooperation parts, $\gamma_2 = \delta_1 \cup \delta_2$ where $\delta_1 \overset{\mathrm{df}}{=} \{p^{\sharp n_i} \in \gamma_2 \parallel n_i \in \{1, ..., n\} \setminus \{n_1, ..., n_k\}\}$ and $\delta_2 \overset{\mathrm{df}}{=} \gamma_2 \setminus \delta_1$. Ports in the cooperation part δ_1 correspond w.r.t. construction of M to ports of those components in N_Ω which are not included in the border $\beta(\Omega)$. In contrary, ports included in δ_2 correspond to ports of components outside of the Ω-subnetwork which are connected to B'. Hence δ_2 is given just by openness of N_Ω.

Note that γ_1 might not be necessarily non-empty. The situation when $\gamma_1 = \langle\emptyset/\emptyset\rangle$ implies $W' = W$ and $R' = R$. Such a situation corresponds to a cooperation imposed by a bus B_j of N_Ω satisfying $B_j \in \beta(\Omega)$ in which no component included in $\beta(\Omega)$ is involved. However, this particular situation is included in the general arguments which follows below.

By isomorphism of the state q of B' and the configuration $\mathtt{N}_{\neg\beta}$, it follows:

$$\begin{tikzcd}
\mathtt{q} \arrow[r, "W'/R'"] \arrow[d, equal] & {}_{B'}\ \mathtt{q}' \arrow[d, equal] \\
\mathtt{N}_{\neg\beta} \arrow[r, "W'/R'"'] & {}_{o}\ \mathtt{N'}_{\neg\beta}
\end{tikzcd}$$

where $\mathtt{N'}_{\neg\beta}$ is a configuration of $N_{\neg\beta}$ and $\mathtt{q}'$ a state of B' such that $\mathtt{N'}_{\neg\beta} \equiv \mathtt{q}'$ w.r.t. isomorphism of the open behavioural model of $N_{\neg\beta}$ and the cooperation machine $cm(B')$. Moreover, the configuration $\mathtt{N'}_{\neg\beta}$ must have the following form:

$$\mathtt{N'}_{\neg\beta} \equiv \mathtt{N}_{\neg\beta}[\bigwedge_{i \in an(\delta_1)} c_i := c'_i, q_j := q'_j]$$

where $j \in \{1, ..., m\}$.

Hence we have a transition in the internal subnetwork of N_Ω. The only rule which can be employed for inference of the respective cooperation of components of $N_{\neg\beta}$ controlled by the bus B_j is the rule (9^*). By decomposition w.r.t. this rule we have a transition $\mathtt{c_i} \overset{\mathcal{E}(p)}{\rightarrow}_{C_i} \mathtt{c'_i}$ for each component C_i such that $i \in an(\delta_1)$ and $p^{\sharp i} \in \delta_1$. Moreover, the cooperation $\langle W'/R' \rangle$ is maximally enabled in the open behavioural model of $N_{\neg\beta}$.

Now we follow by arguing that just the transition discussed above can be extended to the requested transition of N_Ω. Note that as we have stated before, each configuration $\mathtt{c_{n_i}}$ for $n_i \in \delta_1$ is directly contained in $\mathtt{N}$. These are just the configurations which have been removed in construction of $cm(B')$. We know that the configuration $\mathtt{N}$ has the form:

$$\mathtt{N} \equiv \langle \langle \mathtt{c_1}, .., \mathtt{c_n} \rangle, \langle \mathtt{q_1}, .., \mathtt{q_m} \rangle \rangle$$

From what has been inferred above we know that $\mathtt{q_j}$ has an enabled transition under $\langle W'/R' \rangle$ and that each component configuration $\mathtt{c_i}$ for $i \in an(\gamma_1 \cup \delta_1)$ involved in the cooperation controlled by this transition of B_j is contained in $\mathtt{N}$. As also $\mathtt{q_j}$ is contained in $\mathtt{N}$, we only need to argue that the cooperation $\langle W'/R' \rangle$ is maximally enabled in $\mathtt{N}$. Then we achieve according to the rule (9^*) the required transition of N_Ω under $\langle W/R \rangle$.

Note that the configuration $\mathtt{N}$ is constructed from $\mathtt{N}_{\neg\beta}$ by adding of a configuration of each bordering component. Especially, the configuration $\mathtt{c_i}$ for each $i \in \gamma_1$ is included in such an extension. Hence we have all the component transitions required by $\gamma_1 \cup \gamma_2$ enabled in $\mathtt{N}$ and we know that $\gamma_1 \cup \gamma_2$ is maximally enabled i $\mathtt{N}_{\neg\beta}$. Because there are no other ports in γ_1 and $an(\gamma_2) \cap \{n_1, ..., n_k\} = \emptyset$ because of the definition of γ_2, it follows that

$\langle W'/R' \rangle$ is maximally enabled in $\mathtt{N}$. Note that each port in δ_2 is by definition unlinked in N_Ω w.r.t. openness of the bus B_j. Thus we have a transition:

$$\mathtt{N} \xrightarrow{W/R}_o \mathtt{N}' \equiv \mathtt{N}[\bigwedge_{i \in an(\gamma_1)} c_i := c_i', \bigwedge_{i \in an(\delta_1)} c_i := c_i', q_j := q_j']$$

With respect to the arguments above it additionally holds that $(\mathtt{M}', \mathtt{N}') \in Rel$.

- $\gamma \equiv \langle \{w^{\sharp_i}\}/\emptyset \rangle$ for some $i \in \{n_1, ..., n_k\}$

  Performing the supposed transition in this situation employs either the rule $(7')$ or (9^*). In the latter case, the supposed transition implies a state change of the bus B' and the proof follows the same steps as in the previous situation. In contrary, in the former case no cooperation occurs. Hence there must be a transition $\mathtt{c_{n_i}} \xrightarrow{\mathcal{E}(w)}_{C_i} \mathtt{c}'_{\mathtt{n_i}}$ of the component C_{n_i}. But as $C_{n_i} \in \beta(\Omega)$ and $freeports(C_{n_i}, L(M)) = freeports(C_{n_i}, L(N_\Omega))$ we have according to the rule $(7')$ a transition:

$$\mathtt{N} \equiv \langle \langle \mathtt{c}_1, .., \mathtt{c_n} \rangle, \langle \mathtt{q}_1, .., \mathtt{q_m} \rangle \rangle \xrightarrow{\{w^{\sharp_i}\}/\emptyset}_o \mathtt{N}' \equiv \mathtt{N}[c_{n_i} := c'_{n_i}]$$

  As $\mathtt{c_{n_i}}$ is a bordering component and thus not included in $\mathtt{N}_{\neg\beta}$, the respective subconfiguration $\mathtt{N}_{\beta\Omega}$ of the internal subnetwork is not changed in $\mathtt{N}'$. The bus B' does not change its state $\mathtt{q}$ because no cooperation is performed. Hence $(\mathtt{M}', \mathtt{N}') \in Rel$.

- $\gamma \equiv \langle \emptyset/\{r^{\sharp_i}\} \rangle$ for some $i \in \{n_1, ..., n_k\}$

  This situation is analogous to the previous case.

- $\gamma \equiv \langle \emptyset/\emptyset \rangle$

  If this situation is caused by a τ-transition of some component C_{n_i} then the proof is analogous to the previous case with the only difference that the rule $(8')$ is taken into account instead of the rule $(9')$.

  If the rule (9^*) is considered then by decomposition we achieve a transition $\mathtt{q} \xrightarrow{\gamma'}_{B'} \mathtt{q}'$ and there are two possibilities.

  1. $\gamma' \equiv \langle \emptyset/\emptyset \rangle$
     In this situation by the isomorphism of $\mathtt{q}$ and $\mathtt{N}_{\neg\beta}$ it follows:

$$\begin{tikzcd} \mathtt{q} \arrow[r, "\emptyset/\emptyset"] \arrow[d, equal] & {}_{B'}\ \mathtt{q}' \arrow[d, equal] \\ \mathtt{N}_{\neg\beta} \arrow[r, "\emptyset/\emptyset"] & {}_o\ \mathtt{N}'_{\neg\beta} \end{tikzcd}$$

     where $\mathtt{N}'_{\neg\beta}$ is a configuration of $N_{\neg\beta}$ and $\mathtt{q}'$ a state of B' such that $\mathtt{N}'_{\neg\beta} \equiv \mathtt{q}'$ Moreover, the configuration $\mathtt{N}'_{\neg\beta}$ must have either one of the following forms:

- $\mathtt{N'}_{\neg\beta} \equiv \mathtt{N}_{\neg\beta}[c_i := c'_i]$ where $i \in \{1, ..., n\} \setminus \{n_1, ..., n_k\}$
- $\mathtt{N'}_{\neg\beta} \equiv \mathtt{N}_{\neg\beta}[q_j := q'_j]$ where $j \in \{1, ..., m\}$

As $\mathtt{c_i}$ (the former case) or $\mathtt{q}$ (the latter case) is directly included in the configuration $\mathtt{N}$ we have the required transition $\mathtt{N} \overset{\emptyset/\emptyset}{\rightarrow} \mathtt{N'}$ and by the arguments above also $(\mathtt{M'}, \mathtt{N'}) \in Rel$.

2. $\gamma' \not\equiv \langle\emptyset/\emptyset\rangle$
   In this case the proof follows similar steps as in the case $\gamma \equiv \langle W/R\rangle$.

Now suppose $\mathtt{N} \equiv \langle\langle \mathtt{c_1}, .., \mathtt{c_n}\rangle, \langle \mathtt{q_1}, ..., \mathtt{q_m}\rangle\rangle \xrightarrow{\gamma}_o \mathtt{N'}$ for some configuration $\mathtt{N'}$. Similarly as in the opposite situation, there are following possibilities of the form of γ:

- $\gamma \equiv \langle W/R\rangle$ where $W \neq \emptyset$ and $R \neq \emptyset$

As both W and R are non-empty, the supposed transition must be derived in terms of the rule (9^*). Hence according to decomposition w.r.t. this rule there must be a bus B_j for some $j \in \{1, ..., m\}$ such that $B_j \in \beta(\Omega)$ which performs a transition $\mathtt{q_j} \xrightarrow{W'/R'}_{B_j} \mathtt{q'_j}$ where $W' \supset W$ and $R' \supset R$. Additionally, for each $p^{\sharp_i} \in (W' \setminus W) \cup (R' \setminus R)$ there must exist a transition in ith component such that $\mathtt{c_i} \overset{\mathcal{E}(p^{\sharp_i})}{\rightarrow} \mathtt{c'_i}$ for the respective component configuration included in $\mathtt{N}$. Note that the components involved in the cooperation can be of two kinds — components which are included in the border of Ω and components which are not. Components of the former kind are directly included in M and, as the annotation is preserved with respect to note 8.29, the configuration $\mathtt{c_i}$ is directly contained in $\mathtt{M}$ for each of these components. In contrary, components of the latter kind are not included in M. Each component of the latter kind has a behavioural model which contains either internal transitions or transitions which are involved in the considered cooperation. By definition of M, internal transitions of these components are not reflected in the open behavioural model of M.

In general, similarly as in the opposite case the cooperation $\langle W'/R'\rangle$ is union of two mutually disjoint cooperation parts $\langle W'/R'\rangle = \gamma_1 \cup \gamma_2$ such that $\gamma_1 \doteq \{p^{\sharp_i} \in \langle W'/R'\rangle \parallel i \in \{1, ..., n\}\}$ and $\gamma_2 \overset{\text{df}}{=} \langle W'/R'\rangle \setminus \gamma_1$. The cooperation part γ_1 can be further divided into another two cooperation parts, $\gamma_1 = \delta_1 \cup \delta_2$ where $\delta_1 \overset{\text{df}}{=} \{p^{\sharp_i} \in \gamma_1 \parallel C_i \in \beta(\Omega)\}$ and $\delta_2 \overset{\text{df}}{=} \gamma_1 \setminus \delta_1$. Ports in the cooperation part δ_1 correspond to ports of those components in N_Ω which are included in the border $\beta(\Omega)$. In contrary, ports of δ_2 correspond to components of N_Ω not included in the cycle border. Cooperation part γ_2 corresponds to ports of those components of the entire network N which are not included in N_Ω and which are connected to some bus of N_Ω. In other words, γ_2 is given by openness of N_Ω.

With respect to the decomposition of the supposed transition discussed above, it follows according to the transition (9^*) that the respective transition of $N_{\neg\beta}$ can be composed back. More particularly, there must be a cooperation $\langle W''/R''\rangle$ satisfying

$$\langle W/R\rangle \subseteq \langle W''/R''\rangle \subseteq \langle W'/R'\rangle$$

such that $\langle W''/R''\rangle$ is maximally enabled in $\mathtt{N}_{\neg\beta}$. Note that this cooperation corresponds just to the part δ_2 of the original cooperation. Hence the side-condition of the rule (9^*) is satisfied and we have the following transitions:

$$\begin{tikzcd}
\mathtt{N}_{\neg\beta} \arrow[r, "W'/R'"] \arrow[d, equal] & {}_o\ \mathtt{N'}_{\neg\beta} \arrow[d, equal] \\
\mathtt{q} \arrow[r, "W'/R'"] & {}_{B'}\ \mathtt{q}'
\end{tikzcd}$$

Note that if we consider a cooperation part γ' satisfying $\gamma' \subseteq \langle W'/R'\rangle$, $\gamma' \overset{\text{df}}{=} \langle W'/R'\rangle \setminus (\delta_1 \cup \gamma_2)$, then from the fact that component configurations of $\mathtt{M}$ are just the bordering component configurations of $\mathtt{N}$ involved in the supposed transition and that there is no other component configuration with any enabled transition under some port of $\langle W'/R'\rangle$ it follows γ' must be maximally enabled in $\mathtt{M}$. Hence we can finally conclude that with respect to the rule (9^*) and the arguments above it follows that

$$\mathtt{M} \overset{W/R}{\Rightarrow}_o \mathtt{M}' \equiv \mathtt{M}[\bigwedge_{n_i \in an(\delta_1)} c_{n_i} := c'_{n_i}, q := q']$$

and $(\mathtt{M}', \mathtt{N}') \in Rel$.

- $\gamma \equiv \langle \{w^{\sharp_i}\}/\emptyset\rangle$ for some $i \in \{n_1, ..., n_k\}$

  The result follows symmetrically to the respective opposite case.

