IST IP CASCADAS "Component-ware for Autonomic, Situation-aware Communications, And Dynamically Adaptable Services"

Bringing Autonomic Services to Life

Deliverable D2.1 - Report on Pervasive Supervision

Session 6: Towards a Mathematical Framework for Pervasive Supervision

Status and Version:	Final Version	
Date of issue:	23.12.2006	
Distribution:	Project Internal	
Author(s):	Name	Partner
	Peter H. Deussen	FOKUS
Checked by:		

CASCADAS — BRINGING AUTONOMIC SERVICES TO LIVE

Towards a Mathematical Framework for Pervasive Supervision

Peter H. Deussen

This report is part of the CASCADAS Deliverable 2.1. Since it makes heavily use of mathematical notions, the Word typesetting program is not suitable for this type of work. So we decided to use the LaTeX typesetting system instead and to present this work in a separate document.

The document addresses a mathematical foundation of a number of terms necessary to develop a theory and practical application of model-based supervision.

Contents

CHAPTER 1

Introduction

Supervision is understood as the enhancement of a given system by an additional control structure that continuously observes the state and behavior of the system and takes measures if the system enters a non-suitable state to guide it back into an acceptable mode of operation.

This plain definition contains a number of terms that need to be explained in more detail. Firstly, it talks not about a particular type of system (or even a concrete one) but addresses a general notion of the term system. However, it assumes that a system comprises of states (which can be observed) and of some type of dynamics that changes the system state which can also (at least to a certain degree) be observed but also can be controlled. Finally, it assume that there is an assessment of system state and behavior which allows to tell whether a system state or behavior is suitable or not.

Observation means abstraction. It is impossible to sense a real system in its very details, to monitor all the processes of a physical or—in the present case—of a information technological world. Moreover, abstraction is a necessary precondition for understanding (where the term "understanding" is used to indicate the capability to judge whether a particular process serves a given purpose or not and to estimate the effects of this process). It is not possible to grasp a complex computation process in terms of basic processor instructions. Complex communication means (e.g. communication protocols and protocol stacks) cannot adequately described by the exchange of bits and bytes. The correct interpretation of observations of such processes and their control require of knowledge about the mechanisms that are realized by them, of their semantics. This knowledge turns observations, i.e. the gathering if data, into perceptions, i.e. contextualized information, and—symmetrically—goal oriented activities into sequences of actions.

Contextualization, i.e. the interpretation and generation of data against the background of specific knowledge requires the existence of (at least) a situational model of the surrounding world (or system) explaining the causal relationship of those data, but also an understanding of system internal relations. But from the perspective of causality, all situations and the transitions between them are of equal desirability—having just a causal model of its surrounding puts a system into the situations that it knows what is going on but it doesn't care. Thus it is crucial that it is in capable to assess the suitability of an observed situation. Having a notion of "good" and "bad" provides a motive for a system to move from an undesired situation to a desirable one.

Figure 1 shows the relationship of the external "real" system and the internal system model. It assumes that the real system comprises of a set of *situations* and *processes* that lead from one situation into another. Those processes are not single actions but probably complex composed activities. On the system model site, situations correspond to system states, and processes correspond to system actions. The relationship between model and real system is established by observations and control.

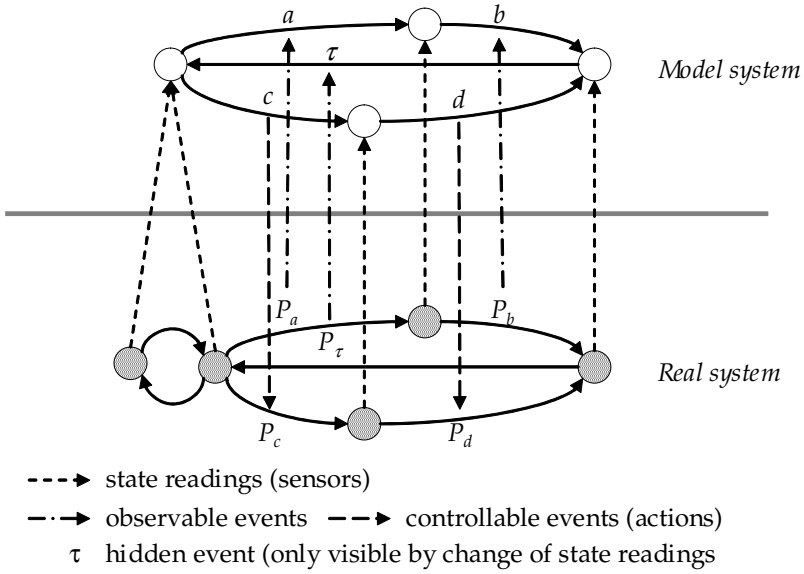


FIGURE 1. Relationship between real system and system model.

— Situations are assumed to be observable (e.g. by sensor readings). Of course, not all aspects of a situation can be observed, thus different situations may not be distinguishable by the observation mechanisms. In the extreme case, the state reading reduced to the simple information that some external state exists.
— Processes may be observable either directly by the perception of system events or indirectly by the change of state readings (we use the "hidden action" τ to express such an indirect perception). There might be even processes which cannot be observed at all, neither by direct perception nor by indirect state reading.
— Finally, some of the processes of the real system might be controllable be the execution of system actions which are described in the system model.

We do in general not assume that the above relationships are actually in place. Instead, we define a number of measures that expresses the suitability of the system model to understand and to control the real system, i.e. the *competence* of the control system with respect to its purpose.

This report is organized into three major parts. Chapter 2 introduces a strict mathematical formalisms of the notions of a "system", "system behavior", and "assessment". At the current stage, we restrict ourselves to the assessment of the suitability of system states. The more generic (and much more complicated) assessment the suitability of system behavior will be addressed in a later project phase. In defining these terms and developing their theoretical foundation, we try to be independent of any concrete formalism to describe system models to give the designer of system models as much freedom in the choice of that formalism as possible.

Chapter 3 continues with the development of the framework with the definition of notions for system abstraction and system composition. Following the "no syntax" approach to stay free from concrete formalisms, system composition is explained without referencing a

concrete composition or communication mechanism. Abstraction and composition are then used to defined hierarchical models.

Chapter 4 then discusses supervision algorithms based on the notion of system models as given before. A basic supervision algorithm to be used with "flat", i.e. non-hierarchical system models is discussed in detail, and metrics for system competence are discussed for this algorithm. Finally, an extension of this supervision algorithm to hierarchical system models is briefly discussed.

This report is on work in progress. So a number of questions is left open. We conclude with a summary and an outlook on further work in Chapter 5

It remains to explain the relationship to the overall work programme of the CASCADAS Work Package 2 on Pervasive Supervision. Given a system and a supervision system, we suppose that this system employs the supervision system as a supplementary service. The contract on supervision basically consists of a supervision model where the system under supervision announces the information it is willing to disclose to the supervision system, its own assessment of states, and the ways in which the supervision system is allowed to control the system under supervision if such a control is necessary. For hierarchical supervision of composite system (ACEs or self-organized ensembles of ACEs), the supervision model (the contract) is composed from the local supervision models of the system components. A precise justification of these ideas, in particular the relationship between different levels of abstraction of within a hierarchical supervision model, is the main motivation of the work presented in this report.

CHAPTER 2

Mathematical Framework

In this chapter, we introduce an approach towards a model based framework for hierarchical supervision. After presenting in Section 1 the mathematical notations we use, we start our course of elaboration in Section 2 with distributed alphabets, which provide a basic notion of actions a that a distributed system can perform. We continue with the general notion of a system that is used throughout this report, which is given by a set of states and a transition relation (labeled with actions from a distributed alphabet) that respects the distributed nature of those actions.

For sequential systems (i.e. systems which run on a single computer), the automata theoretic operational semantics given by sequences of actions that are executable by such a system has been proven as appropriate. For distributed systems however, more general approaches have been suggested. One possible candidate are so-called partial ordered multisets, which are basically sets of actions equipped with a notion of causality. In particular it takes into account that there is no causal relationship between actions executed by system components running on different locations. This type of system semantics is discussed in Section 3.

For supervision, not only the causal structure of a system is needed, but also an assessment of the "suitability" (or "desirability") of system states and system behaviors, as the supervision system are supposed to stay passive when the system under supervision is in a suitable state. In Section 4, we restrict our attention to the assessment of system states. Assessments of system behaviors will be considered in a later phase of the CASCADAS project.

1. Background Notations

For the sake of breviety we adopt the convention that whenever a structure $A = \langle X, Y, \ldots \rangle$ is introduced in a definition, then its components are denoted by X_A, Y_A, ... Thus for instance, in Def. 2.7 we introduce so-called pomsets as structures $v = \langle E, <, \lambda \rangle$. In the further course of this document, the first component of each pomset w will then denoted by E_w, the second one as $<_w$, and the third one as λ_w.

Since we make extensive use of relations throughout this report, we use some non-standard notations: Let $R \subseteq A \times B$ be a binary relation. Then for some $a \in A$ we let

$$R(a) \quad =_{\mathrm{df}} \quad \{b \in B \mid a \; R \; b\}\,.$$

For some set $C \subseteq A$, we extend this notion to

$$R(C) \quad =_{\mathrm{df}} \quad \bigcup_{a \in C} R(a).$$

The *inverse* of R is the relation $R^{-1} \subseteq B \times A$ and is defined by:

$$b \; R^{-1} \; a \quad \Leftrightarrow_{\mathrm{df}} \quad a \; R \; b.$$

If $R \subseteq A \times B$ and $S \subseteq B \times C$, then the composition $R \cdot S \subseteq A \times C$ of R and S is defined to be

$$a\ (R \cdot S)\ c \quad \Leftrightarrow_{\mathrm{df}} \quad c \in S(R(a)).$$

The identity relation $\mathbf{id}_A \subseteq A \times A$ is defined as

$$a\ \mathbf{id}_A\ b \quad \Leftrightarrow_{\mathrm{df}} \quad a = b.$$

If $R \subseteq A \times A$ is a relation, we denote by R^+ the smallest transitive relation that contains R (i.e. $R \cdot R \subseteq R^*$), and R^* is $R^+ \cup \mathbf{id}_A$.

We in general do not distinguish (apart from the notational level) between binary relations and functions, i.e. we associate a (partial or total) function f from A to B by its graph, i.e. the relation

$$G_f \quad = \quad \{\langle a, b\rangle \subseteq A \times B \mid f(a) = b\}$$

Adopting this point of view allows us to apply set operations also to functions, i.e. we use expressions like $f \cup g$. We however use usual notations like $f(a) = b$ instead of $b \in f(a)$ or $a\ f\ b$. By $f : A \rightharpoonup B$ we denote the fact that f is a partial function from A to B (i.e. it might be the case that for some $a \in A$ there is no $b \in B$ such that $\langle a, b\rangle \in f$), while as usual $f : A \to B$ indicates that f is a total function. For $f : A \rightharpoonup B$ and $g : B \rightharpoonup C$ we define $g \circ f : A \rightharpoonup C$ as usual to be

$$(g \circ f)(a) \quad =_{\mathrm{df}} \quad g(f(a))$$

(i. e. $g \circ f = f \cdot g$). Functional restriction is denoted by $f \upharpoonright C$ for $f : A \rightharpoonup B$ and $C \subseteq A$ and is defined as

$$f \upharpoonright C \quad =_{\mathrm{df}} \quad f \cap (C \times B).$$

It is useful to have a disjoint sum: For sets A and B define

$$A + B \quad =_{\mathrm{df}} \quad A \times \{0\} \cup B \times \{1\}.$$

An *equivalence* relation $\equiv \subseteq A \times A$ over some set A is a reflexive, symmetric, and transitive relation. By

$$[a]_{\equiv} \quad =_{\mathrm{df}} \quad \{b \in A \mid a \equiv b\}$$

we denote the *equivalence class* of some $a \in A$ w.r.t. $\equiv$.

$$A/_{\equiv} \quad =_{\mathrm{df}} \quad \{[a]_{\equiv} \mid a \in A\}$$

is the *quotient set* of A w.r.t. $\equiv$.

If $f : A \to B$ is a surjective mapping, then the relation $\equiv_f \subseteq A \times A$ defined by

$$a \equiv_f b \quad \Leftrightarrow_{\mathrm{df}} \quad f(a) = f(b)$$

is obviously an equivalence relation. We then abbreviate $[a]_{\equiv_f}$ by $[a]_f$ and $A/_{\equiv_f}$ by $A/_f$.

2. Distributed Alphabets and Their Interpretations

A *distributed alphabet* is given by a set of system actions together with a notion of independence, namely a symmetric and irreflexive relation. Intuitively, two actions are independent if they can be executed in parallel or distributed at different locations (e.g. computers). The complement of the independence relation is called dependence relation. Thus, dependent actions are those which are in direct causal relationship, e.g. because they have to be executed in sequence, on the same processor, or in the same thread. We furthermore consider a hidden actions which is traditionally denoted by the Greek letter τ. Usually, τ is used to refer to "internal" system actions that cannot be observed from outside of the system but which are nevertheless need to be taken into account in order to get meaningful definitions of notions such as deadlocks and livelocks. In this report, τ is used in a slightly different meaning, namely to refer to actions that are executed by the environment of a system that are not explicitly defined in the system model: τ means that in a given situation (i.e. system state), the environment might do "something". Thus τ is used to model "external non-determinism".

It should be noted that all occurrences of τ are considered to be dependent (i.e. there is no parallel execution of several instances of the hidden action). This choice in mathematical modeling has been made to indicate the fact that the knowledge whether actions can be performed independently already bears structural information on the distribution of system components. A system which in embedded in a "black" environment perceives it as a generator of sequences of events (messages, sensed data) without a way of judging whether the processes that generate these event sequences are executed in parallel or not.

2.1. DEFINITION. Let $\Sigma = \langle A, \tau, D \rangle$ be a structure comprising an alphabet A, a special symbol $\tau \in A$, and a symmetric and reflexive relation $D \subseteq A \times A$. Then Σ is called a *distributed alphabet*. We use the letter I to denote the complement of D, i. e. $I =_{\mathrm{df}} (A \times A) \setminus D$. The relation I is called an *independence relation*. □

The following definition described the term "system" that is used in this report in a very abstract way, namely as a structure that comprises states—although it makes no assumption on the nature of system states, their structure, etc.—and state transitions which are labeled by actions of a (distributed) alphabet. For real applications, system models are not supposed to be given in this ascetic form. To the best knowledge of the author a state/transition semantics can be assigned to *all* modeling formalisms that are candidates for describing supervision models. This includes finite state machines, extended finite state machines, all Petri-net like formalisms, behavioral UML diagrams (as long a formal semantics is available), process algebras, timed state machines (as timing can be approximated by non-determinism), etc. This generic notion of a system however will restricted in the following by a few necessary assumptions.

2.2. DEFINITION (Transition System). A *transition system* over a distributed alphabet Σ is a structure $\left\langle S, \left\{ \xrightarrow{a} \right\}_{a \in A_\Sigma} \right\rangle$, where S is a set of *states* and $\xrightarrow{a} \subseteq S \times S$ assigns a state transformation relation to each symbol $a \in A_\Sigma$. □

We now establish the connection between the notion of independence given by the relation I_Σ and the transition relation of transition systems in an "extensional" way by explaining the necessary effects (or better, non-effects) of the firing of independent system actions.

To give some intuition, let us use the "intensional" picture of a global system state composed from the local states of a set of distributed components (in the most easiest form, such a global state is a vector of local states). Actions are supposed to act on the local states of the executing components, hence, independent system actions act on pairwise disjoint sets of local system states (note that a communication action may act on the local states of all components that participate in the communication). Thus if we consider two independent actions a and b, then (1) the execution of a has no impact to the executability of b and vice versa. (2), the execution order of a and b has no impact to the global system state that is reached after their execution.

2.3. DEFINITION (Diamond properties). Let Σ be a distributed alphabet with associated independence relation I_Σ and let $\mathcal{I}$ be a transition system over Σ. Let $a, b \in A_\Sigma$ be actions with $a\ I_\Sigma\ b$, and let $s_1, s_2, s_3 \in S_\mathcal{I}$ be states. The following conditions on transition systems $\mathcal{I}$ are usually called *diamond properties*:

(1) $s_1 \xrightarrow[\mathcal{I}]{a} s_2 \wedge s_1 \xrightarrow[\mathcal{I}]{b} s_3 \Rightarrow (\exists s_4 \in S) \left[s_2 \xrightarrow[\mathcal{I}]{b} s_4 \wedge s_3 \xrightarrow[\mathcal{I}]{a} s_4 \right]$;

(2) $s_1 \xrightarrow[\mathcal{I}]{a} s_2 \wedge s_2 \xrightarrow[\mathcal{I}]{b} s_3 \Rightarrow (\exists s_4 \in S) \left[s_1 \xrightarrow[\mathcal{I}]{b} s_4 \wedge s_4 \xrightarrow[\mathcal{I}]{a} s_3 \right]$.

We say that $\mathcal{I}$ *enjoys the diamond properties* if the conditions explained above are true for all pairs of actions from Σ and all states of $\mathcal{I}$. □

A second restriction of the class of transition systems we consider throughout this report is the assumption of determinism in the sense that the execution of a specific system actions leads to a uniquely defined result. Non-deterministic choice is still expressible by means of a set of actions enabled at some state, these behavioral alternatives however can be distinguished. An exception is the hidden action. For τ, bounded as well as unbounded non-determinism is permitted.

2.4. DEFINITION (Deterministic Transition System). A transition system $\mathcal{I}$ over Σ is called *deterministic* if for each $a \in A_\Sigma \setminus \{\tau_\Sigma\}$ and for all states $s, s_1, s_2 \in S_\mathcal{I}$ we have

$$s \xrightarrow[\mathcal{I}]{a} s_1 \wedge s \xrightarrow[\mathcal{I}]{a} s_2 \quad \Rightarrow \quad s_1 = s_2$$

□

The third assumption is on the specific behavior of the hidden action τ.

2.5. DEFINITION (Hidden Action). Let $\mathcal{I}$ be a transition system over Σ. Then τ_Σ is called *hidden* in $\mathcal{I}$ if

$$\begin{aligned} \xrightarrow[\mathcal{I}]{\tau_\Sigma} \circ \xrightarrow[\mathcal{I}]{\tau_\Sigma} \quad &\subseteq \quad \xrightarrow[\mathcal{I}]{\tau_\Sigma}, \text{ and} \\ \mathbf{id}_{S_\mathcal{I}} \quad &\subseteq \quad \xrightarrow[\mathcal{I}]{\tau}. \end{aligned}$$

is true. □

The first equation in the above definition states that several occurrences of τ_Σ cannot be distinguished from a single one, or—equivalently—the relation $\xrightarrow{\tau_\Sigma}$ is transitive. If we consider for instance a sensor that reports periodically certain aspects of the state of a system or its environment, and if we assume that we do not have an explicit model of the processes

that are responsible for changes of the system state, then we use the τ action to model the dynamics of these processes in a very abstract way. Thus, an alternative interpretation of the transitivity of the τ action is that the set of states that are visible depends on the sampling rates of its sensors. Furthermore, we assume that τ is always enabled (but does not necessarily have an impact to visible system states), i.e. there is always some (probably hidden) system activity of the environment.

Putting these three assumptions together, we obtain the definition of a "well-formed" system, or that of an *interpretation of a distributed alphabet*.

2.6. DEFINITION (Interpretation). An *interpretation* of a distributed alphabet Σ is a transition system $\mathcal{I}$ such that the following properties do hold:

(1) $\mathcal{I}$ is deterministic.
(2) $\mathcal{I}$ enjoys the diamond properties.
(3) τ_Σ is hidden in $\mathcal{I}$.

By $\mathbf{Int}(\Sigma)$ we denote the class of interpretations of Σ. □

3. Partial Ordered Multisets

If we consider sequential processes, sequences of actions are a suitable semantic structure to describe their behavior. If we however look on parallel or distributed systems, the facts that two actions are executed in parallel (e.g. by distributed system components) cannot adequately expressed if only sequences are considered. We will not contribute to the long ongoing discussion whether actions sequences are *really* not suitable as semantics for distributed systems (a comprehensive and amusing discussion of this topic can be found in [29]), instead we will introduce a representative of a so-called "partial order semantics" that essentially behaves like sequences in the sense that its algebraic structure, namely that of a free monoid, belongs to the same class a the set of sequences over a given alphabet.

Partially ordered multisets (or *pomsets*, for short) have been introduced by Grabowski [16] and further developed by Starke [34] and Pratt [28]. The basic idea is to model system behavior by an ordered set of events. Events stand for the occurrence (or execution) of system actions. Thus events are labeled with system actions. The ordering expresses causality. Thus an event e_1 preceeds an event e_2 if e_1 has to be executed in order to enable e_2. On the other hand, events which are causally unrelated can be executed in any order.

Mathematically, causality is modeled by a partial order, i.e. a relation with the following properties:

— It is *anti-symmetric*, because if e_1 causes e_2 then e_2 cannot be the cause of e_1,
— it is *transitive*, because if e_1 causes e_2 and e_2 causes e_3, then e_1 is also an indirect cause of e_3.

In this report, we are mainly interested in the determination and execution of supervision activities, i.e. of finite "plans". Thus we restrict ourself to finite pomsets.

The material presented in this section bases on previous works of the author. A more detailed presentation including proofs for all lemmas in this section (with the exception of the pretty obvious Lemma 2.11) can be found in [11].

2.7. DEFINITION. Let Σ be a distributed alphabet. Let E be a finite set of event, $< \subseteq E \times E$ be a partial order, and $\lambda : E \to A_\Sigma$ a labeling function such that

$$\lambda(e_1)\ D_\Sigma\ \lambda(e_2) \quad \Rightarrow \quad e_1 < e_2 \vee e_2 < e_1$$

is true, then the structure $v = \langle E, <, \lambda \rangle$ is called a *partially ordered multiset* (a *pomset*, for short) over Σ. By $\mathbf{Pom}(\Sigma)$ we denote the class of pomsets over the distributed alphabet Σ. □

At this point a clarification on notations have to be made. Symbols like $<$, $\sqsubset$, etc. are in general used to denote the irreflexive versions of partial orders, while $\leqslant$, $\sqsubseteq$, etc. are used to refer to their reflexive counterparts.

The above definition establishes also the connection to distributed alphabets by stipulating the assumption that dependent system actions cannot be executed in parallel, i.e. have to be causally ordered.

2.8. REMARK. A notion of isomorphism between pomsets can be defined as follows: Let u and v be pomsets over the same distributed alphabet. Then u and v said to be *isomorphic* if there is a bijective mapping $h : E_u \to E_v$ such that $e_1 <_u e_2 \Leftrightarrow h(e_1) <_v h(e_2)$ and $\lambda_u = \lambda_v \circ h$ do hold; with other words, h is a renaming of events which preserving ordering. The original sources then define a pomset as an isomorphism classes of structures of the form $\langle E, <, \lambda \rangle$ rather than as these structures themselves. In this report, we adopt the somewhat imprecise convention to associate those structures with their equivalence classes, as well-formedness issues never occur. For a more stringent setting see [11]. □

We continue with a number of useful definitions and abbreviations that allow to deal with pomsets in a more convenient way.