- $\gamma \equiv \langle \emptyset/\{r^{\sharp_i}\}\rangle$ for some $i \in \{n_1, ..., n_k\}$

  This case is analogous to the previous situation.

- $\gamma \equiv \langle \emptyset/\emptyset\rangle$

  There are several possibilities of how the supposed transition is inferred in this case. If the rule $(8')$ is employed then we have the result by the same arguments as in the previous case with the only difference that the rule $(8')$ is considered instead of the rule $(7')$.

  If the rule (9^*) is employed then there exists a bus B_j, $j \in \{1, ..., m\}$ which causes a cooperation. There are two possibilities depending on the fact if B_j performs the $\langle \emptyset/\emptyset\rangle$-transition itself or it is a result of some non-trivial cooperation.

If there exists a state $\mathtt{q}'$ of B_j and a transition $\mathtt{q_j} \overset{\emptyset/\emptyset}{\rightarrow}_{B_j} \mathtt{q_j'}$ then we achieve with respect to definition of $\mathtt{N}_{\neg\beta}$ and isomorphism of $N_{\neg\beta}$ and B' the following transition:

$$\begin{tikzcd}
\mathtt{N}_{\neg\beta} \arrow[r, "\emptyset/\emptyset"] \arrow[d, equal] & {}_{o}\ \mathtt{N'}_{\neg\beta} \arrow[d, equal] \\
\mathtt{q} \arrow[r, "\emptyset/\emptyset"] & {}_{B'}\ \mathtt{q}'
\end{tikzcd}$$

Hence we have the transition $\mathtt{M} \overset{\emptyset/\emptyset}{\rightarrow}_o \mathtt{M}' \equiv \mathtt{M}[q := q']$ and $(\mathtt{M}', \mathtt{N}') \in Rel$.

If the supposed transition is inferred on the base of a nontrivial cooperation γ' such that $\mathtt{q_j} \overset{\gamma'}{\rightarrow}_{B_j} \mathtt{q_j'}$ then by the rule (9*) each component C_i, $i \in \{1, ..., n\}$ involved in such a cooperation performs a transition $\mathtt{c_i} \overset{e}{\rightarrow}_{C_i} \mathtt{c_i'}$ where $\mathcal{P}(e^{\sharp_i}) \in \gamma'$. The result follows by similar reasons as in the first case of the bisimulation direction just being proved.

Finally we prove (2). Let C_{n_i} be a component of M for some $n_i \in \{n_1, ..., n_k\}$. By definition 8.17 we need to prove $\Phi^a_{L_K}(K) \approx^{cl} \Phi_B(\pi(B', L_K))$ where $K \overset{\text{df}}{=} \langle\langle C_K\rangle, \langle\pi(B', L_K)\rangle, L_K, \emptyset\rangle$ satisfying

- $L_K \overset{\text{df}}{=} L_M links(C_{n_i}, B')$
- C_K is defined as C_{n_i} with the interface and gate restricted to $ports(L_K)$

As C_{n_i} is included in N_Ω and all relevant links are with respect to the definition of $\beta(\Omega)$ and $L(M)$ remapped to the respective links between C_{n_i} and B', if we denote

$$L'' \equiv \bigcup_{j \geq 1}^{m} L_{N_\Omega} links(C_{n_i}, B_j)$$

the link relation containing all the links of the component C_{n_i} in N_Ω then we have the following equation satisfied:

$$ports(L'') = ports(L_M links(C_K, B')) = ports(L_K)$$

Let $N^c \overset{\text{df}}{=} \langle\langle C'_{n_1}, ..., C'_{n_k}\rangle, \langle B'_1, ..., B'_m\rangle, L_{N_\Omega}, Lrank\rangle$ a network where for each $i \in \{1, ..., k\}$ the component C'_{n_i} is defined as C_{n_i} with the interface and gate restricted to $ports(L(N_\Omega))$ and for each $j \in \{1, ..., m\}$ the bus B'_j is defined as $L(N_\Omega)$-projection of B_j, $B'_j \overset{\text{df}}{=} \pi(B_j, L(N_\Omega))$. By the assumption N_Ω satisfies φ, and according to cycle interoperability stated in theorem 8.19 and note 8.20 we have the following equation:

$$\Phi^a(\langle\langle C_K\rangle, \langle\rangle, \emptyset, \emptyset\rangle) \approx^{cl}_{\varphi} \Phi^a_{L''}(N^c)$$

As all the buses in N^c are closed, we can rewrite the previous equality in the following form:

$$\Phi^a(\langle\langle C_K\rangle, \langle\rangle, \emptyset, \emptyset\rangle) \approx^{cl}_{\varphi} \Phi^a_{oL''}(N^c)$$

Note that if we remove the component $C'_{n_i} \equiv C_K$ from N^c and consider the respective subnetwork as an open network then the openness of such a network is given just by the ports in $ports(L'')$. Hence the cycle intercompatibility can be further transcribe in terms of the following equality:

$$\Phi^a(\langle\langle C_K\rangle, \langle\rangle, \emptyset, \emptyset\rangle) \approx^{cl}_{\varphi} \Phi^a_o(N^*)$$

where $N^* \stackrel{\text{df}}{=} \langle\langle C'_{n_1}, .., C'_{n_{i-1}}, C'_{n_{i+1}}, .., C'_{n_k}\rangle, \langle B'_1, ..., B'_m\rangle, L_{N_\Omega}, Lrank\rangle$.

Next from lemma 8.32 the following equation follows:

$$\Phi^a_o(N^*_{\neg\beta}) \approx^{cl}_{\varphi} \Phi^a_o(N^c_{\neg\beta})$$

The cycle interoperability additionally implies the equality

$$\Phi^a_o(N^*) \approx^{cl}_{\varphi} \pi(\Phi^a_o(N^*_{\neg\beta}), ports(L'')) \approx^{cl}_{\varphi} \pi(\Phi^a_o(N^c_{\neg\beta}), ports(L''))$$

and hence by isomorphism of $N_{\neg\beta}$ and B' semantics and the fact that L_K is given by bijective remapping of L'' we achieve

$$\pi(\Phi^a_o(N^c_{\neg\beta}), ports(L'')) \approx^{cl}_{\varphi} \Phi_B(\pi(B', L_K))$$

Denote $\mathcal{B}$ a bus satisfying $cm(\mathcal{B}) \stackrel{\text{df}}{=} \Phi^a_o(N^*)$ and $\mathcal{B}'$ a bus satisfying $cm(\mathcal{B}') \stackrel{\text{df}}{=} \pi(\Phi^a_o(N^c_{\neg\beta}), ports(L''))$. Then by transitivity of $\approx^{cl}_{\varphi}$ it holds that $cm(\mathcal{B}) \approx^{cl}_{\varphi} cm(\mathcal{B}')$. From lemma 8.5 follows

$$\Phi^a_{L_K}(\langle\langle C_K\rangle, \langle\mathcal{B}\rangle, L_K, \emptyset\rangle) \approx^{cl}_{\varphi} \Phi^a_{L_K}(\langle\langle C_K\rangle, \langle\mathcal{B}'\rangle, L_K, \emptyset\rangle) \approx^{cl}_{\varphi} \Phi^a_{L_K}(K)$$

By cycle interoperability we have

$$\Phi^a_{L''}(N^c) \approx^{cl}_{\varphi} \Phi^a_{L_K}(\langle\langle C_K\rangle, \langle\mathcal{B}\rangle, L_K, \emptyset\rangle$$

and thus

$$\Phi^a(\langle\langle C_K\rangle, \langle\rangle, \emptyset, \emptyset\rangle)) \approx^{cl}_{\varphi} \langle\langle C_K\rangle, \langle\mathcal{B}\rangle, L_K, \emptyset\rangle$$

Finally by transitivity of $\approx^{cl}_{\varphi}$ it follows that $\Phi^a_{L_K}(K) \approx^{cl}_{\varphi} \Phi_B(\pi(B', L_K))$ and hence $C_{n_i} \bowtie^M_{\varphi} B'$.

$\square$

Proof: (of Theorem 8.26) Assume that the conditions $(1-5)$ are satisfied by N. We show that also the entire network N satisfies φ. We proceed the proof by induction on the number $k \stackrel{\text{df}}{=} \|c\kappa(N)\|$ of cycles in $\mathcal{G}(N)$ and we use lemma 8.33 to realise the inductive step.

- $k = 0$

In this case the $\mathcal{G}(N)$ is acyclic. Hence only the conditions $(1 - 3)$ are relevant for such situation. If N contains only one component then the claim that N satisfies φ holds trivially. Assume that $N = \langle\langle C_1, ..., C_n\rangle, \langle B_1, ..., B_m\rangle, L, Lrank\rangle$ is such that both $n, m \geq 1$. The result of lemma 8.18 gives that for establishing the fact that a star topology N' centred around some bus B_j satisfies φ if all the components of N' are compatible with B_j. Therefore we focus on proving that if all star topologies of N satisfy φ then also entire network N satisfies φ. We follow the proof by induction on the number of star topologies in N. This number is equal to the number of buses m.

· $m = 0$

In this situation there is only one star topology, hence the condition (3) is satisfied by virtue of lemma 8.18.

· $m > 0$

Suppose that arbitrary subnetwork of N consisting of m star topologies satisfies φ. Let $S = \langle\langle C'_{n_M+1}, ..., C'_{n'}\rangle, \langle B_{m+1}\rangle, L_S, Lrank_S\rangle$ be a star topology of N which satisfies that nodes of $\mathcal{G}(S)$ are disjunct with nodes of $\mathcal{G}(M)$. We prove that the subnetwork M of N which is constructed by adding the star topology S to some subnetwork of N having m star topologies, $M \overset{\text{df}}{=} \langle\langle C'_1, ..., C'_{n_M}, C'_{n_M+1}, ..., C'_{n'}\rangle, \langle B_1, ..., B_m, B_{m+1}\rangle, L_M, Lrank_M\rangle$ where $L_S \subseteq L_M$, also satisfies φ.

We proceed the proof by transforming the network M to a network M' in which the subnetwork corresponding to the star topology S is replaced with a single component C_S that has the entire star topology S embedded as its component body. If we prove that such a transformation preserves the property φ then we reach the expected result by application of the induction hypothesis to M' consisting of only m star topologies.

Let the component $C_S \overset{\text{df}}{=} \langle S, I, G\rangle$ where $ports(I) \overset{\text{df}}{=} \{p_i \in \mathcal{P} \parallel p^{\sharp i} \in freeports(S)\}$ and G is defined in the following way:

- For each $p^{\sharp i} \in freeports(S)$ there is a mapping $g \in map_G$ satisfying $g(p^{\sharp i}) = p_i$.
- For each $g \in map_G$ the type of g is defined as $type_G(g) = \perp$.
- The gate mappings defined above are the only gate mappings in map_G.

Note that the gate function $gate_G$ of the gate defined above is bijection. To ensure that the validity of φ is preserved from S to C_S we have to discuss the rules (2) and (5) of the network operational semantics from Section 7.2.4. Note that because of the form of G these are the only rules which can be employed for inferring the transition system of C_S. All the observable actions of S are directly visible on the interface of C_S, $\|bbox(C_S)\| = \|wbox(C_S)\|$, and moreover $\Phi^a(S)$ is strongly bisimulation equivalent with $\Phi^a_C(C_S)$ up to the gate mapping function $gate_G$, denoted $\Phi^a_C(S) \sim_{gate_G} \Phi^a_C(C_S)$, because of the rules $(2,5)$. Thus it follows that if S satisfies φ then also C_S satisfies φ. We define the network M' in the following way:

$$M' \stackrel{\text{df}}{=} \langle\langle C'_1, ..., C'_{n_M}, C_S\rangle, \langle B_1, ..., B_m\rangle, L_{M'}, Lrank_{M'}\rangle$$

where $L_{M'} \stackrel{\text{df}}{=} L_M \setminus (L_S \cup \{l \parallel l \in L_M links(C'_i, B_j), i \in \{n_M+1, ..., n'\}, j \in \{1, ..., m\}\}) \cup \{\langle p_i{}^{\sharp\ell}, B_j\rangle \parallel \langle p^{\sharp_i}, B_j\rangle \in L_M, i \in \{n_M+1, ..., n'\}, j \in \{1, ..., m\}\}$, $\ell \stackrel{\text{df}}{=} n_M + 1$ denotes index of the component C_S in M'.

It remains to be proved that the following claim holds:

$$\Phi^a(S) \sim_{gate_G} \Phi^a_C(C_S) \Rightarrow \Phi^a(M) \sim_{gate_G} \Phi^a(M')$$

We prove that relation $Rel \stackrel{\text{df}}{=} \{(\langle\langle \mathtt{c}_1, .., \mathtt{c}_{n_M}, \mathtt{c}_{n_M+1}, .., \mathtt{c}_{n'}\rangle, \langle \mathtt{q}_1, .., \mathtt{q}_m, \mathtt{q}_{m+1}\rangle\rangle_M, \langle\langle \mathtt{c}_1, .., \mathtt{c}_{n_M}, \mathtt{c}_s\rangle, \langle \mathtt{q}_1, ..., \mathtt{q}_m\rangle\rangle_{M'}) \parallel \langle\langle \mathtt{c}_{n_M+1}, .., \mathtt{c}_{n'}\rangle, \langle \mathtt{q}_{m+1}\rangle\rangle \sim_{gate_G} \mathtt{c}_s\}$ is (strong) bisimulation.

Assume $\mathtt{M} \equiv \langle\langle \mathtt{c}_1, .., \mathtt{c}_{n_M}, \mathtt{c}_{n_M+1}, .., \mathtt{c}_{n'}\rangle, \langle \mathtt{q}_1, .., \mathtt{q}_m, \mathtt{q}_{m+1}\rangle\rangle \xrightarrow{e^\sharp}_M \mathtt{M}_1$ for some $e^\sharp \in obs(M)$ and $\mathtt{M}_1$ a configuration of M. With respect to the rule (7) of Section 7.2.4, such a transition can be performed only if $\mathtt{c}_i \xrightarrow{e}_{C_i} \mathtt{c}'$ for some $i \in \{1, ..., n'\}$ and $\mathtt{c}' \in Q(C_i)$. The configuration $\mathtt{M}_1$ has the form $\mathtt{M}_1 \equiv \mathtt{M}[c_i := \mathtt{c}']$. If $i \leq n_M$ then M' can perform the transition $\mathtt{M}' \equiv \langle\langle \mathtt{c}_1, .., \mathtt{c}_{n_M}, \mathtt{c}_s\rangle, \langle \mathtt{q}_1, ..., \mathtt{q}_m\rangle\rangle \xrightarrow{e^\sharp}_{M'} \mathtt{M}'_1 \equiv \mathtt{M}'[c_i := \mathtt{c}']$ and we have $(\mathtt{M}_1, \mathtt{M}'_1) \in Rel$.