2.9. DEFINITION. Let v be a pomset over a distributed alphabet Σ.

(1) Two events $e_1, e_2 \in E_v$ are called *concurrent* is v if they are not related by $\leqslant_v$:
$$e_1\ \mathrm{co}_v\ e_2 \Leftrightarrow_{\mathrm{df}} \neg(e_1 \leqslant_v e_2 \vee e_2 \leqslant_v e_1).$$
(2) On the other hand, events which are not concurrent are *in line*:
$$e_1\ \mathrm{li}_v\ e_2 \Leftrightarrow_{\mathrm{df}} \neg(e_1\ \mathrm{co}_v\ e_2).$$
(3) A *co-set* of v is some set $C \subseteq E_v$ such that $\mathrm{li}_v \cap (C \times C) = \varnothing$. A *cut* of v is a co-set of v which is maximal w. r. t. set inclusion.
(4) A set $C \subseteq E_v$ is called *pre-closed* in v if $C = {\leqslant_v}^{-1}(C)$.
(5) If $C \subseteq E_v$ is a set of events, then $v[C]$ denotes the pomset $\langle C, <_v \cap (C \times C), \lambda_v \restriction C \rangle$.
(6) A *configuration* of v is a pomset w such that $w = v[C]$ for some pre-closed set C of v. By $\mathbf{Cnf}(v)$ we denote the set of configurations of v.
(7) For some set $C \subseteq E_v$ we use the notation $v\lceil C \rceil$ to denote the configuration $v\left[\leqslant_v^{-1}(C)\right]$. A configuration of the form $v\lceil\{e\}\rceil$ is called *local*. For local configurations we drop the parenthesis and write simply $v\lceil e \rceil$. Another configuration which is frequently used is $v\lfloor e \rfloor =_{\mathrm{df}} v\left[<_v^{-1}(e)\right]$.
(8) A pomset w is called a *sequentializing* of v if $E_w = E_v$, $<_v \subseteq <_w$, and $\lambda_w = \lambda_v$. By $\mathbf{Seq}(v)$ we denote the set of sequentializings of v.
(9) For convenience we define another two partial orders on pomsets:
$$v \leqslant w \quad \Leftrightarrow_{\mathrm{df}} \quad w \in \mathbf{Cnf}(v), \text{ and} \qquad v \preccurlyeq w \quad \Leftrightarrow_{\mathrm{df}} \quad w \in \mathbf{Seq}(v).$$
(10) The *empty pomset* is denoted by ϵ and is defined to be $\langle \varnothing, \varnothing, \varnothing \rangle$.
(11) If $a \in A$, then the *letter* $\langle \{0\}, \varnothing, 0 \mapsto a \rangle$ is also denoted by a.

(12) Finally, to simplify notions, if $e \in E_v$ is an event, we write $\tilde{e}$ instead of $\lambda_v(e)$ whenever it is clear from the context which labeling function λ_v is meant.

□

The next definition explains how pomsets can be executed in interpretations of distributed alphabets.

2.10. DEFINITION. Let $\mathcal{I}$ be a transition system over a distributed alphabet Σ. For each pomset $v \in \mathbf{Pom}(\Sigma)$ we define a relation $\xrightarrow[\mathcal{I}]{v} \subseteq S_{\mathcal{I}} \times S_{\mathcal{I}}$ inductively as the smallest relations which satisfies:

$$\xrightarrow[\mathcal{I}]{\epsilon} \;=\; \mathbf{id}_{S_{\mathcal{I}}},$$

$$\xrightarrow[\mathcal{I}]{v} \;=\; \xrightarrow[\mathcal{I}]{v\lceil E_v \setminus \{e\}\rceil} \cdot \xrightarrow[\mathcal{I}]{\tilde{e}} \text{ for some } e \in \max_{<}(E_v).$$

For some state $s \in S_{\mathcal{I}}$ we denote by

$$\mathbf{Pom}_{\mathcal{I}}(s) \;=_{\mathrm{df}}\; \left\{ u \in \mathbf{Pom}(\Sigma) \mid (\exists s' \in S_{\mathcal{I}}) s \xrightarrow[\mathcal{I}]{u} s' \right\}.$$

the *pomset language* of $\mathcal{I}$ at the state s. □

Clearly, the outcome of the execution of a pomset v depends on the order in which maximum events e are selected, thus it is not a partial function but a relation even for deterministic transition systems $\mathcal{I}$. The following lemma shows that if $\mathcal{I}$ is an interpretation of the underlying distributed alphabet and the hidden action τ_Σ is "locally deterministic" in the sense that for each state that is reached during the execution of a pomset the firing of τ_Σ has a unique outcome, then the transition relation for pomsets is a partial function. The lemma follows directly from the diamond properties (c. f. Def. 2.3).

2.11. LEMMA. *For each pomset $v \in \mathbf{Pom}(\Sigma)$ over a distributed alphabet Σ and each state $s \in S_{\mathcal{I}}$ of a well-formed interpretation $\mathcal{I}$ the following does hold: For all events $e \in E_v$ labeled with $\tilde{e} = \tau_\Sigma$ assume that*

$$\left| \xrightarrow[\mathcal{I}]{v\lfloor e \rfloor}(s) \right| = 1 \quad \Rightarrow \quad \left| \xrightarrow[\mathcal{I}]{v\lceil e \rceil}(s) \right| \leqslant 1.$$

Then $\left| \xrightarrow[\mathcal{I}]{v}(s) \right| \leqslant 1.$

We now introduce two operations on the class of pomsets over a distributed alphabet, namely *weakening* and *weak sequential composition*. By Definition 2.7 we found that if e_1 and e_2 are events of a pomset $v \in \mathbf{Pom}(\Sigma)$ such that $e_1 \;\mathrm{co}_v\; e_2$ is true, then their labels have to be independent, i. e. $\tilde{e}_1 \; I_\Sigma \; \tilde{e}_2$ has to be true. The opposite is not required, events with independent labels need not to be concurrent. This is correct as the dependence relation D_Σ is not transitive and thus does not express indirect dependencies. But now we find that if $v \in \mathbf{Pom}(\Sigma)$, then all its sequentializations are also, i.e. $\mathbf{Seq}(v) \subseteq \mathbf{Pom}(\Sigma)$. The question arises whether there is a pomset u which comprises only the those dependencies which are really necessary. Such a pomset v is a minimal element with respect to the sequentialization relation $\preccurlyeq$ in the class $\mathbf{Pom}(\Sigma)$.

The following definitions explains how such a "weakest" pomset can be constructed from a pomset $v \in \mathbf{Pom}(\Sigma)$.

2.12. DEFINITION (Weakening). Let Σ be a distributed alphabet. The operation $\langle\cdot\rangle_\Sigma : \mathbf{Pom}(\Sigma) \to \mathbf{Pom}(\Sigma)$ is define as

$$\begin{aligned} \langle v\rangle_\Sigma &=_{\mathrm{df}} \left\langle E_v, <', \lambda_v\right\rangle, \\ \text{where } e_1 <' e_2 &\Leftrightarrow_{\mathrm{df}} e_1 <_v e_2 \wedge \tilde{e}_1\ D_\Sigma\ \tilde{e}_2 : \end{aligned}$$

$\langle v\rangle_\Sigma$ is called the *weakening* of v. We put $\mathbf{Pom}^{\mathrm{w}}(\Sigma) =_{\mathrm{df}} \{\langle v\rangle_\Sigma \mid v \in \mathbf{Pom}(\Sigma)\}$. □

2.13. LEMMA. $\mathbf{Pom}^{\mathrm{w}}(\Sigma) = \min_{\preccurlyeq} \mathbf{Pom}(\Sigma)$, *i.e.* $\mathbf{Pom}(\Sigma) = \bigcup\{\mathbf{Seq}(u) \mid u \in \mathbf{Pom}^{\mathrm{w}}(\Sigma)\}$.

Once having the notion of *weakening* of a pomset, we may ask if there is a composition operation for pomsets that introduces only the necessary dependencies and leaving the composite pomset as concurrent as possible. The following definition introduces the *weak sequential composition* operation for pomsets. Readers familiar with *Message Sequence Charts* (MSCs) [20] may note the similarity to the weak sequential composition operation for MSCs.

2.14. DEFINITION (Weak sequential composition). Let Σ be a distributed alphabet. The operation $\circ_\Sigma : \mathbf{Pom}(\Sigma) \times \mathbf{Pom}(\Sigma) \to \mathbf{Pom}(\Sigma)$ is defined as

$$\begin{aligned} v \circ_\Sigma w &=_{\mathrm{df}} \left\langle E_v \cup E_w, (<_v \cup <_w \cup R)^*, \lambda_v \cup \lambda_w\right\rangle \\ \text{where } e_1\ R\ e_2 &\Leftrightarrow_{\mathrm{df}} e_1 \in E_v \wedge e_2 \in E_w \wedge \tilde{e}_1\ D_\Sigma\ \tilde{e}_2, \end{aligned}$$

and moreover, $E_v \cap E_w = \varnothing$ is assumed. The pomset $u \circ_\Sigma v$ is called the *weak sequential composition* of u and v. □

The following lemma explains the relationship of the operations $\langle\cdot\rangle_\Sigma$ and $\circ_\Sigma$.

2.15. LEMMA.

(1) *Both* $\langle\cdot\rangle_\Sigma$ *and* $\circ_\Sigma$ *are well-defined operations.*
(2) $v \in \mathbf{Seq}(u) \Rightarrow \langle v\rangle_\Sigma = \langle u\rangle_\Sigma$.
(3) $\langle\langle v\rangle_\Sigma\rangle_\Sigma = \langle v\rangle_\Sigma$.
(4) $\langle u \circ_\Sigma v\rangle_\Sigma = \langle u\rangle_\Sigma \circ_\Sigma \langle v\rangle_\Sigma$.

The structure $\langle A^*, \cdot, \epsilon\rangle$ is generally used as a standard example for a for a monoid, where A^* is the set of all finite sequences of symbols from an alphabet A including the empty sequence ϵ, and $\cdot$ denotes the concatenation operation for sequences, i.e. $v \cdot w = vw$. It is also an example of a "free monoid", i.e. the set A^* can be completely characterized in terms of the underlying alphabet A, ϵ, and concatenation. These pleasant algebraic properties make the mathematical work with sequences very easy and appropriate as semantic structure to express system dynamics, in opposite to partial order semantics which seems to be much more complicated to deal with. The following lemma however shows that this is not really true. It states that the algebraic structure of weakened pomsets together with the empty pomset and weak sequential composition as concatenation operations forms also a free monoid.

2.16. LEMMA. *The structure* $\langle \mathbf{Pom}^{\mathrm{w}}(\Sigma), \circ_\Sigma, \epsilon\rangle$ *is a free monoid over the alphabet* A *the distributed alphabet* $\Sigma = \langle A, \tau, D\rangle$. *More precisely, the following properties are satisfied for* $u, v, w \in \mathbf{Pom}^{\mathrm{w}}(\Sigma)$*:*

(1) $\epsilon \circ_\Sigma v = v \circ_\Sigma \epsilon = v$;
(2) $u \circ_\Sigma (v \circ_\Sigma w) = (u \circ_\Sigma v) \circ_\Sigma w$.

Moreover, $\mathbf{Pom}^{\mathrm{w}}(\Sigma)$ *is the smallest set of pomsets satisfying*

(1′) $\epsilon \in \mathbf{Pom}^{\mathrm{w}}(\Sigma)$;

(2′) $a \in A_\Sigma \Rightarrow a \in \mathbf{Pom}^{\mathrm{w}}(\Sigma)$ *(where the first occurrence of a denotes a symbol and the second one refers to the associated letter, c.f. Def. 2.9.(11));*
(3′) $u, v \in \mathbf{Pom}^{\mathrm{w}}(\Sigma) \Rightarrow u \circ_\Sigma v \in \mathbf{Pom}^{\mathrm{w}}(\Sigma)$.

The final lemma in this section states that the $\circ_\Sigma$ operation is compatible with the firing relation for pomsets in interpretations of distributed alphabets Σ and thus provides the last element of the justification of the usefulness of weakened pomsets as partial order semantics for distributed systems.

2.17. LEMMA. *Let $\mathcal{I} \in \mathbf{Int}(\Sigma)$ for some distributed alphabet Σ, let $u \in \mathbf{Pom}(\Sigma)$, and let $a \in A_\Sigma$. Then*

$$\xrightarrow[\mathcal{I}]{u \circ_\Sigma a} \quad = \quad \xrightarrow[\mathcal{I}]{u} \cdot \xrightarrow[\mathcal{I}]{a}$$

does hold.

4. Valued Interpretation

In this section, we are going to extend the notion of "systems" used in this report by a further element, namely that of values. The main motivation for the introduction of a mechanism to assess the suitability of states is given by the usage of transitions systems as a model structure for supervision. The supervision algorithms presented in Chapter 4 base on the notion of an *admissible* system state. If admissibility is violated, supervision activities are triggered. The explicit representation of those triggers in the model structure enables for a fully generic supervision approach that uses models as contracts.

We start with the definition of value structures. The most obvious choice for such a structure would be that of a finite linear order of values, for instance "*bad* $\sqsubset$ *acceptable* $\sqsubset$ *good* $\sqsubset$ *excellent*", or "*error* $\sqsubset$ *initial* $\sqsubset$ *intermediate* $\sqsubset$ *final*". For concrete models, appropriate value structures will in almost all cases readily at hand, as the designer of a system model are supposed to have a pretty good idea on the semantics and hence on how to assess the suitability of the states of system she is working on.

We however foresee the need to have in some cases more expressive value structure than finite total orderings. If we deal with systems and system states comprising a number of quantitative parameters over infinite domains, a (finite) total order might be difficult to define, while in general a partial order still might be available.

2.18. DEFINITION (Value Structure). A *value structure* $\langle X, \sqsubseteq \rangle$ comprises of

(1) X, a set of *values*,
(2) $\sqsubseteq \subseteq X \times X$, a partial order on values expressing the assumption that if $x \sqsubseteq y$, then y is "better" or "more preferred" than x.

□

For the handling of values, it is necessary to recapitulate the notions of least upper bounds and greatest lower bounds in partial orders.

2.19. DEFINITION. Let $\langle X, \sqsubseteq \rangle$ be a value structure and let $Y \subseteq X$. Then lower bound of Y is some $x \in X$ such that $x \sqsubseteq y$ for all $y \in Y$ does hold. The element x is called a greatest lower bound of Y if for all lower bounds z of Y we have $z \sqsubseteq x$. The greatest lower bound Y is denoted by $\bigsqcap Y$ (provided such an elements do exist in X). A value structure is called *complete* if for each subset $Y \subseteq X$ a greatest lower bound $\bigsqcap Y \in X$ exists. □

Complete value structures are of course nothing than *lower complete lattices* which are fundamental in the theory of partial orders and lattices. A number of so-called "completion operations" are available to extend an arbitrary partial order to a (lower) complete lattice. We will not go into the details of these constructions here. The standard references for lattice theory remain the classical works of Birkhoff [6] and Grätzer [17]. Another excellent source is the text book of Dillworth and Crawley [12].

2.20. DEFINITION. A *valued interpretation* over a distributed alphabet Σ is an interpretation $\mathcal{I}$ of Σ together with a complete value structure $\langle X, \sqsubseteq \rangle$, a mapping $\xi : S_{\mathcal{I}} \to X$ called *state assessment function*, and an *admissible predicate* $\varphi : S_{\mathcal{I}} \to \mathbb{B}$. We furthermore assume

$$(\dagger) \qquad \bigvee_{s \in S} \varphi(s) \wedge \bigsqcap_{s \in S} \xi(s) \sqsubseteq \bigsqcap_{s \in S'} \xi(s) \quad \Rightarrow \quad \bigvee_{s \in S'} \varphi(s).$$

for all subsets $S, S' \subseteq S_{\mathcal{I}}$.

The class of valued interpretations over Σ is denoted by $\mathbf{ValInt}(\Sigma)$. If $\mathcal{I} \in \mathbf{ValInt}(\Sigma)$, then the additional component of $\mathcal{I}$ are denoted by $X_{\mathcal{I}}$, $\sqsubseteq_{\mathcal{I}}$, $\xi_{\mathcal{I}}$, and $\varphi_{\mathcal{I}}$, respectively. Least upper bound and greatest lower bound operations are denoted as $\bigsqcup_{\mathcal{I}}$ and $\bigsqcap_{\mathcal{I}}$, respectively. For further convenience, if $\mathcal{I} \in \mathbf{ValInt}(\Sigma)$, we denote by $\mathbf{Int}(\mathcal{I})$ the "interpretation part" of $\mathcal{I}$, i. e.

$$\mathbf{Int}(\mathcal{I}) \quad =_{\mathrm{df}} \quad \left\langle S_{\mathcal{I}}, \left\{ \xrightarrow[\mathcal{I}]{a} \right\}_{a \in A_\Sigma} \right\rangle.$$

□

The condition (†) requires some explanation. The predicate $\varphi_{\mathcal{I}}(s)$ assesses the admissibility of a state $s \in S_{\mathcal{I}}$, and since values from the set X are used to express the the desirability of a system state, it makes sense to assume

$$\varphi(s) \wedge \xi(s) \sqsubseteq \xi(s') \quad \Rightarrow \quad \varphi(s').$$

(Note that the above condition follows from (†) by putting $S = \{s\}$ and $S' = \{s'\}$.) In the following, we frequently will work with assessments of sets of states that share a common property (thus for instance, that of being the concrete image of a more abstract state). To work with those sets, we use the $\bigsqcap$ operation to assess its collective desirability. To keep the threshold defined by the predicate φ consistent with those collective assessments, the stronger condition is needed.

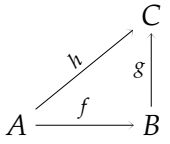
CHAPTER 3

Morphisms

In this chapter, we are going to define the notions of abstraction in Section 1 and embedding in Section 2 firstly on the level of distributed alphabets, secondly on the level of interpretations of distributed alphabets, and thirdly on value structures of interpretations. Of further importance in the notion of the image of a subsystem under an embedding operation which basically describes what "is left" from this subsystem when it is embedded into some environment. Images are used to establish the connection between abstractions and embeddings.

Zooms (and anti-zooms) are presented in Section 3. Zooms provides us with an operation for the local refinement of a system by means of an embedding of an abstract version of subsystem into some environment, and an abstraction that related a more concrete version of this subsystem to the abstract one. Zooming then means basically the replacement of the abstract version of the subsystem in its environment by the concrete one.

The concepts introduced below are presented in a Category Theoretical fashion. Deep knowledge of Category Theory is however not necessary apart from the a basic understanding of so-called commutating diagrams. A commutating diagram comprises of a set of *objects* (think of sets, algebraic structures, or in particular interpretations of distributed alphabets), and a set of *arrows* or *morphisms* between objects (e.g. functions, homomorphisms, or, in the present case, abstractions and embeddings). If A, B, and C are objects, and f, g, and h are arrows, then the diagram



is said to be *commutative* if $\xrightarrow{f} \circ \xrightarrow{g} = \xrightarrow{h}$, where $\circ$ is a *concatenation* (or *composition*) operation on arrows (e.g. the usual composition operation for set function). More complex diagrams commute if each involved triangle of the above form commutes.

The classical reference for Category Theory is the book by Saunders Mac Lane [19]. Another excellent introduction can be found in [15].

1. Abstraction

Figure 1 shows the general idea of the abstraction operation. Abstraction is modeled by means of three mapping, namely α, which assigns an abstract actions to concrete ones, β, which related concrete and abstract states, and γ, which is responsible of translating concrete state assessments into their abstractions. The following three definitions are concerned with these mappings.

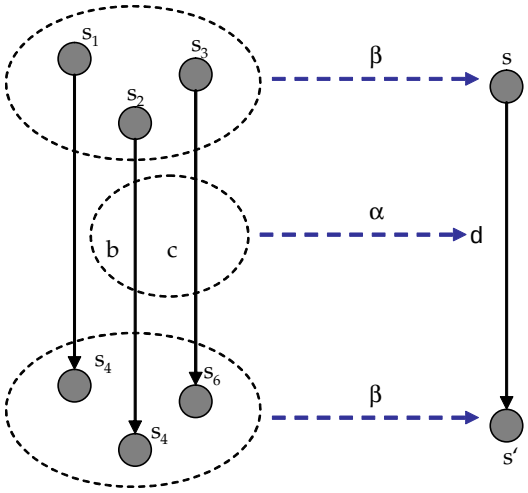


FIGURE 1. Illustration of the concept of abstraction. Several states of the more concrete system may mapped into one abstract state, as well as several concrete actions may have a single abstract counterpart. The abstraction mapping α for actions may not be total.

Considering system actions, abstraction means that several actions of the more concrete system might not be distinguishable on a more abstract level (and thus mapped into one abstract action). It might moreover be the case that some concrete actions do not have an abstract counterpart at all. Thus α is defined as a partial mapping. It is however not possible to "invent" information during the process of abstraction. Thus in particular, the concrete version of the hidden action τ has to be mapped into its abstract counterpart.

We already discussed that the independence relation on system actions provides knowledge about the distribution of system components. Thus we stipulate the assumption that if information on action independence exists on the abstract level, then it has to be available already on the concrete level.

3.1. DEFINITION (Abstraction of actions). Let Σ_1 and Σ_2 be distributed alphabets. An *abstraction map* is a partial mapping $\alpha : A_{\Sigma_1} \rightharpoonup A_{\Sigma_2}$ such that

$$\begin{aligned} \alpha(a)\ I_{\Sigma_2}\ \alpha(b) &\Rightarrow a\ I_{\Sigma_1}\ b, \text{ and} \\ \alpha(\tau_{\Sigma_1}) &= \tau_{\Sigma_2}. \end{aligned}$$

We write **abs** $\langle\alpha\rangle : \Sigma_1 \to \Sigma_2$ if α is an abstraction map. □

Now consider interpretations. We already discussed that we assume a mechanism to obtain information on system states (e.g. by means of sensors). Since this mechanisms is always in place (although it might deliver only void information), we assume that the abstraction mapping for system states β is a total mapping. We moreover require that a concrete system action can be executed at some state if its abstract image (if it exists) can be executed at the abstract image of this state, or expressed in the opposite way, it is possible that an abstract action is enabled at an abstract state even if none of the concrete versions of that actions are enabled at the set of concretizations of this state. This means, that the process of abstraction may be forgetful on the enabledness conditions of system actions. Finally, a concrete action

a that is not visible at the abstract level (i.e. $\alpha(e)$ is not defined) must not lead to any visible state change. Note that the abstract action τ is still available to model the fact that an action is not directly visible at the abstract level but is only perceived by a state transition.

3.2. DEFINITION (Abstraction of interpretations). Let $\mathcal{I}_1 \in \mathbf{Int}(\Sigma_1)$ and $\mathcal{I}_2 \in \mathbf{Int}(\Sigma_1)$ be interpretations of the distributed alphabets Σ_1 and Σ_2, respectively. Then a pair of mapping β, α is called an *abstraction map* if $\mathbf{abs}\ \langle \alpha \rangle : \Sigma_1 \to \Sigma_2$ is an abstraction and moreover, $\beta : S_1 \to S_2$ is a total mapping such that

$$s_1 \xrightarrow[\mathcal{I}_1]{a} s_2 \quad \Rightarrow \quad \alpha(a) \text{ is defined} \wedge \beta(s_1) \xrightarrow[\mathcal{I}_2]{\alpha(a)} \beta(s_2) \vee \alpha(a) \text{ is not defined} \wedge \beta(s_1) = \beta(s_2)$$

We write $\mathbf{abs}\ \langle \alpha, \beta \rangle : \mathcal{I}_1 \to \mathcal{I}_2$ in this case. □

3.3. DEFINITION (Abstraction of valued interpretations). Let $\mathcal{I}_1 \in \mathbf{ValInt}(\Sigma_1)$ and $\mathcal{I}_2 \in \mathbf{ValInt}(\Sigma_2)$ be valued interpretations of the respective distributed alphabets, and $\mathbf{abs}\ \langle \alpha, \beta \rangle : \mathbf{Int}(\mathcal{I}) \to \mathbf{Int}(\mathcal{I}_2)$ an abstraction acting on the "interpretation parts" of $\mathcal{I}_1$ and $\mathcal{I}_2$. Let $\gamma : X_{\mathcal{I}_1} \to X_{\mathcal{I}_2}$ be a surjective mapping such that the following conditions hold true:

$$\gamma\left(\prod_{\mathcal{I}_1}(\xi_{\mathcal{I}_1}(S))\right) \quad = \quad \prod_{\mathcal{I}_2}(\xi_{\mathcal{I}_2}(\beta(S)))\,;$$

for all $S \subseteq S_{\mathcal{I}_1}$, and moreover,

$$\varphi_{\mathcal{I}_2}(s) \quad \Rightarrow \quad \bigvee_{s' \in [s]_\beta} \varphi_{\mathcal{I}_1}(s')$$

Abstractions for valued interpretations are denoted by $\mathbf{abs}\ \langle \alpha, \beta, \gamma \rangle : \mathcal{I}_1 \to \mathcal{I}_2$. □