If $n_M < i \leq n'$ then the star topology S can perform the transition $\mathtt{S} \xrightarrow{e^\sharp}_S \mathtt{S}' \equiv \mathtt{S}[c_i := \mathtt{c}']$ where the configuration $\mathtt{S}$ includes $\mathtt{c}_i$. As $\Phi^a(S) \sim_{gate_G} \Phi^a_C(C_S)$, there must be a transition $\mathtt{c}_s \xrightarrow{e_i}_{C_S} \mathtt{c}'$ of the component C_S such that $gate_G(e^{\sharp_i}) = e_i$ and $\mathtt{S}' \sim_{gate_G} \mathtt{c}'$. Hence we have a network transition $\mathtt{M}' \xrightarrow{e_i{}^{\sharp\ell}}_{M'} \mathtt{M}'_1 \equiv \mathtt{M}'[c_\ell := \mathtt{c}']$ where $\ell \stackrel{\text{df}}{=} n_M + 1$ and also $(\mathtt{M}_1, \mathtt{M}'_1) \in Rel$.

Now suppose $\mathtt{M} \xrightarrow{\tau}_M \mathtt{M}_1$. There are two possibilities of which rule can cause such a network transition. If the rule (8) is employed then the situation is similar to the previous case. As the τ action is preserved by any gate mapping function, the proof is even simpler. However, when the rule (9) is employed then the circumstances under which the transition occurs are different. Especially, there must be a transition in a bus B_j for some $j \in \{1, ..., m+1\}$ which causes synchronisation of some components of M. According to the rule (9) the group of components involved in synchronisation is characterised by an index set $\theta \subseteq \{1, ..., n'\}$.

If $j \leq m$ then the bus B_j is contained also in M'. Here it depends if there is some $i \in \theta$ such that $i > n_M$. If there is no such i all the components involved in synchronisation are contained also in the network M'. As the subnetwork of M containing those components is isomorphic to the respective subnetwork of M', the result is achieved trivially. However, when there is some component C_i such that $i \in \theta$ and $i > n_M$, we have to ensure that the relevant transition of C_i can be realised also by the component C_S in M' and that all links between B_j and C_i are contained also in the link relation $L_{M'}$ of the network M'. The former is achieved by the same arguments like in the case when the rule (7) has been treated. The latter is satisfied directly looking into the definition of $L_{M'}$. Note that no matter how many of the synchronised components are contained in the subnetwork of M which coincides with S, there can be always only one component such as that one discussed above. The reason for that is that $\mathcal{G}(M)$ is acyclic and if there was more than one component of index greater than n_M then the acyclicity of $\mathcal{G}(M)$ would be contradicted. Thus under these circumstances, for each transition $\mathtt{M} \xrightarrow{\tau}_M \mathtt{M}_1 \equiv \mathtt{M}[\bigwedge_{i\in\theta} c_i := c'_i, q_j := q'_j]$ there can be a transition $\mathtt{M}' \xrightarrow{\tau}_{M'} \mathtt{M}'_1 \equiv \mathtt{M}'[\bigwedge_{i\in\theta'} c_i := c'_i, q_j := q'_j]$ inferred where $\theta' \overset{\text{df}}{=} \{i \in \theta \parallel i \le n_M\} \cup \theta_0$ and

$$\theta_0 \overset{\text{df}}{=} \begin{cases} \emptyset & \text{if } \forall i \in \theta.\, i \le n_M \\ \{i\} & \text{if } \exists i \in \theta.\, i > n_M \end{cases}$$

By similar arguments as in the case in which the rule (7) has been discussed we also have $(\mathtt{M}_1, \mathtt{M}'_1) \in \mathit{Rel}$.

If $j = m + 1$ then each component which is involved in synchronisation must be a component of the star topology S. Hence the index set θ which characterises the group of synchronising components satisfies $\theta \subseteq \{n_M + 1, ..., n'\}$ in this case. The transition $\mathtt{M} \xrightarrow{\tau}_M \mathtt{M}_1 \equiv \mathtt{M}[\bigwedge_{i\in\theta} c_i := c'_i, q_j := q'_j]$ implies the following transition of S:

$$\mathtt{S} \equiv \langle\langle \mathtt{c}_{\mathtt{n_M}+1}, ..., \mathtt{c}_{\mathtt{n}'}\rangle, \langle \mathtt{q}\rangle\rangle \xrightarrow{\tau} \mathtt{S}' \equiv \mathtt{S}[\bigwedge_{i\in\theta} c_i := c'_i, q := q']$$

Therefore there is a transition of the component C_S, $\mathtt{c_s} \xrightarrow{\tau}_{C_S} \mathtt{c}'$ and thus $\mathtt{M}' \xrightarrow{\tau}_{M'} \mathtt{M}'_1 \equiv \mathtt{M}'[c_\ell := \mathtt{c}', q_m := \mathtt{q}']$ where $\ell \overset{\text{df}}{=} n_M + 1$. From the structure of the reached configurations and the fact that $\mathtt{S}' \sim_{gate_G} \mathtt{c}'$ it follows that $(\mathtt{M}_1, \mathtt{M}'_1) \in \mathit{Rel}$.

Now we prove the opposite direction of the bisimulation. Suppose $\mathtt{M}' \equiv \langle\langle \mathtt{c}_1, .., \mathtt{c}_{\mathtt{n_M}}, \mathtt{c_s}\rangle, \langle \mathtt{q}_1, ..., \mathtt{q_m}\rangle\rangle \xrightarrow{e^{\sharp i}}_{M'} \mathtt{M}'_1$ for some $e^{\sharp i} \in \mathit{obs}(M')$ and a configuration $\mathtt{M}'_1$ of M'. Such a transition can be inferred only in terms of the rule (7). Hence there is a component C_i for some $i \in \{1, ..., n_M + 1\}$ such that $\mathtt{c_i} \xrightarrow{e}_{C_i} \mathtt{c}'$. If $i \le n_M$ then the relevant transition of M is inferred trivially because $\mathtt{c_i}$ is also included in $\mathtt{M}$ and the link relation L_M leaves the port of $e^{\sharp i}$ free. If $i = n_M + 1$ then by equivalence of $\Phi^a(S)$ and $\Phi^a(C_S)$ we have a transition $\mathtt{S} \equiv \langle\langle \mathtt{c}_{\mathtt{n_M}+1}, ..., \mathtt{c}_{\mathtt{n}'}\rangle, \langle \mathtt{q}\rangle\rangle \xrightarrow{e'^{\sharp j}} \mathtt{S}' \equiv \mathtt{S}[c_j := \mathtt{c}']$ satisfying $gate_G(e'^{\sharp j}) = e$ for some $j \in \{n_M + 1, ..., n'\}$. Moreover, $\mathtt{S}' \sim_{gate_G} \mathtt{c}'$.

In M we can infer a transition $\mathtt{M} \xrightarrow{e'^{\sharp j}}_M \mathtt{M_1} \equiv \mathtt{M}[c_j := \mathtt{c}']$ and we also have $(\mathtt{M_1}, \mathtt{M'_1}) \in \mathit{Rel}$ by the arguments above.

Now suppose $\mathtt{M}' \xrightarrow{\tau}_{M'} \mathtt{M'_1}$. Similarly as in the opposite direction proved above, there are two possibilities of which rule can cause such a network transition. If the rule (8) is employed then the situation is similar to the previous case. Note that the τ action occurring in the component C_S can be also caused by synchronisation of some components in the star topology. Such a situation corresponds just to synchronisation of the relevant components in M by the bus B_{m+1} and hence the result is also achieved. When the rule (9) is employed then there must exist a transition in a bus B_j for some $j \in \{1, ..., m\}$ which causes synchronisation of some components of M'. According to the rule (8) the group of components involved in synchronisation is characterised by the index set $\theta \subseteq \{1, ..., n_M, n_M + 1\}$.

The group of components synchronised by the transition of the bus B_j can include the component C_S. If it does not contain this component, the result follows trivially. Similarly to the opposite case, to the subnetwork A of M' containing the relevant components and the bus B_j there exists a corresponding subnetwork B of M such that behaviour of A is isomorphic to the behaviour of B.

When $n_M + 1 \in \theta$ the supposed τ transition in M' represents synchronisation of a group of components including C_S. In more particular, we have $\mathtt{M}' \xrightarrow{\tau} \mathtt{M'_1} \equiv \mathtt{M}'[\bigwedge_{i \in \theta} c_i := c'_i, q_j := q']$. Such a transition according to the rule (8) implies the transition $\mathtt{q} \xrightarrow{W/R}_{B_j} \mathtt{q}'$ of the bus B_j. Assume that the transition of C_S involved in the synchronisation has the form $\mathtt{c}_\ell \xrightarrow{e_u}_{C_S} \mathtt{c}'_\ell$ for some $u \in \{n_M + 1, ..., n'\}$ where $\ell \overset{\text{df}}{=} n_M + 1$ and $e_u \in \mathcal{E}(W) \cup \mathcal{E}(R)$. We have to show that there is a component C'_u in the star topology S involved in the same synchronisation instead of C_S so that $\mathtt{S} \xrightarrow{e^{\sharp u}} \mathtt{S}' \equiv \mathtt{S}[c_u := \mathtt{c}']$ and $\mathtt{S}' \sim_{gate_G} \mathtt{c}'_\ell$. But from the equivalence of $\Phi^a(S)$ and $\Phi^a_S(C_S)$ up to the gate mapping function it follows that such a transition exists. According to definition of $L_{M'}$ and with respect to the fact that $\langle {p_u}^{\sharp \ell}, B_j \rangle \in L_{M'}$ where $\mathcal{E}(\{p_u\}) = \{e_u\}$ we have that $\langle p^{\sharp u}, B_j \rangle \in L_M$. Thus there exist for each $i \in \theta$ an event $e \in \mathcal{E}(W) \cup \mathcal{E}(R)$ and a transition $\mathtt{c_i} \xrightarrow{e}_C \mathtt{c'_i}$. Moreover, $\mathit{maxenabled}(\langle W/R \rangle, \mathtt{M}, \mathtt{q})$ is true because all the component configurations $\mathtt{c_i}$ for $i \leq n_M$ were also contained in $\mathtt{M}'$ and the configuration $\mathtt{c_s}$ has been replaced with an equivalent subnetwork configuration $\langle \langle \mathtt{c}_{\mathtt{n_M}+1}, ..., \mathtt{c}_{\mathtt{n}'} \rangle, \langle \mathtt{q} \rangle \rangle$ and by the acyclicity of $\mathcal{G}(M)$ no of the components $\{C'_{n_M+1}, ..., C'_{n'}\}$ with the exception of C'_u which is connected to B_j by links of L_M. Hence we have according to the rule (8) the required transition $\mathtt{M} \xrightarrow{\tau} \mathtt{M_1} \equiv \mathtt{M}[\bigwedge_{i \in \theta'} c_i := c'_i, q_j := q']$ of M where $\theta' \overset{\text{df}}{=} \{i \in \theta \parallel i \leq n_M\} \cup \{u\}$. Moreover, $(\mathtt{M_1}, \mathtt{M'_1}) \in \mathit{Rel}$ because of the arguments above.

Finally we have that M' is a network with m star topologies which by the induction hypothesis satisfies φ. As $M' \cong_{\varphi} M$, it follows that M satisfies φ.

- $k > 0$

In this case $\mathcal{G}(N)$ is cyclic and hence all the conditions $(1-5)$ must be taken into account. Let the result hold for some $k \geq 0$ and let N be a network satisfying $(1-5)$ which has the following form:

$$N \overset{\text{df}}{=} \langle\langle C_1, ..., C_{n_k}, C_{n_k+1}, ..., C_n\rangle, \langle B_1, ..., B_{m_k}, B_{m_k+1}, ..., B_m\rangle, L, Lrank\rangle$$

where

- $1 \leq n_k < n$
- $1 \leq m_k < m$
- $\|c\kappa(N)\| = k + 1$.

Further let $S \overset{\text{df}}{=} \langle\langle C_{n_k+1}, ..., C_n\rangle, \langle B_{m_k}, ..., B_m\rangle, L_S, Lrank_S\rangle$ be an open Ω-subnetwork of N for some cycle $\Omega \in c\kappa(N)$. Note that n_k and m_k are chosen in such a way that S must contain at least two components and at least two buses. This property is consistent with the fact that $\mathcal{G}(S)$ is required to be purely cyclic. We virtue of the condition (5) we know that S satisfies φ and we have to prove that also the entire network N satisfies φ.

We follow the proof by transforming the network N to a network N' in which the Ω-subnetwork S is replaced with a star topology S'. If we prove that such a transformation preserves the property φ and the conditions $(1-5)$ we achieve the result by application of the induction hypothesis to N' having only k cycles.

Let N' be defined in the following way:

$$N' \overset{\text{df}}{=} \langle\langle C_1, ..., C_{n_k}, C^*_{n_k+1}, ..., C^*_{n'}\rangle, \langle B_1, ..., B_{m_k}, B'\rangle, L_{N'}, Lrank_{N'}\rangle$$

where

- $S' \equiv \langle\langle C^*_{n_k+1}, ..., C^*_{n'}\rangle, \langle B'\rangle, L', \emptyset\rangle$ is an open network satisfying the conditions of lemma 8.33;
- $L_{N'} \overset{\text{df}}{=} (L(N) \setminus L(S)) \cup L'$;
- $Lrank_{N'} \overset{\text{df}}{=} Lrank_{\|L_{N'}}$.

Note that S' is a star topology of N' and $\|c\kappa(N')\| = k$. Moreover, according to definition of the link relation $L_{N'}$ the equation $freeports(N) = freeports(N')$ holds.

From the lemma 8.33 it follows that the reduction of S to S' preserves φ as $\Phi^a_o(S) \approx^{cl} \Phi^a_o(S')$. Hence we prove that the following claim holds:

$$\Phi^a_o(S) \approx^{cl} \Phi^a_o(S') \wedge N \text{ satisfies } (1-5) \Rightarrow \Phi^a(N) \approx^{cl} \Phi^a(N') \wedge N' \text{ satisfies } (1-5)$$

At first we prove that the transformation from N to N' preserves conditions $(1-5)$. Recall the assumption that conditions $(1-5)$ are satisfied in N.

1. Each component of N' is included also in N and there are no more components introduced in N'. Hence the condition (1) is satisfied in N'.
2. For each $j \in \{1, ..., m_k\}$ the bus B_j of N' is also included in N. It remains to be proved that $\Phi_B(B')$ satisfies φ. Note that $\Phi_B(B') = \Phi^a_{rL_{B'}}(S)$ according to the construction in lemma 8.33. By the initial assumption the conditions $(1,2,5)$ hold in N. Therefore each bus and component of S satisfies φ and moreover the entire S satisfies φ. Hence we have $\Phi^a(S)$ satisfies φ and we need $\Phi^a_{rL_{B'}}(S)$ satisfies φ. Note that it follows from the construction in lemma 8.33 that $\Phi^a_{rL_{B'}}(S)$ is isomorphic with $\Phi^a(S)$ up to bijection of transition labels. Hence with respect to the format of the property φ, its validity is preserved. Thus B' satisfies φ and the condition (2) holds in N'.
3. Let $j \leq m_k$ and $i > n_k$ such that $Llinks(C_i, B_j) \neq \emptyset$. This choice of i and j implies that B_j is directly connected to some component of the cycle S. Such a situation corresponds just to the condition (3). By the initial assumption we know that (3) holds in N and therefore $C_i \bowtie_\varphi B_j$. As $C_i \in \beta(\Omega)$, we have that C_i must be included also in N' as $C_i^* \equiv C_i$. Moreover, $Llinks(C_i, B_j) = L_{N'}links(C_i^*, B_j)$. Note that by lemma 8.33 there cannot exist another link of B_j to some component of S different from C_i. Therefore the arguments above are sufficient and the condition (3) is satisfied in N.
4. Let $j > m_k$ and $i \leq n_k$ such that $Llinks(C_i, B_j) \neq \emptyset$. Such a choice of i and j corresponds to the situation when B_j contained in the cycle S is directly connected to some component outside of S. Hence the condition (4) is employed here. By the initial assumption the condition (4) is satisfied in N and therefore $C_i \bowtie^N_\varphi B_j$. Note that C_i must be included also in the network N'. However, this is not the case of B_j. By the construction in lemma 8.33, there is a bus B' in N' which satisfies $Llinks(C_i, B_j) = L_{N'}links(C_i, B')$. We have to prove $C_i \bowtie^{N'}_\varphi B'$. By definition 8.17 we need $\Phi^a_{L_M}(M) \approx^{cl}_\varphi \Phi_B(\pi(B', L_M))$ where $M \overset{\text{df}}{=} \langle\langle C' \rangle, \langle \pi(B', L_M) \rangle, L_M, Lrank_M \rangle$ satisfying
   - $L_M \overset{\text{df}}{=} \{l \in links(L_{N'}, B') \| \exists p \in I(C_i).p^{\sharp_i} = port(l)\}$
   - $Lrank_M$ is $Lrank$ restricted to L_M
   - C' is defined as C_i with the interface and gate restricted to L_M

   As $L_M = Llinks(C_i, B_j) = L_{N'}links(C_i, B')$, it suffices to prove that $\pi(B', L_M) \approx^{cl}_\varphi \pi(B_j, L_M)$. But this follows from lemma 8.33. We only sketch the crucial arguments. Note that each transition of B_j which is taken into account is just an open transition of $\Phi^a_o(S)$ and by the construction of S' there must be an equivalent transition in $\Phi^a_o(S')$ inferred only of the transition in B' in terms of the rule (8^*). The opposite direction is symmetric.

Note that by lemma 8.33 there exists no another link of B_j to some component of S different from C_i. Therefore the arguments above are sufficient and the condition (4) is satisfied in N.

5. The condition (5) is satisfied in N' because for each cycle $\Omega \in c\kappa(N)$ the respective Ω-subnetwork is also Ω-subnetwork of N' with the only exception of S and there are no more cycles introduced in N'.

Subsequently we prove that the relation $Rel \overset{\text{df}}{=} \{((\langle\langle \mathtt{c}_1, ..., \mathtt{c}_{\mathtt{n}_\mathtt{k}}, \mathtt{c}_{\mathtt{n}_\mathtt{k}+1}, ..., \mathtt{c}_\mathtt{n}\rangle, \langle \mathtt{q}_1, ..., \mathtt{q}_{\mathtt{m}_\mathtt{k}}, \mathtt{q}_{\mathtt{m}_\mathtt{k}+1}, ..., \mathtt{q}_\mathtt{m}\rangle\rangle, \langle\langle \mathtt{c}_1, ..., \mathtt{c}_{\mathtt{n}_\mathtt{k}}, \mathtt{c}^*_{\mathtt{n}_\mathtt{k}+1}, ..., \mathtt{c}^*_{\mathtt{n}'}\rangle, \langle \mathtt{q}\rangle\rangle) \parallel \langle\langle \mathtt{c}_{\mathtt{n}_\mathtt{k}+1}, ..., \mathtt{c}_\mathtt{n}\rangle, \langle \mathtt{q}_{\mathtt{m}_\mathtt{k}+1}, ..., \mathtt{q}_\mathtt{m}\rangle\rangle \approx^{cl} \langle\langle \mathtt{c}^*_{\mathtt{n}_\mathtt{k}+1}, ..., \mathtt{c}^*_{\mathtt{n}'}\rangle, \langle \mathtt{q}\rangle\rangle\}$ is weak bisimulation.

First of all suppose the transition:

$$\mathtt{N} \equiv \langle\langle \mathtt{c}_1, ..., \mathtt{c}_{\mathtt{n}_\mathtt{k}}, \mathtt{c}_{\mathtt{n}_\mathtt{k}+1}, ..., \mathtt{c}_\mathtt{n}\rangle, \langle \mathtt{q}_1, ..., \mathtt{q}_{\mathtt{m}_\mathtt{k}}, \mathtt{q}_{\mathtt{m}_\mathtt{k}+1}, ..., \mathtt{q}_\mathtt{m}\rangle\rangle \xrightarrow{e^{\sharp_i}} \mathtt{N}_1 \equiv \mathtt{N}[c_i := c']$$

for some $i \in \{1, ..., n\}$. If $i \leq n_k$ then by the argument that each free port of N is also a free port of N' the same component transition is performed in N' and hence we have the transition in N':

$$\mathtt{N}' \equiv \langle\langle \mathtt{c}_1, ..., \mathtt{c}_{\mathtt{n}_\mathtt{k}}, \mathtt{c}^*_{\mathtt{n}_\mathtt{k}+1}, ..., \mathtt{c}^*_{\mathtt{n}'}\rangle, \langle \mathtt{q}\rangle\rangle \overset{e^{\sharp_i}}{\Rightarrow} \mathtt{N}'_1 \equiv \mathtt{N}'[c_i := c']$$

Moreover, configurations of S and S' do not contain the component configuration $\mathtt{c}_\mathtt{i}$, hence no transition occurs there and we have $(\mathtt{N}_1, \mathtt{N}'_1) \in Rel$. If we take an internal action τ instead of the action $e^{\sharp_i}$ of the component C_i then the result is achieved by the same arguments as above.

If $i > n_k$ the supposed transition causes a transition of S of the form $\mathtt{S} \equiv \langle\langle \mathtt{c}_{\mathtt{n}_\mathtt{k}+1}, ..., \mathtt{c}_\mathtt{n}\rangle, \langle \mathtt{q}_{\mathtt{m}_\mathtt{k}+1}, ..., \mathtt{q}_\mathtt{m}\rangle\rangle \xrightarrow{\gamma} \mathtt{S}_1 \equiv \mathtt{S}[c_i := c']$ where $\gamma \doteq \{e^{\sharp_i}\}$. Note that S' is constructed from S by the lemma 8.33 and such construction ensures that all free ports of S are preserved in S'. Moreover, no free port of S can be included in some cooperation of B' because of the definition of the link relation $L(S')$. Thus by equivalence of $\Phi^a(S)$ and $\Phi^a(S')$ we have a transition $\mathtt{S}' \equiv \langle\langle \mathtt{c}^*_{\mathtt{n}_\mathtt{k}+1}, ..., \mathtt{c}^*_{\mathtt{n}'}\rangle, \langle \mathtt{q}\rangle\rangle \overset{\gamma}{\Rightarrow} \mathtt{S}'_1 \equiv \mathtt{S}'[c^*_i := c']$ and $\mathtt{S}_1 \approx^{cl} \mathtt{S}'_1$. Hence $\mathtt{N}' \overset{e^{\sharp_i}}{\Rightarrow} \mathtt{N}'_1 \equiv \mathtt{N}'[c^*_i := c']$ and $(\mathtt{N}_1, \mathtt{N}'_1) \in Rel$. If we take an internal action τ instead of the action $e^{\sharp_i}$ of the component C_i then we have one of the following two situations:

1. If $C_i \in \beta(\Omega)$ then the result is achieved by the same arguments as above.

2. If $C_i \notin \beta(\Omega)$ then by construction in the lemma 8.33 the internal action of C_i is not simulated by the bus B'. Hence $\mathtt{S} \equiv \langle\langle \mathtt{c}_{\mathtt{n_k}+1}, ..., \mathtt{c}_{\mathtt{n}}\rangle, \langle \mathtt{q}_{\mathtt{m_k}+1}, ..., \mathtt{q}_{\mathtt{m}}\rangle\rangle \overset{\emptyset/\emptyset}{\rightarrow} \mathtt{S}_1 \equiv \mathtt{S}[c_i := c']$ implies $\mathtt{S}' \equiv \langle\langle \mathtt{c}^*_{\mathtt{n_k}+1}, ..., \mathtt{c}^*_{\mathtt{n}'}\rangle, \langle \mathtt{q}\rangle\rangle \equiv \mathtt{S}'_1$ and $\mathtt{S}_1 \approx^{cl} \mathtt{S}'_1$. Thus we have $\mathtt{N}' \equiv \mathtt{N}'_1$ and $(\mathtt{N}_1, \mathtt{N}'_1) \in \mathit{Rel}$.

Now suppose the transition:

$$\mathtt{N} \equiv \langle\langle \mathtt{c}_1, .., \mathtt{c}_{\mathtt{n_k}}, \mathtt{c}_{\mathtt{n_k}+1}, .., \mathtt{c}_{\mathtt{n}}\rangle, \langle \mathtt{q}_1, .., \mathtt{q}_{\mathtt{m_k}}, \mathtt{q}_{\mathtt{m_k}+1}, .., \mathtt{q}_{\mathtt{m}}\rangle\rangle \overset{\tau}{\rightarrow}$$
$$\mathtt{N}_1 \equiv \mathtt{N}[\textstyle\bigwedge_{i\in\theta} c_i := c'_i, q_j := q']$$

where $\theta \subseteq \{1, ..., n\}$ and $j \in \{1, ..., m\}$. Here we distinguish three different situations determined by the actual choice of j and θ.

1. If $j \leq m_k$ and moreover $\theta \subseteq \{1, ..., n_k\}$ then we reach the result by isomorphism of the relevant subnetworks of N and N' containing $\{C_i \| i \in \theta\}$ and the bus B_j.
2. Let $j \leq m_k$ and there is some $u \in \theta$ such that $u > n_k$. Note that by lemma 8.33 there can be at most one such u. In order to realise the supposed transition the bus B_j must perform $\mathtt{q_j} \overset{\gamma}{\rightarrow}_{B_j} \mathtt{q}'$ for some $\gamma \in \mathit{coop}(B_j)$ and C_u must evolve $\mathtt{c_u} \overset{e}{\rightarrow} \mathtt{c}'$ for some e such that $\mathcal{P}(e^{\sharp_u}) \in \gamma$. As $C_u \in \beta(\Omega)$ and links between C_u and B_j are preserved also in N', it follows that $C^*_u \equiv C_u$ and there exists a transition $\mathtt{N}' \overset{\tau}{\rightarrow} \mathtt{N}'_1 \equiv \mathtt{N}'[\bigwedge_{i\in\theta'} c_i := c'_i, c^*_u := c', q_j := q']$ where $\theta' \overset{\mathrm{df}}{=} \theta \setminus \{u\}$ and moreover $(\mathtt{N}_1, \mathtt{N}'_1) \in \mathit{Rel}$ because $\mathtt{S} \overset{\gamma'}{\rightarrow} \mathtt{S}_1 \equiv \mathtt{S}[c_u := c']$ implies $\mathtt{S}' \overset{\gamma'}{\Rightarrow} \mathtt{S}'_1 \equiv \mathtt{S}'[c_u := c']$ where $\gamma' \doteq \{e^{\sharp_u}\}$ and $\mathtt{S}_1 \approx^{cl} \mathtt{S}'_1$.
3. Let $j > m_k$ and θ of the form $\theta = \theta_1 \cup \theta_2$ such that $\theta \neq \emptyset$, $\theta_1 \subseteq \{1, ..., n_k\}$, and $\theta_2 \subseteq \{n_k + 1, ..., n\}$. Furthermore, denote the index set of components forming the cycle border $\beta(\Omega)$ as θ_β satisfying $\theta_\beta \subseteq \theta_2$, $\theta_\beta \overset{\mathrm{df}}{=} \{i \in \theta_2 \| C_i \in \beta(\Omega)\}$. The supposed τ transition implies that B_j must change its state according to the rule (8) by transition $\mathtt{q_j} \overset{\gamma}{\rightarrow}_{B_j} \mathtt{q}'$ for some $\gamma \in \mathit{coop}(B_j)$ which has the form $\gamma = \gamma_1 \cup \gamma_\beta \cup \gamma'$ where the cooperation parts γ_1, γ_β, $\gamma' \subseteq \gamma$ are distinguished in the following way:
   - $\gamma_1 \doteq \{p^{\sharp_i} \in \gamma \| i \in \theta_1\}$
   - $\gamma_\beta \doteq \{p^{\sharp_i} \in \gamma \| i \in \theta_\beta\}$
   - $\gamma' \doteq \{p^{\sharp_i} \in \gamma \| i \in \theta_2 \setminus \theta_\beta\}$

   By the rule (8*) we have in S a transition $\mathtt{S} \equiv \langle\langle \mathtt{c}_{\mathtt{n_k}+1}, ..., \mathtt{c}_{\mathtt{n}}\rangle, \langle \mathtt{q}_{\mathtt{m_k}+1}, ..., \mathtt{q}_{\mathtt{m}}\rangle\rangle \xrightarrow{\gamma_1\cup\gamma_\beta\cup\gamma''} \mathtt{S}_1 \equiv \mathtt{S}[\bigwedge_{i\in\theta_2} c_i := c'_i, q_j := q']$ where

$$\gamma'' \overset{\mathrm{df}}{=} \begin{cases} \gamma' & B_j \in \beta(\Omega) \\ \emptyset & B_j \notin \beta(\Omega) \end{cases}$$

By $\mathtt{S} \approx^{cl} \mathtt{S}'$ from lemma 8.33 it follows that there exists a configuration $\mathtt{S}'_1$ of S' and a transition $\mathtt{S}' \equiv \langle\langle \mathtt{c}^*_{\mathtt{n_k}+1}, \ldots, \mathtt{c}^*_{\mathtt{m}'}\rangle, \langle \mathtt{q}\rangle\rangle \overset{\gamma_1 \cup \gamma_\beta \cup \gamma''}{\Rightarrow} \mathtt{S}'_1 \equiv \mathtt{S}'[\bigwedge_{i\in\theta_\beta} c^*_i := c'_i, q := q']$ such that $\mathtt{S}_1 \approx^{cl} \mathtt{S}'_1$. Moreover, $cm(B') = \Phi^a_{oL_{B'}}(S)$ with respect to lemma 8.33 and hence it must hold that $\mathtt{q} \xrightarrow{\gamma_1 \cup \gamma_\beta \cup \gamma''}_{B'} \mathtt{q}'$.