Thus the abstraction arrow

$$\mathcal{I}_1 \xrightarrow{\mathbf{abs}\langle \alpha, \beta, \gamma \rangle} \mathcal{I}_2$$

unfolds to three parallel arrows:

$$\begin{array}{ccc} \Sigma_1 & S_{\mathcal{I}_1} & X_{\mathcal{I}_1} \\ \downarrow{\scriptstyle \alpha} & \downarrow{\scriptstyle \beta} & \downarrow{\scriptstyle \gamma} \\ \Sigma_2 & S_{\mathcal{I}_2} & X_{\mathcal{I}_2} \end{array}$$

It is moreover easy to check that the "continuity assumption"

$$\gamma\left(\prod_{\mathcal{I}_1}(\xi_{\mathcal{I}_1}(S))\right) = \prod_{\mathcal{I}_2}(\xi_{\mathcal{I}_2}(\beta(S)))$$

implies "monotonicity", i.e.

$$\xi_{\mathcal{I}_1}(s_1) \sqsubseteq_{\mathcal{I}_1} \xi_{\mathcal{I}_1}(s_2) \quad \Rightarrow \quad \xi_{\mathcal{I}_2}(\beta(s_1)) \sqsubseteq_{\mathcal{I}_2} \xi_{\mathcal{I}_2}(\beta(s_2))$$

To see this, note that $\prod_{\mathcal{I}_1}\{x\} = x$, and if $x \sqsubseteq_{\mathcal{I}_1} y$, then $\prod_{\mathcal{I}_1}\{x, y\} = x$; and of course, the same is true for the value structure of $\mathcal{I}_2$. Thus the chain $\{x, y\}$ is mapped to $\{\gamma(x), \gamma(y)\}$, its greatest lower bound to $\gamma(x)$. Together, this implies $\gamma(x) \sqsubseteq_{\mathcal{I}_2} \gamma(y)$. It can moreover be shown that both properties (continuity and monotonicity) are equivalent for finite value structures.

3.4. DEFINITION (Reduced abstractions). An abstraction $\mathbf{abs}\,\langle\alpha,\beta\rangle : \mathscr{I}_1 \to \mathscr{I}_2$ is called *reduced* if β is surjective. If $\mathbf{abs}\,\langle\alpha,\beta\rangle$ is reduced, then abstractions $\mathbf{abs}\,\langle\alpha,\beta,\gamma\rangle$ are also called *reduced*. □

In general we assume throughout this report that abstractions are reduced. This assumption is reasonable if we consider sets of abstractions of a real system (e.g. a hierarchy of models ordered by their degree of abstraction), but not if we consider the relationship between the real system and the collection of its models. In general we do not have a way to determine whether the abstraction of a potential system state has a correspondence in the real system. In the following we are however more concerned with the relationship of models, in particular with refinement and abstraction operations. We thus consider all abstractions that occur in this report as being reduced, if not stated differently.

The following lemma states that there is an identity abstraction, as well a composition operation on abstractions. Thus, interpretations together with abstraction arrows form a category.

3.5. LEMMA.

(1) *Let* $\mathscr{I} \in \mathbf{ValInt}(\Sigma)$. *Then* $\mathbf{abs}\,\langle \mathbf{id}_{A_\Sigma}, \mathbf{id}_{S_\mathscr{I}}, \mathbf{id}_{X_\mathscr{I}}\rangle : \mathscr{I} \to \mathscr{I}$ *is an abstraction.*
(2) *Let* $\mathscr{I}_i \in \mathbf{ValInt}(\Sigma_i)$ *for* $i = 1,2,3$. *If* $\mathbf{abs}\,\langle\alpha_1,\beta_1,\gamma_1\rangle : \mathscr{I}_1 \to \mathscr{I}_2$ *and* $\mathbf{abs}\,\langle\alpha_2,\beta_2,\gamma_2\rangle : \mathscr{I}_2 \to \mathscr{I}_3$ *are abstractions, then also* $\mathbf{abs}\,\langle\alpha_2\circ\alpha_1, \beta_2\circ\beta_1, \gamma_2\circ\gamma_1\rangle : \mathscr{I}_1 \to \mathscr{I}_3$.

We now are going to investigate the relationship between abstractions and pomset languages of interpretations. For that, we firstly have to explain how a mapping on action sets of distributed alphabets can be extended to pomsets over these alphabets.

3.6. DEFINITION. Let Σ_1, Σ_2 be distributed alphabets such that $\alpha : A_{\Sigma_1} \to A_{\Sigma_2}$ is a mapping. Then α is inductively extended to a mapping $\alpha^* : \mathbf{Pom}(\Sigma_1) \to \mathbf{Pom}(\Sigma_2)$ as follows:

$$\begin{array}{rcl} \alpha^*(\epsilon) & =_{\mathrm{df}} & \epsilon; \\ \alpha^*(u \circ_{\Sigma_1} a) & =_{\mathrm{df}} & \left\{ \begin{array}{rl} \alpha^*(u) \circ_{\Sigma_2} \alpha(a), & \text{if } \alpha(a) \text{ is defined;} \\ \alpha^*(u), & \text{otherwise;} \end{array} \right. \end{array}$$

for all $u \in \mathbf{Pom}(\Sigma_2)$ and $a \in A_{\Sigma_2}$. □

We continue with some sanity checking:

3.7. LEMMA. *Let* Σ_1, Σ_2 *be distributed alphabets such that* $\mathbf{abs}\,\langle\alpha\rangle : \Sigma_1 \to \Sigma_2$ *is an abstraction. Then* α^* *is a well-defined mapping. Furthermore, if* $u \in \mathbf{Pom}^{\mathrm{w}}(\Sigma_1)$, *then* $\alpha^*(u) \in \mathbf{Pom}^{\mathrm{w}}(\Sigma_2)$.

The following lemma justifies the relationship of the notions of refinement and abstraction on distributed alphabets and interpretations by means of the pomset languages of interpretations.

3.8. LEMMA. *Assume* $\mathbf{abs}\,\langle\alpha,\beta\rangle : \mathscr{I}_1 \to \mathscr{I}_2$ *is an abstraction for* $\mathscr{I}_1 \in \mathbf{Int}(\Sigma_1)$ *and* $\mathscr{I}_2 \in \mathbf{Int}(\Sigma_2)$ *and let* $s_1, s_2 \in S_{\mathscr{I}_1}$. *Then*

$$s_1 \xrightarrow[\mathscr{I}_1]{u} s_2 \quad \Rightarrow \quad \beta(s_1) \xrightarrow[\mathscr{I}_2]{\alpha^*(u)} \beta(s_2)$$

for each $u \in \mathbf{Pom}^{\mathrm{w}}(\Sigma_2)$.

PROOF. The proof is carried out by induction. First assume $u = \epsilon$. Then

$$s \xrightarrow[\mathscr{I}_1]{\epsilon} s \quad \Rightarrow \quad \beta(s) \xrightarrow[\mathscr{I}_2]{\epsilon} \beta(s).$$

For induction assume

$$s_1 \xrightarrow[\mathscr{I}_1]{u} s_2 \quad \Rightarrow \quad \beta(s_1) \xrightarrow[\mathscr{I}_2]{\alpha^*(u)} \beta(s_2),$$

and further suppose

$$s_1 \xrightarrow[\mathscr{I}_1]{u \circ_{\Sigma_1} a} s_2.$$

for some $a \in A_{\Sigma_1}$. Assume $\alpha(a)$ is not defined. Then

$$\beta(s_1) \xrightarrow[\mathscr{I}_2]{\alpha^*(u)} \beta(s_2)$$

which is true by induction hypothesis. If, on the other hand, $\alpha(a)$ is defined, then we compute:

$$\begin{aligned}
s_1 \xrightarrow[\mathscr{I}_1]{u \circ_{\Sigma_1} a} s_2 \quad &\Rightarrow \quad s_1 \xrightarrow[\mathscr{I}_1]{u} \cdot \xrightarrow[\mathscr{I}_1]{a} s_2 && \text{(Lemma 2.17)} \\
&\Rightarrow \quad \beta(s_1) \xrightarrow[\mathscr{I}_2]{\alpha^*(u)} \cdot \xrightarrow[\mathscr{I}_2]{\alpha(a)} \beta(s_2) && \text{(Def.'s 3.2, 3.6)} \\
&\Rightarrow \quad \beta(s_1) \xrightarrow[\mathscr{I}_2]{\alpha^*(u) \circ_{\Sigma_2} \alpha(a)} \beta(s_2) && \text{(Lemma 2.17)} \\
&\Rightarrow \quad \beta(s_1) \xrightarrow[\mathscr{I}_2]{\alpha^*(u \circ_{\Sigma_1} a)} \beta(s_2) && \text{(Def. 3.6)}
\end{aligned}$$

which concludes the proof. □

2. Embedding and Image

The embedding operation provides us with a generic notion on how to understand a system component as part of a larger system, or the relationship of a system with its environment. Note that the composition of a number of system components can be understood as the embedding of each of these components into the composed system. Therefore, embeddings give us also a generic notion of composition without assuming any specific composition operation such as synchronous or asynchronous communication.

As in the case of abstractions, the following three definitions concentrate on each of the three aspects of a valued interpretation of a distributed alphabet, namely the underlying alphabet, the transition system, and the value structure.

3.9. DEFINITION (Embeddings of distributed alphabets). Let Σ_1 and Σ_2 be distributed alphabets and let $f : A_{\Sigma_1} \to A_{\Sigma_2}$ be an injective mapping such that

$$\begin{aligned}
f(a)\ I_{\Sigma_2}\ f(b) \quad &\Rightarrow \quad a\ I_{\Sigma_1}\ b \text{ and} \\
f(\tau_{\Sigma_1}) \quad &= \quad \tau_{\Sigma_2}
\end{aligned}$$

for all $a, b \in A_{\Sigma_1}$. Then f is called an *embedding* of Σ_1 into Σ_2. We write $\mathbf{emb}\ \langle f \rangle : \Sigma_1 \to \Sigma_2$ in this case. □

Hence, an embedding of a distributed alphabet into another one preserves the hidden action of the embedded alphabet. Independence in the embedding alphabet has to be also

present in the embedded one, but since a system environment may create new dependencies, the reverse implication is not necessarily true.

3.10. DEFINITION (Embeddings of interpretations). Let $\mathscr{I}_1 \in \mathbf{Int}(\Sigma_1)$ and $\mathscr{I}_2 \in \mathbf{Int}(\Sigma_2)$ be interpretations of the distributed alphabets Σ_1 and Σ_2, respectively. Assume $\mathbf{emb}\,\langle f\rangle : \Sigma_1 \to \Sigma_2$ is an embedding. Let $g : S_{\mathscr{I}_2} \to S_{\mathscr{I}_1}$ be surjective mapping. Then the pair f, g is called an embedding of $\mathscr{I}_1$ into $\mathscr{I}_2$ if there are states $s_1, s_2 \in S_{\mathscr{I}_1}$ such that

$$s_1 \xrightarrow[\mathscr{I}_2]{f(a)} s_2 \quad \Rightarrow \quad g(s_1) \xrightarrow[\mathscr{I}_1]{a} g(s_2)$$

We write $\mathbf{emb}\,\langle f, g\rangle : \mathscr{I}_1 \to \mathscr{I}_2$ in this case. □

Embedding of a system into another one means that the states of the embedded system $\mathscr{I}_1$ contribute to the states of the embedding system $\mathscr{I}_2$ (think of these global states as vectors of the states of the embedded components). Thus we use a mapping that extracts the information specific to the states of the system $\mathscr{I}_1$ from the states of the system $\mathscr{I}_2$.

The embedding of the value structure of an interpretation is defined similar to the abstraction of value structures. The basic idea is that the embedding system may have a richer (i.e. more fine grained) selection of assessments than the embedded one. Moreover, if a state of the embedded system is not admissible then this non-admissibility manifests at all the images of this state in the embedding system.

3.11. DEFINITION (Embeddings of valued interpretations). Let $\mathscr{I}_1 \in \mathbf{ValInt}(\Sigma_1)$ and $\mathscr{I}_2 \in \mathbf{ValInt}(\Sigma_2)$ be valued interpretations of the respective distributed alphabets, and $\mathbf{emb}\,\langle f, g\rangle : \mathbf{Int}(\mathscr{I}) \to \mathbf{Int}(\mathscr{I}_2)$ an embedding acting on the "interpretation parts" of $\mathscr{I}_1$ and $\mathscr{I}_2$. Let $h : X_{\mathscr{I}_2} \to X_{\mathscr{I}_1}$ be a surjective mapping such that the following conditions hold true:

$$h\left(\prod\nolimits_{\mathscr{I}_2}(\xi_{\mathscr{I}_2}(S))\right) \;=\; \prod\nolimits_{\mathscr{I}_1}(\xi_{\mathscr{I}_1}(g(S)))\,;$$

for all $S \subseteq S_{\mathscr{I}_2}$, and moreover,

$$\neg\varphi_{\mathscr{I}_1}(s) \quad \Rightarrow \quad \bigvee_{s' \in [s]_g} \neg\varphi_{\mathscr{I}_2}(s')$$

Embeddings for valued interpretations are denoted as $\mathbf{emb}\,\langle f, g, h\rangle : \mathscr{I}_1 \to \mathscr{I}_2$. □

Thus the embedding

$$\mathscr{I}_1 \xrightarrow{\mathbf{emb}\langle f,g,h\rangle} \mathscr{I}_2$$

unfolds to two parallel arrows and an opposite arrow, i.e.

$$\begin{array}{ccc} \Sigma_1 & S_{\mathscr{I}_1} & X_{\mathscr{I}_1} \\ \Big\downarrow{\scriptstyle f} & \Big\uparrow{\scriptstyle g} & \Big\uparrow{\scriptstyle h} \\ \Sigma_2 & S_{\mathscr{I}_2} & X_{\mathscr{I}_2} \end{array}$$

Again, we have to establish the fact that the class of valued interpretations over distributed alphabets and the class of embedding arrows between these interpretations form a category.

3.12. LEMMA.

(1) *Let* $\mathscr{I} \in \mathbf{ValInt}(\Sigma)$. *Then* $\mathbf{emb}\,\langle \mathbf{id}_{A_\Sigma}, \mathbf{id}_{S_\mathscr{I}}, \mathbf{id}_{X_\mathscr{I}}\rangle : \mathscr{I} \to \mathscr{I}$ *is an embedding.*
(2) *Let* $\mathscr{I}_i \in \mathbf{ValInt}(\Sigma_i)$ *for* $i = 1,2,3$. *If* $\mathbf{emb}\,\langle f_1, g_1, h_1\rangle : \mathscr{I}_1 \to \mathscr{I}_2$ *and* $\mathbf{emb}\,\langle f_2, f_2, h_2\rangle : \mathscr{I}_2 \to \mathscr{I}_3$ *are embeddings, then also* $\mathbf{emb}\,\langle f_2 \circ f_1, g_1 \circ g_2, h_1 \circ h_2\rangle : \mathscr{I}_1 \to \mathscr{I}_3$.

To understand the relationship between abstraction and embedding, it is useful to compute the image of an interpretation under an embedding map:

3.13. DEFINITION (Image). Let $\mathscr{I}_1 \in \mathbf{ValInt}(\Sigma_1)$ and $\mathscr{I}_2 \in \mathbf{ValInt}(\Sigma_2)$ be interpretations of the distributed alphabets Σ_1 and Σ_2, respectively. Assume $\mathbf{emb}\,\langle f, g, h\rangle : \mathscr{I}_1 \to \mathscr{I}_2$ is an embedding. Then the *image* of $\mathscr{I}_1$ under $\mathbf{emb}\,\langle f, g, h\rangle$ is a transition system $\mathbf{img}\,\langle f, g, h\rangle\,(\mathscr{I}_1)$ over a distributed alphabet Σ^f with the following components:

(1) $A_{\Sigma^f} =_{\mathrm{df}} A_{\Sigma_1}$, $a\ I_{\Sigma^f}\ b \Leftrightarrow_{\mathrm{df}} f(a)\ I_{\Sigma_2}\ f(b)$, and $\tau_{\Sigma^f} =_{\mathrm{df}} \tau_{\Sigma_1}$;
(2) $S_{\mathbf{img}\langle f,g,h\rangle(\mathscr{I}_1)} =_{\mathrm{df}} S_{\mathscr{I}_2}/_g$;
(3) $[s_1]_g \xrightarrow[\mathbf{img}\langle f,g,h\rangle(\mathscr{I}_1)]{a} [s_2]_g \Leftrightarrow_{\mathrm{df}} (\exists s_1' \in [s_1]_g)(\exists s_2' \in [s_2]_g) s_1' \xrightarrow[\mathscr{I}_2]{f(a)} s_2'$.
(4) $X_{\mathbf{img}\langle f,g,h\rangle(\mathscr{I}_1)} =_{\mathrm{df}} X_{\mathscr{I}_2}$, with
   (i) $x \sqsubseteq_{\mathbf{img}\langle f,g,h\rangle(\mathscr{I}_1)} y \Leftrightarrow_{\mathrm{df}} x \sqsubseteq_{\mathscr{I}_2} y$,
   (ii) $\varphi_{\mathbf{img}\langle f,g,h\rangle(\mathscr{I}_1)}([s]_g) \Leftrightarrow_{\mathrm{df}} \bigvee_{s' \in [s]_g} \varphi_{\mathscr{I}_2}([s])$,
   (iii) $\xi_{\mathbf{img}\langle f,g,h\rangle(\mathscr{I}_1)}([s_1]_g) =_{\mathrm{df}} \bigsqcap_{\mathscr{I}_2} \xi_{\mathscr{I}_2}([s_1]_g)$.

□

For consistency, the following lemma is needed:

3.14. LEMMA. *Let* $\mathscr{I}_1 \in \mathbf{ValInt}(\Sigma_1)$ *and* $\mathscr{I}_2 \in \mathbf{ValInt}(\Sigma_2)$ *be interpretations of the distributed alphabets* Σ_1 *and* Σ_2, *respectively. Assume* $\mathbf{emb}\,\langle f, g, h\rangle : \mathscr{I}_1 \to \mathscr{I}_2$ *is an embedding. Then* $\mathbf{img}\,\langle f, g, h\rangle\,(\mathscr{I}_1) \in \mathbf{ValInt}(\Sigma_2)$.

PROOF. To see that $\mathbf{img}\,\langle f, g, h\rangle\,(\mathscr{I})$ enjoys the diamond properties let $[s_1]_g, [s_2]_g, [s_3]_g \in S_{\mathbf{img}\langle f,g,h\rangle(\mathscr{I})}$ and let $a, b \in A_{\Sigma^f}$ such that $a\ I_{\Sigma^f}\ b$ is true.

(1) To establish $\mathbf{img}\,\langle f, g, h\rangle\,(\mathscr{I}) \in \mathbf{Int}(\Sigma^f)$, we only show that Def. 2.3.(2) is satisfied, Def.'s 2.3.(1), 2.4, and 2.5 are proved in a similar way. We compute

$$
\begin{aligned}
& [s_1]_g \xrightarrow[\mathbf{img}\langle f,g,h\rangle(\mathscr{I})]{a} [s_2]_g \xrightarrow[\mathbf{img}\langle f,g,h\rangle(\mathscr{I})]{b} [s_3]_g \\
\Rightarrow\ & s_1' \xrightarrow[\mathscr{I}_2]{f(a)} s_2' \xrightarrow[\mathscr{I}_2]{f(b)} s_3' \text{ for some } s_i' \in [s_i]_g,\ i = 1,2,3 && \text{(Def. 3.13.(3))} \\
\Rightarrow\ & (\exists s_4 \in S_{\mathscr{I}_2}) s_1' \xrightarrow[\mathscr{I}_2]{f(b)} s_4 \xrightarrow[\mathscr{I}_2]{f(a)} s_3' \text{ for some } s_i' \in [s_i]_g,\ i = 1,3 \\
& && \text{(Def.'s 3.13.(1) and 2.3.(2))} \\
\Rightarrow\ & [s_1]_g \xrightarrow[\mathbf{img}\langle f,g,h\rangle(\mathscr{I})]{b} [s_4]_g \xrightarrow[\mathbf{img}\langle f,g,h\rangle(\mathscr{I})]{a} [s_3]_g && \text{(Def. 3.13.(3))}
\end{aligned}
$$

(2) To prove $\mathbf{img}\,\langle f, g, h\rangle\,(\mathscr{I}) \in \mathbf{Int}(\Sigma^f)$, we just have to show that $\varphi_{\mathbf{img}\langle f,g,h\rangle(\mathscr{I})}([s]_g)$ and $\xi_{\mathbf{img}\langle f,g,h\rangle(\mathscr{I})}([s]_g) \sqsubseteq_{\mathbf{img}\langle f,g,h\rangle(\mathscr{I})} \xi_{\mathbf{img}\langle f,g,h\rangle(\mathscr{I})}([s']_g)$ implies $\varphi_{\mathbf{img}\langle f,g,h\rangle(\mathscr{I})}([s]_g)$ for

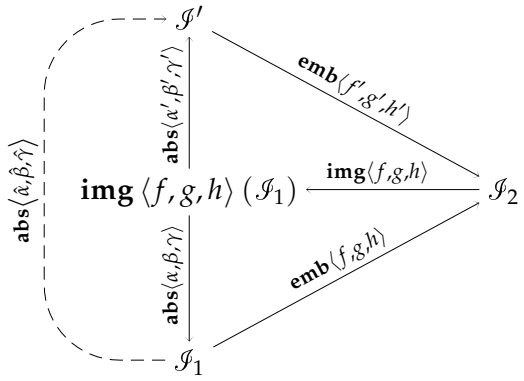


FIGURE 2. Image of an embedding.

all states $[s]_g, [s']_g \in S_{\mathbf{img}\langle f,g,h\rangle(\mathscr{I})}$:

$$\begin{aligned}
&\varphi_{\mathbf{img}\langle f,g,h\rangle(\mathscr{I})}([s]_g) \wedge \xi_{\mathbf{img}\langle f,g,h\rangle(\mathscr{I})}([s]_g) \sqsubseteq_{\mathbf{img}\langle f,g,h\rangle(\mathscr{I})} \xi_{\mathbf{img}\langle f,g,h\rangle(\mathscr{I})}([s']_g)\\
&\quad\Rightarrow \bigvee_{s''\in[s]_g} \varphi_{\mathscr{I}_2}(s'') \wedge \prod_{s''\in[s]_g} \xi_{\mathscr{I}_2}(s'') \sqsubseteq_{\mathscr{I}_2} \prod_{s''\in[s']_g} \xi_{\mathscr{I}_2}(s'') \quad \text{(Def. 3.13)}\\
&\quad\Rightarrow \bigvee_{s''\in[s']_g} \varphi_{\mathscr{I}_2}(s'') \quad \text{(Def. 2.20.(†))}\\
&\quad\Rightarrow \varphi_{\mathbf{img}\langle f,g,h\rangle(\mathscr{I})}([s']_g) \quad \text{(Def. 3.13)}
\end{aligned}$$

□

Thus the image of an system $\mathscr{I}_1$ which is embedded into some environment $\mathscr{I}_2$ is defined as those elements of $\mathscr{I}_2$ which are images of constituents of $\mathscr{I}_1$, and—for actions—are allowed to take place in the context of $\mathscr{I}_2$. Thus the image of a system under some embedding is a behavioral restriction of this system to the embedding context. We already discussed the fact that abstraction means to forget details, in particular enabledness conditions of system actions. Since the embedding of a system into an environment adds those conditions (which are presented in its image), it turns out that the a the embedded system is in fact an abstraction of its image, and moreover, it is the "most concrete" abstraction of its image that can be embedded into the environment.