If $B_j \in \beta(\Omega)$ then $\gamma'' = \gamma'$. As for each $i \in \theta_1$ the component C_i is involved in the respective synchronisation in N, C_i must perform the transition $\mathtt{c_i} \xrightarrow{e} \mathtt{c}'_\mathtt{i}$ for some e such that $\mathcal{P}(e^{\sharp_i}) \in \gamma_1$. Note that for each $i \in \theta_1$ the configuration $\mathtt{c_i}$ is directly included in $\mathtt{N}'$ because of the definition of N'. Moreover, the relevant links between each component C_i and the bus B_j are remapped to links between C_i and B' by construction of N'. Similarly, for each $u \in \theta_2$ the component C_u performs the transition $\mathtt{c_u} \xrightarrow{e} \mathtt{c}'_\mathtt{u}$ for some e satisfying $\mathcal{P}(e^{\sharp_u}) \in \gamma_\beta \cup \gamma'$. Note that here we cannot say in general that C_u is directly included in N'. It happens only in the case when $C_u \in \beta(\Omega)$. In that case C_u is included in N' as $C^*_u \equiv C_u$. Moreover, the relevant links between C_u and the bus B_j are remapped to links between C^*_u and B' by construction of N'. Therefore there is a transition $\mathtt{c}^*_\mathtt{u} \equiv \mathtt{c_u} \xrightarrow{e} \mathtt{c}'_\mathtt{u}$ for some e such that $\mathcal{P}(e^{\sharp_u}) \in \gamma_\beta$. In the case of $C_u \notin \beta(\Omega)$ the respective transition of C_u is according to definition of $cm(B')$ reflected in γ'' because it is included in $L_{B'}$-observation of S. As all the relevant links between C_u and the bus B_j are remapped to links between C^*_u and B' by construction of N', we have a transition $\mathtt{N}' \xrightarrow{\tau} \mathtt{N}'_1 \equiv \mathtt{N}'[\bigwedge_{i\in\theta_1} c_i := c'_i, \bigwedge_{i\in\theta_\beta} c^*_i := c'_i, q := q']$ and by the arguments above also $(\mathtt{N}_1, \mathtt{N}'_1) \in Rel$.

If $B_j \notin \beta(\Omega)$ then $\gamma_1 \cup \gamma_\beta = \langle\emptyset/\emptyset\rangle$ and $\gamma'' = \langle\emptyset/\emptyset\rangle$. Hence $\theta_1 = \theta_\beta = \emptyset$ and in this case $\mathtt{S}'_1 \equiv \mathtt{S}'[q := q']$. Thus there must be according to the lemma 8.33 a transition $\mathtt{q} \xrightarrow{\emptyset/\emptyset}_{B'} \mathtt{q}'$ in B'. From that follows by the rule (10) a transition $\mathtt{N}' \xrightarrow{\tau} \mathtt{N}'_1 \equiv \mathtt{N}'[q := q']$ and by the arguments above also $(\mathtt{N}_1, \mathtt{N}'_1) \in Rel$.

The opposite simulation is proved similarly. At first we suppose the transition:

$$\mathtt{N}' \equiv \langle\langle \mathtt{c}_1, .., \mathtt{c_{n_k}}, \mathtt{c}^*_{\mathtt{n_k}+1}, .., \mathtt{c}^*_{\mathtt{n}'}\rangle, \langle \mathtt{q}_1, .., \mathtt{q_{m_k}}, \mathtt{q}\rangle\rangle \xrightarrow{e^{\sharp_i}} \mathtt{N}'_1 \equiv \mathtt{N}'[c_i := c'_i]$$

Both cases when $i \leq n_k$ or $i > n_k$ are symmetric to the respective cases of the opposite simulation proved above. When we take the τ action of the ith component instead of $e^{\sharp_i}$ then in the case $i \leq n_k$ the proof is also symmetric. If $i > n_k$ then we have $C^*_i \equiv C_i$ where $C_i \in \beta(\Omega)$ of the network N and hence N' can perform the required τ transition and we establish the needed arguments for the respective simulation in similar way like in the opposite case.

Now suppose the transition:

$$\mathtt{N'} \equiv \langle\langle \mathtt{c_1},..,\mathtt{c_{n_k}},\mathtt{c^*_{n_k+1}},..,\mathtt{c^*_{n'}}\rangle,\langle \mathtt{q_1},..,\mathtt{q_{m_k}},\mathtt{q}\rangle\rangle \xrightarrow{\mathcal{T}}$$
$$\mathtt{N'_1} \equiv \mathtt{N'}[\textstyle\bigwedge_{i\in\theta} c_i := c'_i, q := q']$$

The case when $\theta \subseteq \{1,...,n_k\}$ and $j \leq m_k$ is treated symmetrically to the respective case (1) of the opposite simulation. The same holds for the case of $\theta \cap \{n_k,...,n'\} \neq \emptyset$ and $j \leq m_k$, here the situation is symmetrical to the case (2) of the opposite simulation.

In the case when the bus B' causes the supposed transition and $\theta = \theta_1 \cup \theta_2$ where $\theta_1 \subseteq \{1,...,n_k\}$ and $\theta_2 \subseteq \{n_k+1,...,n'\}$, the proof is different than in the respective situation of the opposite simulation. Note that in contrary to the opposite case we have all the component configurations of N' which are involved in the particular synchronisation included also in N. If we denote γ the cooperation $\gamma \doteq \{p^{\sharp_i} \| i \in \theta_1, \mathtt{c_i} \xrightarrow{p}_{C_i} \mathtt{c'_i}\} \cup \{p^{\sharp_i} \| i \in \theta_2, \mathtt{c^*_i} \xrightarrow{p}_{C^*_i} \mathtt{c'_i}\}$ containing the relevant ports currently involved in the synchronisation then there exists $\gamma' \in \mathrm{Coops}$ such that $\gamma \subseteq \gamma'$ and B' performs just the transition $\mathtt{q} \xrightarrow{\gamma'}_{B'} \mathtt{q'}$. Suppose $\gamma' \neq \langle\emptyset/\emptyset\rangle$. Note that each $p^{\sharp_u}$ of the cooperation part $\gamma' \setminus \gamma$ for some $u \notin \theta$ corresponds to a particular port of component C_u in N which satisfies $C_u \notin \beta(\Omega)$. The transition of B' implies by the rule (8*) an open transition of S' such that $\mathtt{S'} \equiv \langle\langle \mathtt{c^*_{n_k+1}},...,\mathtt{c^*_{n'}}\rangle,\langle q\rangle\rangle \xrightarrow{\gamma'} \mathtt{S'_1} \equiv \mathtt{S'}[\bigwedge_{i\in\theta_2} c^*_i := c'_i, q := q']$. By $\mathtt{S'} \approx^{cl} \mathtt{S}$ we have a transition $\mathtt{S} \equiv \langle\langle \mathtt{c_{n_k+1}},...,\mathtt{c_n}\rangle,\langle \mathtt{q_{m_k+1}},...,\mathtt{q_m}\rangle\rangle \overset{\gamma'}{\Rightarrow} \mathtt{S_1} \equiv \mathtt{S}[\bigwedge_{i\in\theta_2} c_i := c'_i, \bigwedge_{i\in\theta'} c_i := c'_i, q_j := q']$ for some $j \in \{m_k+1,...,m\}$ where $\theta' \overset{\mathrm{df}}{=} \{i \in \{1,...,n'\} \| \exists p \in ports(I(C_i)).p^{\sharp_i} \in \gamma' \setminus \gamma\}$ and $\mathtt{S'_1} \approx^{cl} \mathtt{S_1}$.

By the construction in lemma 8.33 all relevant links between each component C_i, $i \in \theta$ and the bus B' are bijectively remapped to links between C_i and B_j and moreover for each port $p^{\sharp_u} \in \gamma' \setminus \gamma$ there must be a link from some component $C'_u \notin \beta(\Omega)$ of N to the bus B_j. It follows by the rule (8) that there is a transition of N such that $\mathtt{N} \overset{\mathcal{T}}{\Rightarrow} \mathtt{N_1} \equiv \mathtt{N}[\bigwedge_{i\in\theta} c_i := c'_i, \bigwedge_{i\in\theta'} c_i := c'_i, q_j := q']$ and by the arguments above $(\mathtt{N_1},\mathtt{N'_1}) \in Rel$.

If $\gamma' = \langle\emptyset/\emptyset\rangle$ then the $\langle\emptyset/\emptyset\rangle$-transition of B' implies by the rule (8*) that $\mathtt{S'} \equiv \langle\langle \mathtt{c^*_{n_k+1}},...,\mathtt{c^*_{n'}}\rangle,\langle q\rangle\rangle \xrightarrow{\emptyset/\emptyset} \mathtt{S'_1} \equiv \mathtt{S'}[q := q']$. By $\mathtt{S'} \approx^{cl} \mathtt{S}$ we have a transition $\mathtt{S} \equiv \langle\langle \mathtt{c_{n_k+1}},...,\mathtt{c_n}\rangle,\langle \mathtt{q_{m_k+1}},...,\mathtt{q_m}\rangle\rangle \overset{\emptyset/\emptyset}{\Rightarrow} \mathtt{S_1}$ for some $\mathtt{S'_1}$ such that $\mathtt{S'_1} \approx^{cl} \mathtt{S_1}$. Hence we have a transition $\mathtt{N} \overset{\mathcal{T}}{\Rightarrow} \mathtt{N'}[\bigwedge_{i\in\theta'} c_i := c'_i, q_j := q'_j]$ where $\theta' \subseteq \{n_k+1,...,n\} \setminus \theta_2$ and $j \in \{m_k+1,...,m\}$ such that $B_j \notin \beta(\Omega)\}$. By the arguments above it holds that $(\mathtt{N_1},\mathtt{N'_1}) \in Rel$. Note that q_j may not be necessarily different from q'_j. The situation when $q_j \equiv q'_j$ corresponds to internal transition of some component C_u where $\{u\} = \theta'$.

Finally we have a network N' with $c\kappa(N') = k$ and satisfying all the conditions $(1-5)$. Hence by the induction hypothesis N' satisfies φ. We have just proved that $N' \cong_{\varphi} N$ and thus N satisfies φ too.

□

We follow with discussion of synchronous gate interoperability correctness. More precisely, we show that the problem of checking if the gate does not violate some property φ, which has been illustrated in Section 8.4.1, can be reduced to the network interoperability checking problem already treated in previous paragraphs.

We begin with the situation when some synchronous gate relates components which are of independent partitions of the network dependency graph. As the following statements follow from the fact that a universal and synchronous gates can be considered as particular memory-less buses, we only present the lemmas end the theorem without proofs.

Lemma 8.34 *Let $N = \langle\langle C_1, ..., C_n\rangle, \langle B_1, ..., B_m\rangle, L, Lrank\rangle \in \mathbf{T}_{\mathrm{st}}$ a network and $C = \langle N, I, G\rangle \in \mathbf{CT}_{\mathrm{st}}$ a component with an interface I including the port $p \in ports(I)$ and G a gate. Furthermore, let $\mathcal{I} \subseteq \{1, ..., n\}$ some nonempty index set satisfying $\{C_i \parallel i \in \mathcal{I}\} \subseteq \{C_1, ..., C_n\}$ so that each C_i ($i \in \mathcal{I}$) is included in $\mathcal{G}(N)$ in a* **separate** *maximally connected partition than each component of $\{C_j \parallel j \in \mathcal{I} \setminus \{i\}\}$. Moreover, let $g \in map_G$ a gate mapping which has the following properties:*

- *$type(g) = \times$*
- *For each $i \in \mathcal{I}$ there exists $e \in bbox(C_i)$ so that $g(e^{\sharp_i}) = p$.*
- *For each $i \in \{1, ..., n\} \setminus \mathcal{I}$ it holds that $g(e^{\sharp_i})$ is not defined for all $e \in bbox(C_i)$.*

Let $B_g = \langle Q, T, q_0\rangle \in \mathrm{Buses}$ a bus defined in the following way:

- *$Q \overset{\mathrm{df}}{=} \{q_0\}$*
- *$T \overset{\mathrm{df}}{=} \{\langle q_0, \gamma, q_0\rangle \parallel \gamma \doteq \bigcup_{i\in\mathcal{I}}\{e^{\sharp_i} \parallel g(e^{\sharp_i}) = p\}\}$*

If for each $i \in \mathcal{I}$ component C_i satisfies φ and the compatibility $C_i \bowtie_{\varphi}^{N} B_g$ then the gate mapping g preserves the property φ.