3.15. THEOREM. *Let $\mathscr{I}_1 \in \mathbf{ValInt}(\Sigma_1)$ and $\mathscr{I}_2 \in \mathbf{ValInt}(\Sigma_2)$ be interpretations of the distributed alphabets Σ_1 and Σ_2, respectively such that $\mathbf{emb}\langle f,g,h\rangle : \mathscr{I}_1 \to \mathscr{I}_2$ is an embedding. Then*

(1) *There is an abstraction $\mathbf{abs}\langle\alpha,\beta,\gamma\rangle : \mathbf{img}\langle f,g,h\rangle(\mathscr{I}_1) \to \mathscr{I}_1$, and moreover,*
(2) *If there is another interpretation $\mathscr{I}' \in \mathbf{ValInt}(\Sigma_2)$ such that $\mathbf{emb}\langle f',g',h'\rangle : \mathscr{I}' \to \mathscr{I}_2$ is an embedding and $\mathbf{abs}\langle\alpha',\beta',\gamma'\rangle : \mathbf{img}\langle f,g,h\rangle(\mathscr{I}_1) \to \mathscr{I}'$ is an abstraction for another valued interpretation $\mathscr{I}' \in \mathbf{ValInt}(\Sigma')$, then there is an unique abstraction $\mathbf{abs}\langle\hat{\alpha},\hat{\beta},\hat{\gamma}\rangle : \mathscr{I}_1 \to \mathscr{I}'$.*

In other words, the arrow $\mathbf{abs}\langle\hat{\alpha},\hat{\beta},\hat{\gamma}\rangle$ that makes the diagram shown in Figure 2 commute is unique.

PROOF. For (1), we construct the abstraction $\mathbf{abs}\langle\alpha,\beta,\gamma\rangle$ as follows:

(a) $\alpha =_{\mathrm{df}} f^{-1}$: since f is injective, f^{-1} is an injective partial function. We furthermore have $f^{-1}(\tau_{\Sigma_2}) = \tau_{\Sigma_1}$, and $f^{-1}(a)\ I_{\Sigma_1}\ f^{-1}(b) \Rightarrow a\ I_{\Sigma_2}\ b$ by Def. 3.10. Thus $\mathbf{abs}\,\langle\alpha\rangle : \Sigma_2 \to \Sigma_1$ is an abstraction.

(b) $\beta([s]_g) =_{\mathrm{df}} g(s)$; by Def. 3.13.(3) we have

$$[s_1]_g \xrightarrow[\mathbf{img}\langle f,g,h\rangle(\mathscr{I}_1)]{a} [s_2]_g \quad\Rightarrow\quad \beta([s_1]_g) \xrightarrow[\mathscr{I}_1]{\alpha(a)} \beta([s_2]_g)$$

whenever $\alpha(a)$ is defined; on the other hand there are no transitions labeled with actions a in $\mathbf{img}\,\langle f,g,h\rangle\,(\mathscr{I}_1)$ such that $\alpha(a)$ is undefined. Note that β is in fact a bijection.

(c) $\gamma =_{\mathrm{df}} h$: The required properties are just by Def. 3.3.

The proof of the part (2) of the Lemma is carried out as follows:

(a) By defining $\hat{\alpha} \circ \alpha =_{\mathrm{df}} \alpha'$ we get a uniquely defined partial function $\hat{\alpha} : A_{\Sigma_1} \rightharpoonup A_{\Sigma'}$ preserving τ_{Σ_1} and I_{Σ_1}.

(b) Since $\beta' : S_{\mathscr{I}_2}/_g \to S_{\mathscr{I}'}$ is surjective, and β is a bijection, there is a uniquely defined surjective function $\hat{\beta} : S_{\mathscr{I}_1} \to S_{\mathscr{I}'}$ defined by $\hat{\beta} \circ \beta =_{\mathrm{df}} \beta'$.

(c) The last part, namely that $\hat{\gamma}$ can be defined by $\hat{\gamma} \circ \gamma =_{\mathrm{df}} \gamma'$, follows by the fact that greatest lower bounds are unique. More precisely, suppose another mapping $q : X_{\mathscr{I}_1} \to X_{\mathscr{I}'}$ such that makes the diagram in Figure 2 commute (i. e. $\mathbf{abs}\,\langle\hat{\alpha},\hat{\beta},q\rangle : \mathscr{I}_1 \to \mathscr{I}'$ is an abstraction). Then

$$\begin{aligned} q\left(\textstyle\prod_{\mathscr{I}_1}(\xi_{\mathscr{I}_1}(S))\right) &= \textstyle\prod_{\mathscr{I}'}(\xi_{\mathscr{I}'}(\hat{\beta}(S))) \\ &= \hat{\gamma}\left(\textstyle\prod_{\mathscr{I}_1}(\xi_{\mathscr{I}_1}(S))\right) \end{aligned}$$

by two applications of Def. 3.3.

□

From the proof of Theorem 3.15 we obtain the following notion of a *canonical abstraction*. The value structure is not needed during the application of the canonical abstraction, thus all elements concerned with values an value mappings are omitted.

3.16. DEFINITION (Canonical abstraction). If $\mathbf{emb}\,\langle f,g\rangle : \mathscr{I}_1 \to \mathscr{I}_2$ is an embedding, then the pair abstraction $\mathbf{abs}\,\langle f^{-1},g\rangle : \mathscr{I}_2 \to \mathbf{img}\,\langle f,g\rangle\,(\mathscr{I}_2)$ is called the *canonical abstraction* of $\mathbf{img}\,\langle f,g\rangle\,(\mathscr{I}_2)$. □

The following corollary is a direct conclusion from Lemma 3.8 and Theorem 3.15.

3.17. COROLLARY. *Let $\mathscr{I}_1 \in \mathbf{Int}(\Sigma_1)$ and $\mathscr{I}_2 \in \mathbf{Int}(\Sigma_2)$ be interpretations of the distributed alphabets Σ_1 and Σ_2, respectively such that $\mathbf{emb}\,\langle f,g\rangle : \mathscr{I}_1 \to \mathscr{I}_2$ is an embedding. Then*

$$g(s_1) \xrightarrow[\mathscr{I}_2]{(f^{-1})^*(u)} g(s_2) \quad\Rightarrow\quad s_1 \xrightarrow[\mathscr{I}_1]{u} s_2$$

for all $u \in \mathbf{Pom}(\Sigma_1)$.

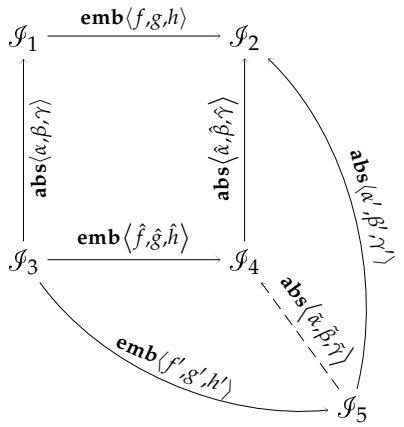


FIGURE 3. Zoom.

3. Zooming

We now consider the following situation. Let $\mathbf{emb}\,\langle f,g,h\rangle : \mathscr{I}_1 \to \mathscr{I}_2$ be an embedding and let $\mathbf{abs}\,\langle\beta,\alpha,\gamma\rangle : \mathscr{I}_3 \to \mathscr{I}_1$ be an abstraction. The question arises whether we can use the embedding process (described by $\mathbf{emb}\,\langle f,g,h\rangle$) also to embed the refined version of $\mathscr{I}_1$ into (a refined version) of $\mathscr{I}_2$? Such an operation would provide us with a *magnifying glass*, a notion of local refinement or *zooming*. Of course, we are not looking for an arbitrary zooming operation but for a universal one in the sense that the resulting refined system is the most abstract one that embeds the interpretation $\mathscr{I}_3$.

3.18. THEOREM. *Let $\mathscr{I}_i \in \mathbf{ValInt}(\Sigma_i)$ interpretations for $i = 1,2,3$ such that $\mathbf{emb}\,\langle f,g,h\rangle : \mathscr{I}_1 \to \mathscr{I}_2$ is an embedding and $\mathbf{abs}\,\langle\alpha,\beta,\gamma\rangle : \mathscr{I}_3 \to \mathscr{I}_1$ is an abstraction. Then there is an interpretation $\mathscr{I}_4 \in \mathbf{ValInt}(\Sigma_4)$, an embedding $\mathbf{emb}\left\langle\hat{f},\hat{g},\hat{h}\right\rangle : \mathscr{I}_3 \to \mathscr{I}_4$, and an abstraction $\mathbf{abs}\,\langle\hat{\alpha},\hat{\beta},\hat{\gamma}\rangle : \mathscr{I}_4 \to \mathscr{I}_2$ such that for all interpretations $\mathscr{I}_5$ and all embedings $\mathbf{emb}\,\langle f',g',h'\rangle : \mathscr{I}_3 \to \mathscr{I}_5$ and abstractions $\mathbf{abs}\,\langle\alpha',\beta',\gamma'\rangle : \mathscr{I}_5 \to \mathscr{I}_2$ there is a uniquely defined abstraction $\mathbf{abs}\,\langle\tilde{\alpha},\tilde{\beta},\tilde{\gamma}\rangle : \mathscr{I}_5 \to \mathscr{I}_4$. In other words, $\mathbf{abs}\,\langle\tilde{\alpha},\tilde{\beta},\tilde{\gamma}\rangle$ is the only abstraction arrow that makes the diagram shown in Figure 3 commute.*

PROOF.

(1) Σ_4 is constructed as follows:
   (i) $A_{\Sigma_4} =_{\mathrm{df}} (A_{\Sigma_3} \setminus \{\tau_{\Sigma_3}\}) + \left(A_{\Sigma_2} \setminus \left(f\left(f^{-1}\left(A_{\Sigma_2}\right)\right) \cup \{\tau_{\Sigma_2}\}\right)\right) \cup \{\tau\}$, where $\tau \notin A_{\Sigma_2} \cup A_{\Sigma_3}$ (recall that $A + B = A \times \{0\} \cup B \times \{1\}$); with other words, the 0-labeled elements of the disjoint union are those actions of $\mathscr{I}_3$ (with the exception of τ_{Σ_3}), while the 1-labeled actions are actions of $\mathscr{I}_2$ which are not the image of actions of $\mathscr{I}_2$ (which in turn are images of actions of $\mathscr{I}_3$), where τ_{Σ_2} is again excluded. To compensate the now missing hidden actions a new element τ is introduced.

(ii) $\langle a, m\rangle \; D_{\Sigma_4} \; \langle b, n\rangle \Leftrightarrow_{\text{df}} m = n \wedge (m = 0 \wedge a \; D_{\Sigma_3} \; b \vee m = 1 \wedge a \; D_{\Sigma_2} \; b)$, with

$$\tau \; D_{\Sigma_4} \; \tau,$$
$$\tau \; D_{\Sigma 4} \; \langle a, 0\rangle \Leftrightarrow \langle a, 0\rangle \; D_{\Sigma 4} \; \tau \Leftrightarrow a \; D_{\Sigma_3} \; \tau, \text{ and}$$
$$\tau \; D_{\Sigma 4} \; \langle a, 1\rangle \Leftrightarrow \langle a, 1\rangle \; D_{\Sigma 4} \; \tau \Leftrightarrow a \; D_{\Sigma_2} \; \tau$$

as special treatment for the newly inserted hidden action τ. Clearly, D_{Σ_4} is a symmetric, irreflexive relation.