Lemma 8.35 *Let $N = \langle\langle C_1, ..., C_n\rangle, \langle B_1, ..., B_m\rangle, L, Lrank\rangle \in \mathbf{T}_{\mathrm{st}}$ a network and $C = \langle N, I, G\rangle \in \mathbf{CT}_{\mathrm{st}}$ a component with an interface I including the port $p \in ports(I)$ and G a gate. Furthermore, let $\mathcal{I} \subseteq \{1, ..., n\}$ some nonempty index set satisfying $\{C_i \parallel i \in \mathcal{I}\} \subseteq \{C_1, ..., C_n\}$ so that each C_i ($i \in \mathcal{I}$) is included in $\mathcal{G}(N)$ in the* **same** *maximally connected partition as each component of $\{C_j \parallel j \in \mathcal{I} \setminus \{i\}\}$. Moreover, let $\mathcal{J} \subseteq \{1, ..., m\}$ an index set defined as the set $\{j \in \mathcal{J} \parallel \exists i \in \mathcal{I}. Llinks(C_i, B_j) \neq \emptyset\}$. Finally, let $g \in map_G$ a gate mapping which has the following properties:*

- $type(g) = \times$
- *For each* $i \in \mathcal{I}$ *there exists* $e \in bbox(C_i)$ *so that* $g(e^{\sharp_i}) = p$.
- *For each* $i \in \{1, ..., n\} \setminus \mathcal{I}$ *it holds that* $g(e^{\sharp_i})$ *is not defined for all* $e \in bbox(C_i)$.

Let $B_g = \langle Q, T, q_0 \rangle \in \text{Buses}$ *a bus defined in the following way:*

- $Q \overset{\text{df}}{=} \{q_0\}$
- $T \overset{\text{df}}{=} \{\langle q_0, \gamma, q_0 \rangle \parallel \gamma \doteq \bigcup_{i \in \mathcal{I}} \{e^{\sharp_i} \parallel g(e^{\sharp_i}) = p\}\}$

Furthermore let $N' \overset{\text{df}}{=} \langle \langle C'_1, ..., C'_{n'} \rangle, \langle B'_1, ..., B'_{m'}, B_g \rangle, L', Lrank' \rangle$ *a subnetwork of* N *satisfying*

- $n' \overset{\text{df}}{=} \|\mathcal{I}\|$
- $L' \overset{\text{df}}{=} L \cup \{\langle e^{\sharp_i}, B_g \rangle \parallel g(e^{\sharp_i}) \text{ defined }\}$
- $Lrank' \overset{\text{df}}{=} Lrank_{\|_{L'}}$
- $\{B'_1, ..., B'_{m'}\} = \{B_j \parallel j \in \mathcal{J}\}$
- *For each* $i \in \mathcal{I}$, $C_i = \langle T_i, I_i, G_i \rangle$ *is transformed to* $C'_i \overset{\text{df}}{=} \langle T_i, I'_i, G'_i \rangle$ *where*
  - $I'_i \overset{\text{df}}{=} I_{i\|_P}$ *where* $P \overset{\text{df}}{=} \{p \in ports(I_i) \parallel \exists l \in L'. p^{\sharp_i} = port(l)\}$
  - $G'_i \overset{\text{df}}{=} G_{i\|_{I'_i}}$

If N' *satisfies* φ *then the gate mapping* g *preserves the property* φ.

Theorem 8.36 *Let* $C = \langle N, I, G \rangle \in \mathbf{CT}_{\text{st}}$ *a component. If each gate mapping* $g \in map_G$ *such that* $type(g) = \times$ *preserves* φ *and* N *satisfies* φ *then* C *satisfies* φ.

8.5 Additional Notes

There is an exhaustive piece of work by Bernardo et al. on the topic of interoperability checking of architectural descriptions formalised in traditional process algebraic framework. In [BCD02] it has been proved that for checking of an acyclic component topology it suffices to check interaction compatibility of all pairs of mutually connected components. The notion of such compatibility is based on a weak bisimilarity of the two components in a pair. Abstraction of both components is taken comprising only the actions of mutual interaction, while all the other actions are hidden. To overcome the potential problem with internal nondeterminism of components, the compatibility checking is realised in such a way that one of the components is checked on weak bisimilarity against the behaviour of its parallel composition with the other component.

We show that the similar approach can be applied to the behavioural model of VCN. In contrast to [BCD02], dependency graph of an architecture in VCN is bipartite, as VCN distinguishes connectors and components as two semantically different members of the architecture. In other words, we have to define the notion of architectural compatibility between its two arbitrary adjacent nodes, which are always a connector and a component. Moreover, VCN introduces set-labelled transition operational semantics to capture behavioural model of connectors. This extension lifts the expressiveness of the coordination model. In this paper, the notion of architectural interoperability is revisited and extended to fulfil the needs of such a setting. Especially, there is no traditional notion of the parallel composition operator in VCN and therefore the congruence results applied in [BCD02] cannot be used here. However, the main result of this paper shows that the framework of [BCD02] applies with some extension also to the VCN setting of an architectural description.

Chapter 9

Prototype Implementation

To support the visual design in VCN, especially in terms of primary evaluation of VCN features, we have developed VCNE [Sim, Reh, Reh06] — a prototype of an editor for Visual Coordination Networks. This editor is implemented in Java and hence allows multi-platform use. An example of a specification in this editor is depicted in Figure 9.1.

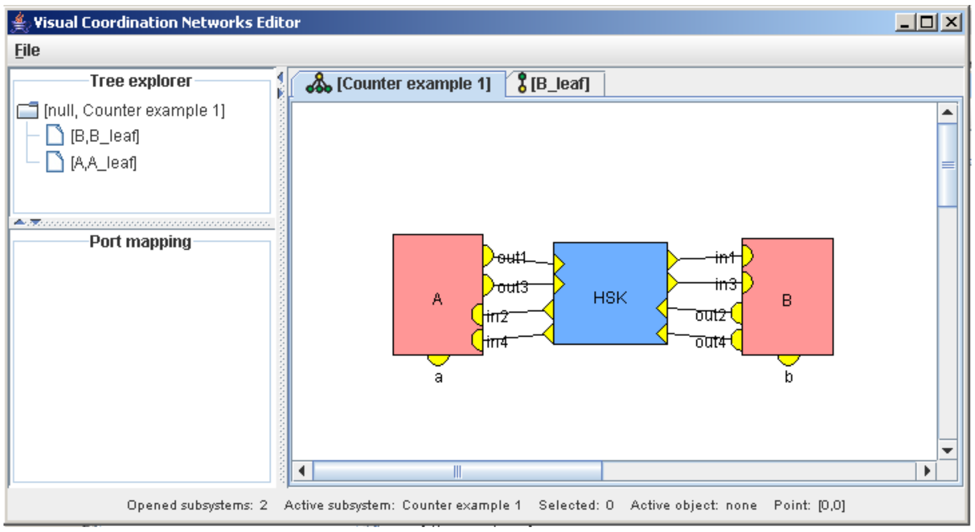


Figure 9.1: An example of a network in VCNE

Currently supported features of VCNE are as follows, concerning the particular features of VCN.

Visual Notation

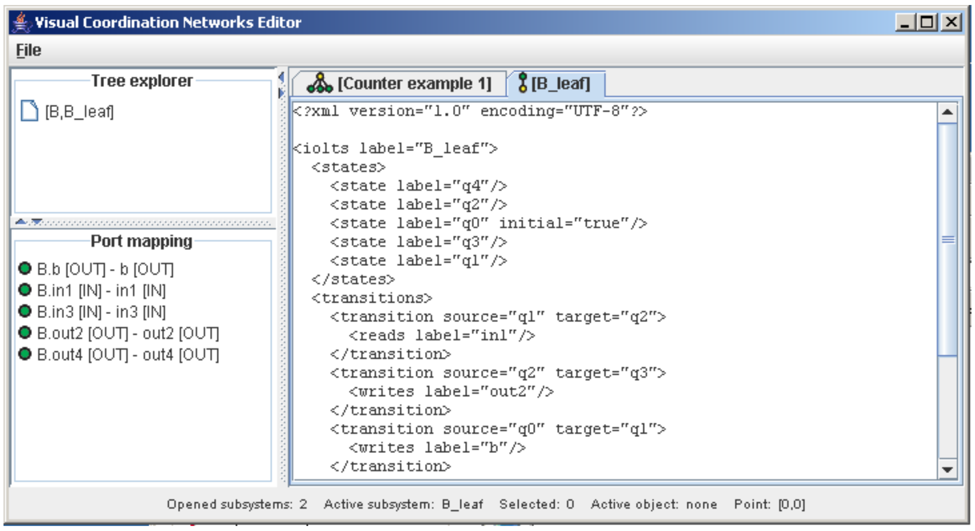
With the exception of port roles, full graphical notation of VCN is supported by VCNE.

Representation of Networks

Networks are represented in XML format which allows modular embedding of subnetworks. An entire architecture architecture is represented in VCNE directly as a tree and can be stored in just one XML file (`*.pat`). This way of representing VCN networks also enables reuse of architectures as component attached to other architectures.

Hierarchy

The hierarchy is supported with respect to the tree representation mentioned above. However, concerning the kind of supported gates, only one-to-one mapping are currently supported.



● B.b [OUT] - b [OUT]
● B.in1 [IN] - in1 [IN]
● B.in3 [IN] - in3 [IN]
● B.out2 [OUT] - out2 [OUT]
● B.out4 [OUT] - out4 [OUT]

```
<?xml version="1.0" encoding="UTF-8"?>

<iolts label="B_leaf">
  <states>
    <state label="q4"/>
    <state label="q2"/>
    <state label="q0" initial="true"/>
    <state label="q3"/>
    <state label="q1"/>
  </states>
  <transitions>
    <transition source="q1" target="q2">
      <reads label="in1"/>
    </transition>
    <transition source="q2" target="q3">
      <writes label="out2"/>
    </transition>
    <transition source="q0" target="q1">
      <writes label="b"/>
    </transition>
```

Opened subsystems: 2   Active subsystem: B_leaf   Selected: 0   Active object: none   Point: [0,0]

Figure 9.2: XML representation of leaf behavioural model in VCNE

Behavioural Model

The behavioural model is represented in terms of labelled transition systems implemented in XML format, as it is demonstrated in Figure 9.2. Specification of leaves is currently supported only in terms of direct specification of the transition system. Similarly, also the behaviour of buses has to be explicitly given provided that cooperation machines are also represented in XML. VCNE implements the semantics of network composition operator and offers automatised construction of a transition system for any level of the specification. Moreover, the generated transition system can be stored as a set of CCS expressions in the syntax of Concurrency Workbench of New Century [CS96], which allows model checking and equivalence checking to be applied on architectures.

Concerning the possibility of visual specification of leaf behaviour, we aim to adapt the visual editor PAXION [dM04] for description of state-transition systems. Although PAXION has been originally developed for the purpose of visual description of Buchi automata, it has tight relations with XML structures of VCNE.

Support for Consistency and Compatibility Checking

At first, static consistency analysis of architectures is implemented, provided that designer cannot specify a VCN which does not conform to the definition given in Chapter 5.

For the purpose of behavioural analysis, i.e., the architectural compatibility checking in terms of Chapter 8, VCNE supports synthesis of all the transition systems needed for respective bisimulation checking. The result is stored as a definition of CWB-NC CCS processes, hence this way the compatibility test can be applied on an architecture. VCNE currently allows checking of acyclic topologies only.

Chapter 10

A Case-Study: Rail Line Signalling

In this chapter, we evaluate the VCN language on a real example of a railway automatic line signalling system. In design of automatic railway systems, ad hoc methods are typically applied. There are approaches to employ formal methods in design of such systems [Pen06, BCJ$^+$04]. To our knowledge, none of such approaches uses an architectural description method. However, embedded railway systems such as line segmentation and signalling are composed of a number of components. We demonstrate how the VCN language can be employed for formalization of such systems.

10.1 Automatic Line Signalling

A typical problem with high-speed trains is that drivers cannot stop their trains within sighting distance of another train or within sighting distance of a signal [Pen06]. Therefore, automatic signalling is used on some railway lines.

A line is considered to be divided into segments $l = \langle s_1, s_2, ..., s_{i-1}, s_i, s_{i+1}, ..., s_n \rangle$. Such a line l connects two stations A and B. The line can be in just one of three modes – opened in direction from A to B, opened in the opposite direction, or closed. Each segment of a line can be in two states – $segFree$, when no train is present in the segment, and $segOccupied$, if a train is detected in the segment. The signal of each segment can have one of three colours for the currently set direction, or it can be switched off.

10.1.1 First Segment of the Line

The first segment of the line (s_1) has attached only a one automatically controlled signal. In particular, this is the signal of the $B \rightarrow A$ direction. The signal of the opposite direction is controlled manually from the station A.

There are four possible states of the signal have the following specification:

1. *Red* – if and only if the line l is opened in the $B \to A$ direction and the segment s_1 is occupied;
2. $Green$ – if and only if the line is opened in $B \to A$ direction, the segment s_1 is free, and the station A is opened for receiving trains from B;
3. $Yellow$ – if and only if the line is opened in $B \to A$ direction, the segment s_1 is free, and the station A is closed for receiving trains from B;
4. *Off* – if and only if the line is opened in $A \to B$ direction or when it is closed.

10.1.2 Inner Segments of the Line

Each inner segment s_i for $i \in \{2, ..., n-1\}$ is equipped with two automatic signals each for the particular direction. Specification of the ith $A \to B$ signal is the following:

1. *Red* – if and only if the line l is opened in the $A \to B$ direction and the segment s_i is occupied;
2. $Green$ – if and only if the line is opened in $A \to B$ direction, the segments s_i and s_{i+1} are free;
3. $Yellow$ – if and only if the line is opened in $A \to B$ direction, the segment s_i is free, and the segment $si + 1$ is occupied;
4. *Off* – if and only if the line is opened in $B \to A$ direction or when it is closed.

States of the ith $B \to A$ signal are the following:

1. *Red* – if and only if the line l is opened in the $B \to A$ direction and the segment s_i is occupied;
2. $Green$ – if and only if the line is opened in $B \to A$ direction, the segments s_i and s_{i-1} are free;
3. $Yellow$ – if and only if the line is opened in $B \to A$ direction, the segment s_i is free, and the segment $si - 1$ is occupied;
4. *Off* – if and only if the line is opened in $A \to B$ direction or when it is closed.

10.1.3 Last Segment of the Line

The last segment of the line has only one automatic signal in the $A \to B$ direction. The opposite signal is controlled manually from the station B. The signal has the following specification:

1. Red – if and only if the line l is opened in the $A \to B$ direction and the segment s_n is occupied;
2. $Green$ – if and only if the line is opened in $A \to B$ direction, the segment s_n is free, and the station B is opened for receiving trains from A;
3. $Yellow$ – if and only if the line is opened in $A \to B$ direction, the segment s_n is free, and the station B is closed for receiving trains from A;
4. *Off* – if and only if the line is opened in $B \to A$ direction or when it is closed.

10.1.4 Architecture of the Line Signalling System

The architecture of the line signalling system for two inner segments is formalized by VCN networks depicted in Figure 10.1 and Figure 10.2. The two buses appearing in both networks of the architecture hierarchy employs the same coordination model. A bus class for this coordination model has the following specification:

$$\begin{aligned}
&\mathcal{B}(\mathtt{In},\mathtt{Out},\mathtt{rank},\mathtt{capacity}) := \{\\
&\quad \mathtt{In},\mathtt{Out},\mathtt{rank} \neq \emptyset \wedge \mathtt{capacity} = 1\\
&\quad \mathtt{i}\Uparrow\mathtt{ab}/?\mathtt{ba},\{\mathtt{o_1},...,\mathtt{o_n}\} \wedge \mathtt{rank(i)} = 1 \wedge \textstyle\bigwedge_{j=1}^{n} \mathtt{rank(o_j)} = 1\\
&\quad \mathtt{i}\Uparrow\mathtt{ab}/\!\downarrow\!\mathtt{ba},\{\mathtt{o_1},...,\mathtt{o_n}\} \wedge \mathtt{rank(i)} = 1 \wedge \textstyle\bigwedge_{j=1}^{n} \mathtt{rank(o_j)} = 1\\
&\quad \mathtt{i}\Uparrow\mathtt{ba}/?\mathtt{ab},\{\mathtt{o_1},...,\mathtt{o_n}\} \wedge \mathtt{rank(i)} = 2 \wedge \textstyle\bigwedge_{j=1}^{n} \mathtt{rank(o_j)} = 2\\
&\quad \mathtt{i}\Uparrow\mathtt{ba}/\!\downarrow\!\mathtt{ab},\{\mathtt{o_1},...,\mathtt{o_n}\} \wedge \mathtt{rank(i)} = 2 \wedge \textstyle\bigwedge_{j=1}^{n} \mathtt{rank(o_j)} = 2\\
&\quad \mathtt{i}/?\mathtt{ba},\mathtt{o} \wedge \mathtt{rank(i)} = 3 \wedge \mathtt{rank(o)} = 3\\
&\quad \mathtt{i}/?\mathtt{ab},\mathtt{o} \wedge \mathtt{rank(i)} = 4 \wedge \mathtt{rank(o)} = 4\\
&\}
\end{aligned}$$

Particular leaves are specified by the statecharts depicted in Figure 10.3. Note that these statecharts contain two internal actions – $segOccup$ and $segFree$ which model the arrival and departure of a train to/from a particular segment, respectively. Additionally, the lower-level component $DirController$ is assumed to behave as a one-place buffer for incoming direction setting requests provided that only a single particular request can

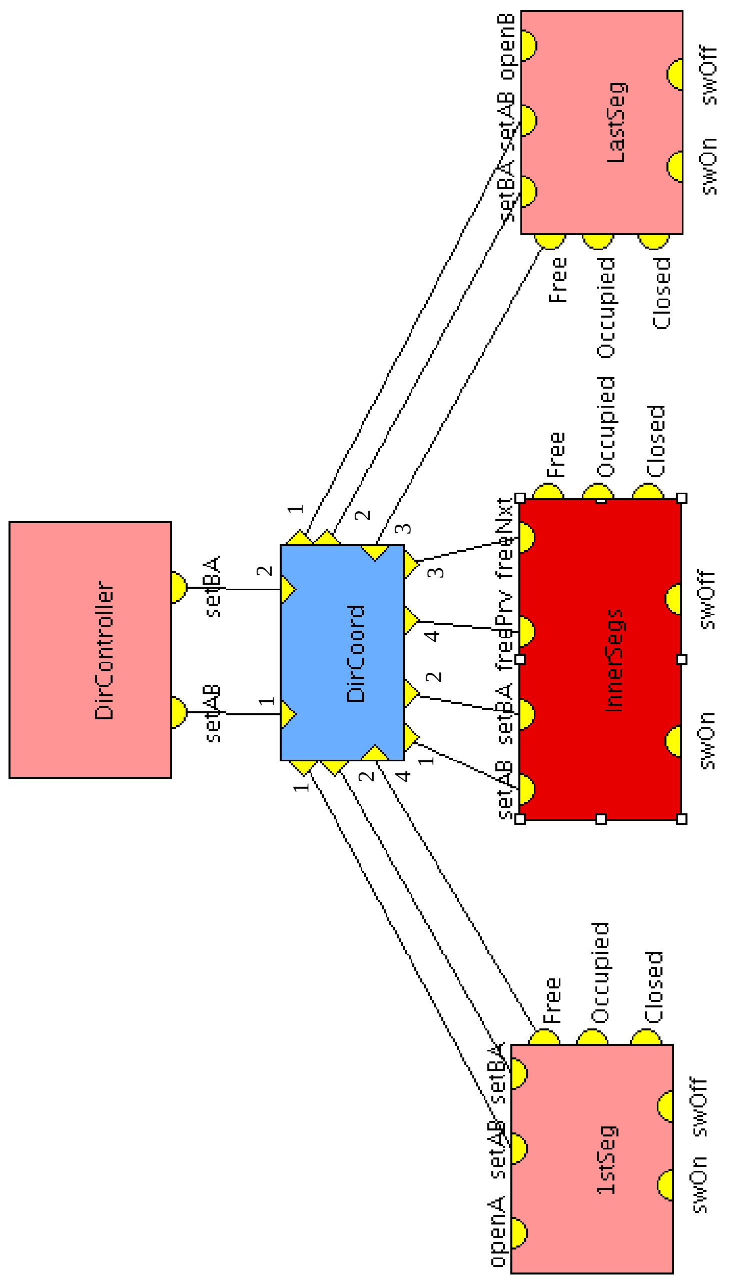


Figure 10.1: Top-most network of the line signalling system

be stored inside the buffer. The top-most *DirController* component is assumed to be modelled as a nondeterministic transmitter of direction setting requests.

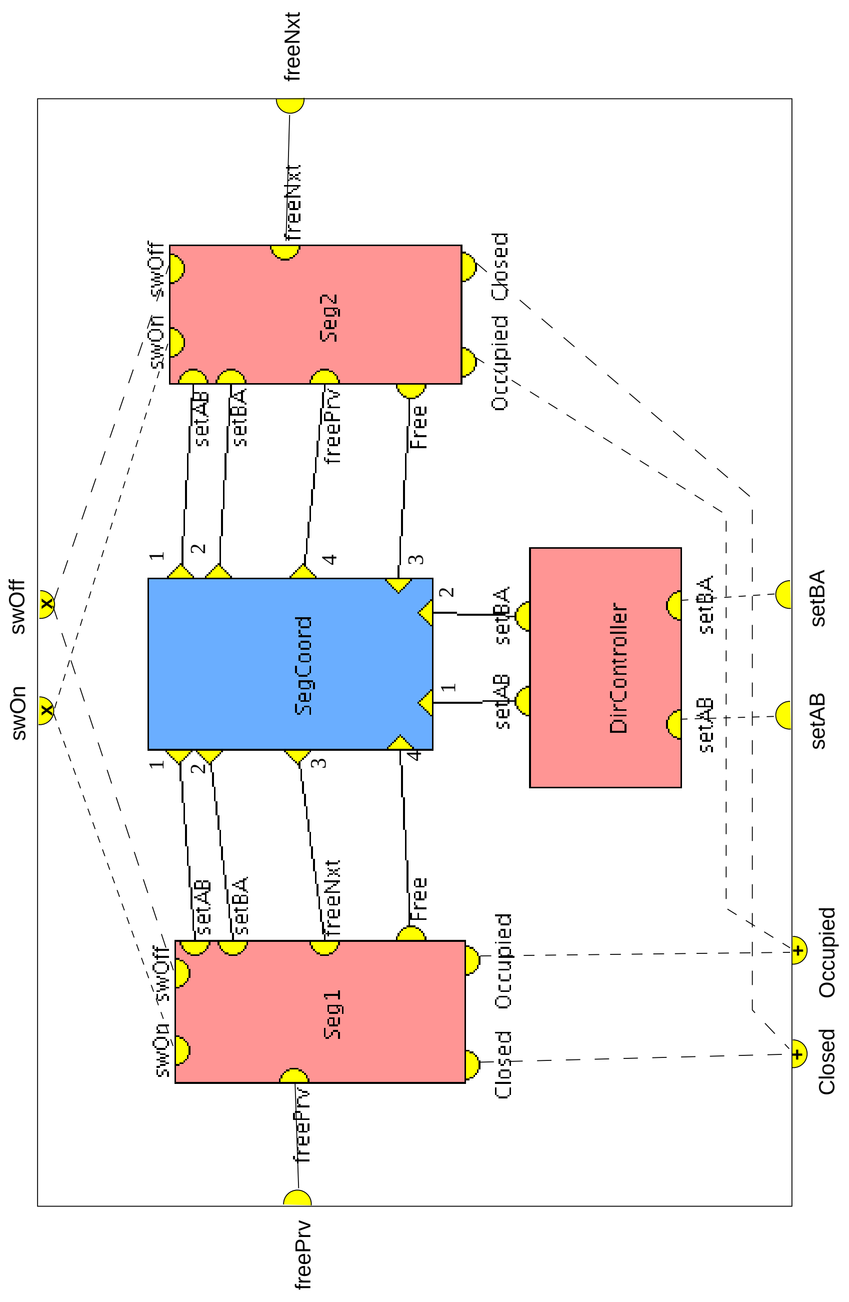


Figure 10.2: Network of inner segments of the line signalling system

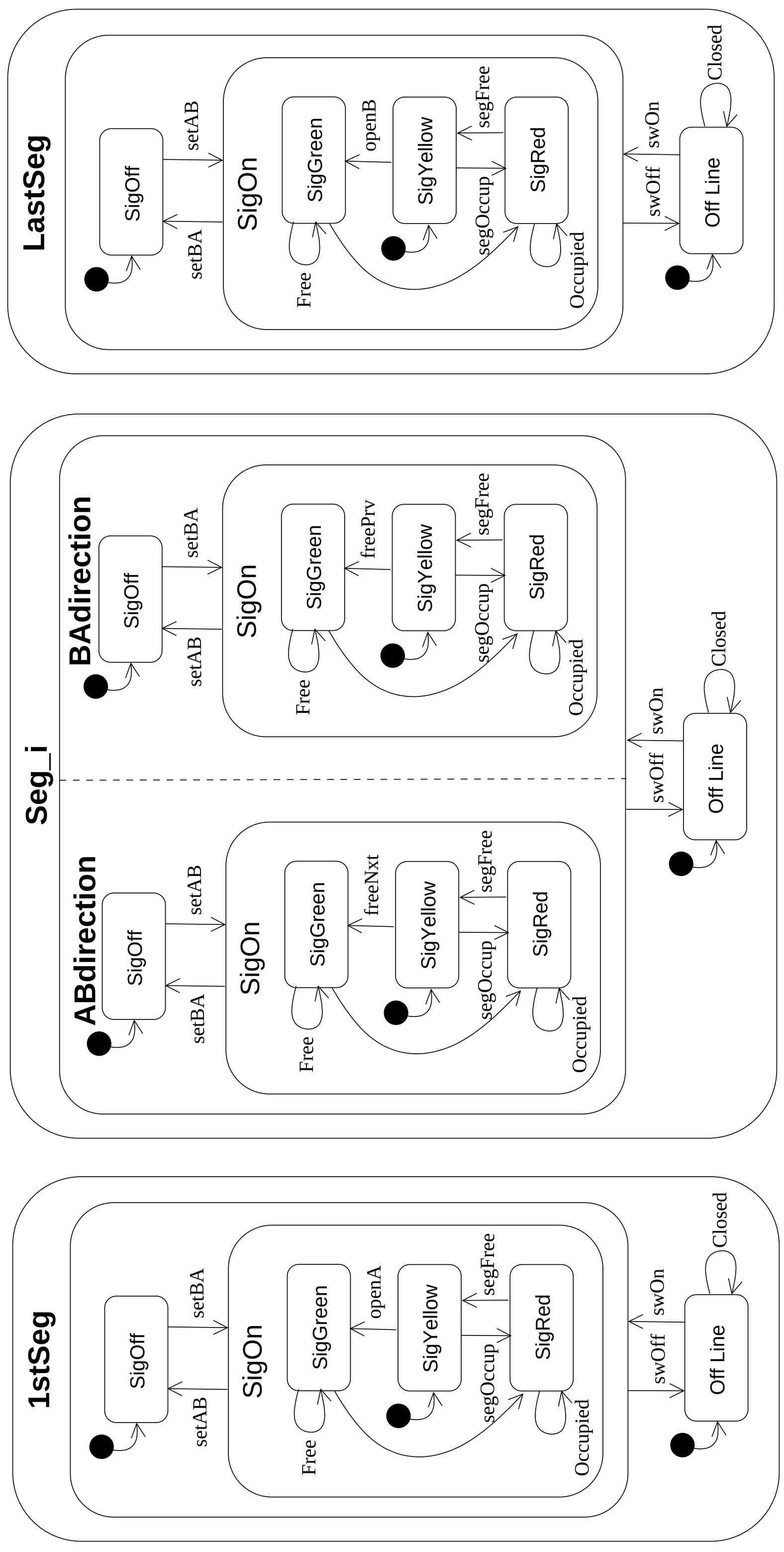


Figure 10.3: Statecharts representing all kinds of line segments

Chapter 11

Conclusion

In this thesis, we have introduced Visual Coordination Diagrams (VCN), a visual formalism for architectural description of reactive and interactive systems. In contrary to common visual notations, the language is founded on a precise formal base in two aspects:

- It has defined a precise mapping from graphical notation to mathematical structures (so-called structural terms).
- The behavioural model of the language is also formal, it is based on the notion of labelled state-transition systems.

With respect to the above mentioned aspects, the language VCN satisfies the needs for formal architectural design of interactive and reactive systems. VCN employs results of two research communities — research on architectural description languages (ADLs) and research on coordination. From the former, it takes the aspects of components/connectors distinction and hierarchal component-based comprehension of system architecture. From the latter, the notion of exogenous coordination is applied. In consequence of both, VCN allows high-level description of component-based topologies of systems, on the one hand. On the other hand, a method for abstract description of connector types employing coordination models of both asynchronous and synchronous kinds is provided.

All the above mentioned aspects of the language VCN enables it for application in architectural description of interactive and reactive systems. As the framework for behavioural consistency of architectures is developed in terms of equivalence checking, VCN supports also the correctness-by-construction feature of modern component-based design.

11.0.5 Future Work

However, there are still many aspect in which the language has to be improved in order to be satisfactorily applied for description of real systems.

First of all, the implementation of the VCNE — the prototype editor of VCN architectures — has to be extended to capture all the aspects of the language defined and discussed throughout the thesis.

The following aspects has to be implemented:

- Value-passing behavioural model.
- Support for all the features of gates, i.e., synchronous and universal types of gates has to be implemented.
- Architectural interoperability checking of cyclic architectures.
- The results of Chapter 6 — implementation of the bus specification language and of the algorithm for bus instance construction.

Implementation of the above mentioned features would enable realisation of exhaustible evaluation of the language, which is necessary for satisfying of the main aim of the language — to be practical used by system developers.

Concerning the theoretical questions, we believe that the general coordination framework defined in Chapter 6 gives a unified fundamental base for comparative study of different coordination models. We plan to extend comparison results achieved in [BJ03] to other models of coordination.

Finally, another future research topic concerning this work is development of an interface-oriented operational model of VCN architectures, as it was described in Section 5.2 of Chapter 5.

Bibliography

[AB05] Alessandro Aldini and Marco Bernardo. On the usability of process algebra: An architectural view. *Theor. Comput. Sci.*, 335(2-3):281–329, 2005.

[AFN03] D. Antoš, O. Fučík, and J. Novotný. Project of IPv6 Router with FPGA Hardware Accelerator. In *Proceeding of 13th International Conference on Field-Programmable Logic and Applications*, volume 2778, pages 964–967. LNCS, Springer-Verlag, 2003.

[AFV01] L. Aceto, W.J. Fokkink, and C. Verhoef. Structural Operational Semantics. In *Handbook of Process Algebra*, pages 197–292. Elsevier, 2001.

[AG97] R. Allen and D. Garlan. A formal basis for architectural connection. *ACM Trans. Softw. Eng. Methodol.*, 6(3):213–249, 1997.

[AP05] J. Adamek and F. Plasil. Component composition errors and update atomicity: static analysis: Research articles. *J. Softw. Maint. Evol.*, 17(5):363–377, 2005.

[APF00] Charles Andre and Marie-Agnes Peraldi-Frati. Behavioral Specification of a Circuit Using SyncCharts: A Case Study. In *Proc. of EUROMICRO'00*. IEEE Computer Society, 2000.

[AR02] F. Arbab and J. Rutten. A coinductive calculus of component connectors, 2002.

[Arb98] F. Arbab. What do you mean, coordination? *Bulletin of the Dutch Association for Theoretical Computer Science (NVTI)*, 1998.

[Arb04] Farhad Arbab. Reo: a channel-based coordination model for component composition. *Mathematical. Structures in Comp. Sci.*, 14(3):329–366, 2004.

[Bal02] D. Balek. *Connectors in Software Architectures*. PhD thesis, Charles University, Prague, 2002.

[BCD02] Marco Bernardo, Paolo Ciancarini, and Lorenzo Donatiello. Architecting families of software systems with process algebras. *ACM Trans. Softw. Eng. Methodol.*, 11(4):386–426, 2002.

[BCJ$^+$04] Dines Bjørner, Peter Chiang, Morten S. T. Jacobsen, Jens Kielsgaard Hansen, Michael P. Madsen, and Martin Penicka. Towards a formal model of cyberrail. In *IFIP Congress Topical Sessions*, pages 657–664, 2004.

[Ber98] Gerard Berry. The Foundations of Esterel. In *Proof, Language and Interaction: Essays in Honour of Robin Milner*. MIT Press, 1998.

[BG92] Gerard Berry and Georges Gonthier. The Esterel Synchronous Programming Language: Design, Semantics, Implementation. *Science of Computer Programming*, 19(2):87, 1992.

[BJ03] Antonio Brogi and Jean-Marie Jacquet. On the expressiveness of coordination via shared dataspaces. *Sci. Comput. Program.*, 46(1-2):71–98, 2003.

[BK98] J. A. Bergstra and P. Klint. The discrete time ToolBus — a software coordination architecture. *Science of Computer Programming*, 31(2–3):205–229, 1998.

[Blo95] B. Bloom. Structural Operational Semantics for Weak Bisimulations. *Theor. Comput. Sci.*, 146(1-2):25–68, 1995.

[BM93] Jean-Pierre Banatre and Daniel Le Métayer. Programming by multiset transformation. *Commun. ACM*, 36(1):98–111, 1993.

[BRS93] G. Berry, S. Ramesh, and R. K. Shyamasundar. Communicating reactive processes. In *Conference Record of the Twentieth Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 85–98, Charleston, South Carolina, 1993.

[CDS00] R. Cleaveland, X. Du, and S. A. Smolka. GCCS: A Graphical Coordination Language for System Specification. In *Proceedings of COORD'00*. LNCS, Springer Verlag, 2000.

[CES06] CESNET, z.s.p.o. *Liberouter project home page*, 2006. `http://www.liberouter.org/`.

[CG89] Nicholas Carriero and David Gelernter. Linda in context. *Commun. ACM*, 32(4):444–458, 1989.

[Cia96] P. Ciancarini. Coordination Models and Languages as Software Integrators. *ACM Computing Surveys*, 28(2):300, 1996.

[Cle96] Paul C. Clements. A survey of architecture description languages. In *IWSSD '96: Proceedings of the 8th International Workshop on Software Specification and Design*, page 16, Washington, DC, USA, 1996. IEEE Computer Society.

[CS96] R. Cleaveland and S. Sims. The NCSU Concurrency Workbench. In *Computer-Aided Verification (CAV '96)*, page 394. LNCS 1102, Springer-Verlag, 1996.

[CS01a] R. Cleaveland and O. Sokolsky. Equivalence and preorder checking for finite-state systems. In *Handbook of Process Algebra*. North-Holland, 2001.

[CS01b] R. Cleaveland and O. Sokolsky. Equivalence and Preorder Checking for Finite-State Systems. In *Handbook of Process Algebra*, pages 391–424. Elsevier, 2001.

[dAH01a] Luca de Alfaro and Thomas A. Henzinger. Interface automata. In *Proceedings of the Ninth Annual Symposium on Foundations of Software Engineering*. ACM Press, 2001.

[dAH01b] Luca de Alfaro and Thomas A. Henzinger. Interface theories for component-based design. In *EMSOFT '01: Proceedings of the First International Workshop on Embedded Software*, pages 148–165, London, UK, 2001. Springer-Verlag.

[dM04] G. de Menten. Graphical Environment for Buchi Automata. Master's thesis, University of Namur (FUNDP), Namur, 2004.

[DS97] I. Lee D.Clarke, H. Ben-Abdallah and O. Sokolsky. PARAGON: A Paradigm for the Specification, Verification, and Testing of Real-Time Systems. In *IEEE Aerospace Conference*, 1997.

[EMCP99] Orna Grumberg Edmund M. Clarke and Doron A. Peled. *Model Checking*. Cambridge : MIT Press, 1999.

[Geh84] N.H. Gehani. Broadcasting Sequential Processes. *Transactions on Software Engineering*, SE-10(4):343–351, 1984.

[Gra99] D. Gray. *Introduction to the Formal Design of Real-Time Systems*. Springer, 1999.

[Har87] D. Harel. Statecharts: A Visual Formalism for Complex Systems. Technical report, The Weizmann Institute, 1987.

[HC01] George T. Heineman and William T. Councill. *Component-Based Software Engineering: Putting the Pieces Together*. Addison Wesley, 2001.

[HKR$^+$04] J. Holecek, T. Kratochvila, V. Rehak, D. Safranek, and P. Simecek. How to Formalize FPGA Hardware Design. Technical Report 4/2004, CESNET z.s.p.o., 2004.

[Hoa85] C.A.R. Hoare. *Communicating Sequential Processes*. Prentice-Hall, 1985.

[KB99] M. Koutny and E. Best. Operational and denotational semantics for the box algebra. *Theoretical Computer Science*, 211, 1999.

[Leu94] S. Leue. *Methods and Semantics for Telecommunications Systems Engineering*. PhD thesis, University of Berne, 1994.

[Mar91] F. Maraninchi. The Argos language: Graphical Representation of Automata and Description of Reactive Systems. In *IEEE Workshop on Visual Languages*, Kobe, Japan, 1991.

[Mic] Microsoft. *Microsoft COM Technology*. http://www.microsoft.com/com.

[Mil83] R. Milner. A Calculus for Synchrony and Asynchrony. *Theoretical Computer Science*, 25, 1983.

[Mil89] R. Milner. *Communication and Concurrency*. Prentice-Hall, 1989.

[Mil99] R. Milner. *Communicating and Mobile Systems: the Pi-Calculus*. Cambridge University Press, 1999.

[MK96] J. Magee and J. Kramer. Dynamic Structure in Software Architectures. *SIGSOFT Softw. Eng. Notes*, 21(6):3–14, 1996.

[OMG] OMG. *CORBA Component Model Specification*. http://www.omg.org/technology/documents/formal/components.htm.

[OMG03] OMG. *Unified Modeling Language. Version 2.0*. OMG, 2003.

[PBJ98] F. Plasil, D. Balek, and R. Janecek. SOFA/DCUP: Architecture for Component Trading and Dynamic Updating. In *Proceedings of the International Conference on Configurable Distributed Systems*, page 43. IEEE Computer Society, 1998.

[Pen06] Martin Penicka. *Towards a Theory of Railways*. PhD thesis, Czech Technical University, 2006.

[Pla05] F. Plasil. Enhancing component specification by behavior description - the sofa experience. In *Proceedings of the 4th International Symposium on Information and Communication Technologies (WISICT 2005)*, pages 185–190. ACM, 2005.

[Pra91] K. V. S. Prasad. A calculus of broadcasting systems. In *TAPSOFT '91: Proceedings of the international joint conference on theory and practice of software development on Colloquium on trees in algebra and programming (CAAP '91): vol 1*, pages 338–358, New York, NY, USA, 1991. Springer-Verlag New York, Inc.

[RC03] A. Ray and R. Cleaveland. Architectural Interaction Diagrams: AIDs for System Modeling. In *Proc. of ICSE 2003*. IEEE, 2003.

[Reh] Z. Rehak. Visual Coordination Diagrams Editor. Bachelor's Thesis, Masaryk University, Brno, 2006.

[Reh06] Z. Rehak. *Visual Coordination Networks Editor (VCNE) project home page*. ParaDiSe Laboratory, Masaryk University Brno, 2006. `http://anna.fi.muni.cz/~xrehak5/vcne/`.

[Saf01] D. Safranek. Graphical Specification of Concurrent Systems (in Czech). Master's thesis, Faculty of Informatics, Masaryk University Brno, 2001.

[Saf02] D. Safranek. SGCCS: A Graphical Language for Real-Time Coordination. In *Proceedings of FOCLASA'02*, volume 68 of *ENTCS*. Elsevier Science, 2002.

[Saf03] D. Safranek. Visual Specification of Concurrent Systems. In *Proceedings of 18th IEEE International Conference on Automated Software Engineering*, pages 369–375. IEEE Computer Society, 2003.

[Saf04] D. Safranek. Visual Specification of Systems with Heterogeneous Coordination Models. In *3st International Workshop on Foundations of Coordination Languages and Software Architectures*, volume 68 of *ENTCS*. Elsevier Science, 2004.

[Saf06] D. Safranek. Architectural Interoperability Checking in Visual Coordination Networks. In *Proceedings of 2nd International Workshop on Methods and Tools for Coordinating Concurrent, Distributed and Mobile Systems*. Elsevier, 2006. to appear in ENTCS.

[SDK$^+$95] M. Shaw, R. DeLine, D. V. Klein, T. L. Ross, D. M. Young, and G. Zelesnik. Abstractions for Software Architecture and Tools to Support Them. *IEEE Trans. Softw. Eng.*, 21(4):314–335, 1995.

[Sif05] Joseph Sifakis. A framework for component-based construction extended abstract. In *SEFM '05: Proceedings of the Third IEEE International Conference on Software Engineering and Formal Methods*, pages 293–300, Washington, DC, USA, 2005. IEEE Computer Society.

[Sim] J. Simsa. Implementation of a Graphical Editor for Specification of Concurrent Systems (in Czech). Bachelor's Thesis, Masaryk University, Brno, 2004.

[SRV$^+$06] A. Smrcka, V. Rehak, T. Vojnar, D. Safranek, P. Matousek, and Z. Rehak. Verifying VHDL Designs with Multiple Clocks in SMV. In *Proceedings of 11th International Workshop on Formal Methods for Industrial Critical Systems*, LNCS, to appear. Springer, 2006.

[SS05] D. Safranek and J. Simsa. VCD: A Visual Formalism for Specification of Heterogeneous Software Architectures. In *Theory and Practice of Computer Science: 31st Conference on Current Trends in Theory and Practice of Computer Science*, volume 3381 of *LNCS*, pages 320–329. Springer, 2005.

[UP97] I. Ulidowski and I. Phillips. Formats of Ordered SOS Rules with Silent Actions. In *TAPSOFT '97: Proceedings of the 7th International Joint Conference CAAP/FASE on Theory and Practice of Software Development*, pages 297–308, London, UK, 1997. Springer-Verlag.

[Vaa93] F. Vaandrager. Expressiveness Results for Process Algebras. Technical report, No. CS-930, CWI, University of Amsterdam, 1993.

[vEVD89] P. H. J. van Eijk, C. A. Vissers, and M. Diaz. *The Formal Description Technique LOTOS*. Elsevier Science Publishers B.V., 1989.

[vEVD90] P. H. J. van Eijk, C. A. Vissers, and M. Diaz. *VHDL Cookbook*. Dept. of Computer Science, University of Adelaide, South Australia, 1990.

[vG95] R. van Glabbeek. On the Expressiveness of ACP. In A. Ponse, C. Verhoef, and S. F. M. van Vlijmen, editors, *Algebra of Communicating Processes: Proc. of the 1st Workshop on the Algebra of Communicating Processes (ACP-94)*, pages 188–217. Springer, Berlin, Heidelberg, 1995.

[vG01] R.J. van Glabbeek. The Linear Time - Branching Time Spectrum I. In *Handbook of Process Algebra*, pages 3–99. Elsevier, 2001.

[Win93] Glynn Winskell. *The formal semantics of programming languages: an introduction*. MIT Press, 1993.