(iii) Finally, $\tau_{\Sigma_4} =_{\text{df}} \tau$;

(2) The transition system $\mathcal{I}_4$ comprises the following components:

(i) $S_{\mathcal{I}_4} =_{\text{df}} S_{\mathcal{I}_3} \times S_{\mathcal{I}_2}$ is the state set of $\mathcal{I}_4$

(ii) Its transition relation is defined by

$$\langle s_1, s_2\rangle \xrightarrow[\mathcal{I}_4]{\langle a,1\rangle} \langle s_1', s_2'\rangle \quad \Leftrightarrow_{\text{df}} \quad s_1 \xrightarrow[\mathcal{I}_2]{a} s_1' \wedge s_2 = s_2',$$
$$\langle s_1, s_2\rangle \xrightarrow[\mathcal{I}_4]{\langle a,0\rangle} \langle s_1', s_2'\rangle \quad \Leftrightarrow_{\text{df}} \quad s_2 \xrightarrow[\mathcal{I}_3]{f\circ\alpha(a)} s_2' \wedge s_1 = s_1',$$
$$\langle s_1, s_2\rangle \xrightarrow[\mathcal{I}_4]{\tau} \langle s_1', s_2'\rangle \quad \Leftrightarrow_{\text{df}} \quad s_1 \xrightarrow[\mathcal{I}_3]{\tau_{\mathcal{I}}} s_1' \wedge s_2 = s_2' \vee s_2 \xrightarrow[\mathcal{I}_2]{\tau_{\mathcal{I}}} s_2' \wedge s_1 = s_1'.$$

(3) The value structure of $\mathcal{I}_4$ is defined by

(i) $X_{\mathcal{I}_4} =_{\text{df}} X_{\mathcal{I}_3} \times X_{\mathcal{I}_2}$;

(ii) $\langle x_1, x_2\rangle \sqsubseteq_{\mathcal{I}_4} \langle y_1, y_2\rangle \Leftrightarrow_{\text{df}} x_1 \sqsubseteq_{\mathcal{I}_3} y_1 \wedge x_2 \sqsubseteq_{\mathcal{I}_2} y_2$;

(iii) $\varphi_{\mathcal{I}_4}(\langle s_1, s_2\rangle) =_{\text{df}} \varphi_{\mathcal{I}_3}(s_1) \wedge \varphi_{\mathcal{I}_2}(s_2)$;

(iv) $\xi_{\mathcal{I}_4}(\langle s_1, s_2\rangle) =_{\text{df}} \langle \xi_{\mathcal{I}_3}(s_1), \xi_{\mathcal{I}_2}(s_2)\rangle$;

(4) The abstraction $\mathbf{abs}\,\langle \hat{\alpha}, \hat{\beta}, \hat{\gamma}\rangle : \mathcal{I}_4 \to \mathcal{I}_2$ is defined by the following clauses:

(i) The mapping $\alpha : A_{\Sigma_4} \to A_{\Sigma_2}$ is given by

$$\hat{\alpha}(\langle a, m\rangle) =_{\text{df}} \begin{cases} f(\alpha(a)), & m = 0; \\ a, & m = 1; \end{cases} \text{ and}$$
$$\hat{\alpha}(\tau) =_{\text{df}} \tau_{\Sigma_2};$$

$\hat{\alpha}$ is surjective, and also total in the current case.

(ii) $\hat{\beta}(\langle s_1, s_2\rangle) =_{\text{df}} s_1$;

(iii) $\hat{\gamma}(\langle x, y\rangle) =_{\text{df}} x$;

(5) Finally, the embedding $\mathbf{emb}\left\langle \hat{f}, \hat{g}, \hat{h}\right\rangle$ comprises the following mapping:

(i) $\hat{f}(a) =_{\text{df}} \langle a, 0\rangle$ which is clearly injective

(ii) $\hat{g}(\langle s_1, s_2\rangle) =_{\text{df}} s_2$

(iii) $\hat{h}(\langle x, y\rangle) =_{\text{df}} y$

Having these elements, we firstly have to validate that $\mathcal{I}_4 \in \mathbf{ValInt}(\Sigma_4)$ is a valued interpretation of Σ_4. We only sketch some parts of the proof, as they are immediate from the fact that $\mathcal{I}_2 \in \mathbf{ValInt}(\Sigma_2)$ and $\mathcal{I}_3 \in \mathbf{ValInt}(\Sigma_3)$ are valued interpretations of the respective distributed alphabets.

To show that $\mathbf{Int}(\mathcal{I}_4) \in \mathbf{Int}(\Sigma_4)$ does hold we firstly to validate the diamond properties for $\mathcal{I}_4$. Assume $\langle a, n\rangle \; D_{\Sigma_4} \; \langle b, m\rangle$ for $\langle a, n\rangle, \langle b, m\rangle \in A_{\Sigma_4}$. If $n = m = 0$ ($n = m = 1$), then the diamond properties follow from the fact that $\mathcal{I}_3$ ($\mathcal{I}_2$) enjoys the diamond properties. If $n \neq m$, then we note that $\langle a, n\rangle$ and $\langle b, m\rangle$ operate on different components of the states of $\mathcal{I}_4$

and therefore, they do not influence each other. The cases where one of the action symbols is the hidden action τ_{Σ_4} follow by similar arguments. $\mathscr{I}_4$ is clearly deterministic, because $\mathbf{Int}(\mathscr{I}_2) \in \mathbf{Int}(\Sigma_2)$ and $\mathbf{Int}(\mathscr{I}_3) \in \mathbf{Int}(\Sigma_3)$ are deterministic, and by the same argument, τ_{Σ_4} is hidden in $\mathscr{I}_4$. To see that $\mathbf{Int}(\mathscr{I}_4) \in \mathbf{ValInt}(\Sigma_4)$, we have to prove that the implication 2.20.(†) does hold. We omit the straightforward details.

Now given the arrows $\mathbf{abs}\,\langle \alpha', \beta', \gamma' \rangle : \mathscr{I}_5 \to \mathscr{I}_2$ and $\mathbf{emb}\,\langle f', g', h' \rangle : \mathscr{I}_5 \to \mathscr{I}_3$, we have to construct a unique arrow $\mathbf{abs}\,\langle \tilde{\alpha}, \tilde{\beta}, \tilde{\gamma} \rangle : \mathscr{I}_5 \to \mathscr{I}_4$.

(1) $\tilde{\alpha} : A_{\Sigma_5} \to A_{\Sigma_4}$ is chosen as

$$\tilde{\alpha}(a) \quad =_{\mathrm{df}} \quad \begin{cases} \tau_{\Sigma_4} & \text{if } a = \tau_{\Sigma_5} \\ \left\langle f'^{-1}(a), 0 \right\rangle & \text{if } a = f'(a') \text{ for some } a' \in A_{\Sigma_3} \text{ and } a \neq \tau_{\Sigma_5} \\ \text{otherwise} & \begin{cases} \langle \alpha'(a), 1 \rangle, & \text{if } \alpha'(a) \text{ is defined} \\ \text{otherwise} & \text{undefined} \end{cases} \end{cases}$$

(2) $\tilde{\beta} : S_{\mathscr{I}_5} \to S_{\mathscr{I}_4}$ is defined by $\tilde{\beta}(s) =_{\mathrm{df}} \langle g'(s), \beta'(s) \rangle$.

(3) Finally, $\tilde{\gamma} : X_{\mathscr{I}_5} \to X_{\mathscr{I}_4}$ is given by $\tilde{\gamma}(s) =_{\mathrm{df}} \langle h'(s), \gamma'(s) \rangle$.

It is now an straightforward but tedious exercise to validate that the mappings $\tilde{\alpha}$, $\tilde{\beta}$, and $\tilde{\gamma}$ form an unique abstraction arrow. □

3.19. DEFINITION (Zoom). Let $\mathscr{I}_i \in \mathbf{ValInt}(\Sigma_i)$ interpretations for $i = 1, 2, 3$ such that $\mathbf{emb}\,\langle f, g, h \rangle : \mathscr{I}_1 \to \mathscr{I}_2$ is an embedding and $\mathbf{abs}\,\langle \alpha, \beta \rangle\, \gamma : \mathscr{I}_3 \to \mathscr{I}_1$ is an abstraction. Then the pair $\left\langle \mathbf{emb} \left\langle \hat{f}, \hat{g}, \hat{h} \right\rangle, \mathbf{abs}\,\langle \hat{\alpha}, \hat{\beta}, \hat{\gamma} \rangle \right\rangle$ as defined in proof of the previous theorem is called a *zoom*.

The application of a zoom to the interpretations $\mathscr{I}_1$, $\mathscr{I}_2$, and $\mathscr{I}_3$ yielding the system $\mathscr{I}_4$ is denoted by $\left\langle \mathbf{emb} \left\langle \hat{f}, \hat{g}, \hat{h} \right\rangle, \mathbf{abs}\,\langle \hat{\alpha}, \hat{\beta}, \hat{\gamma} \rangle \right\rangle (\mathscr{I}_1, \mathscr{I}_2, \mathscr{I}_3)$. □

3.20. REMARK. Zooms are not *pushouts* in the sense of category theory, also the diagram looks similar. The problem is that the abstraction arrows go into the wrong direction. If we however modify the category of interpretations, abstractions and embeddings to that of interpretations, refinement and embedding pre-order, then zooms in fact form pushouts. More formally, this new category comprises of objects $\langle \Sigma, \mathscr{I} \rangle$, where $\mathscr{I} \in \mathbf{ValInt}(\Sigma)$, and two types of arrows:

(1) $\langle \Sigma_1, \mathscr{I}_1 \rangle \xrightarrow{\leq} \langle \Sigma_2, \mathscr{I}_2 \rangle$ is an arrow if there is an abstraction $\mathbf{abs}\,\langle \beta, \alpha, \gamma \rangle : \mathscr{I}_2 \to \mathscr{I}_1$, and

(2) $\langle \Sigma_1, \mathscr{I}_1 \rangle \xrightarrow{\subseteq} \langle \Sigma_2, \mathscr{I}_2 \rangle$ is an arrow if there is an embedding $\mathbf{emb}\,\langle f, g, h \rangle : \mathscr{I}_1 \to \mathscr{I}_2$.

It is easy to verify that these objects and arrows indeed form a category. Note that the dual of the zoom concept, the anti-zoom as defined below, forms a *pullback* in this category. □

With help of the observations explained in Remark 3.20 we are able to state the following theorem. It is an application of the pushout lemma of Category Theory (see e.g. [15], pp. 67, here the "dual" *pullback lemma* is presented) translated in our terminology.

3.21. THEOREM. *Consider the diagram shown in Figure 4. If the inner squares form two zooms, then the outer rectangle is also a zoom. If the right hand side inner squares and the outer rectangle are zooms, then the left hand side inner square is also.*

The concept dual to that of a zoom is an anti-zoom. The idea here is that if we have an embedding of a system component into some environment, and an abstraction of the composite system, to extract the associated abstraction of the embedded component. Thus if

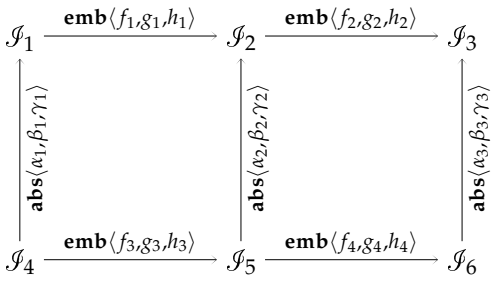


FIGURE 4. Composition of zooms.

the zoom can be used to step "into" the system, the anti-zoom is to do one step back and to look to the embedded subsystem from a more abstract perspective.

3.22. THEOREM. *Let $\mathscr{I}_i \in \mathbf{ValInt}(\Sigma_i)$ interpretations for $i = 1,2,3$ such that $\mathbf{emb}\,\langle f,g,h\rangle : \mathscr{I}_2 \to \mathscr{I}_1$ is an embedding and $\mathbf{abs}\,\langle\alpha,\beta,\gamma\rangle : \mathscr{I}_1 \to \mathscr{I}_3$ is an abstraction. Then there is an interpretation $\mathscr{I}_4 \in \mathbf{ValInt}(\Sigma_4)$, an embedding $\mathbf{emb}\left\langle \hat{f},\hat{g},\hat{h}\right\rangle : \mathscr{I}_4 \to \mathscr{I}_3$, and an abstraction $\mathbf{abs}\,\langle\hat{\alpha},\hat{\beta},\hat{\gamma}\rangle : \mathscr{I}_2 \to \mathscr{I}_4$ such that for all interpretations $\mathscr{I}_5$ and all embeddings $\mathbf{emb}\,\langle f',g',h'\rangle : \mathscr{I}_3 \to \mathscr{I}_5$ and abstractions $\mathbf{abs}\,\langle\alpha',\beta',\gamma'\rangle : \mathscr{I}_5 \to \mathscr{I}_2$ there is a uniquely defined abstraction $\mathbf{abs}\,\langle\tilde{\alpha},\tilde{\beta},\tilde{\gamma}\rangle : \mathscr{I}_4 \to \mathscr{I}_5$. In other words, $\mathbf{abs}\,\langle\tilde{\alpha},\tilde{\beta},\tilde{\gamma}\rangle$ is the only abstraction arrow that makes the diagram shown in Figure 5 commute.*

The concept of anti-zooms are not needed at the present state of our investigations, therefore, the details of the necessary constructions are omitted. It will become of relevance when we start looking into automated system composition and abstraction which is needed for contract based supervision.

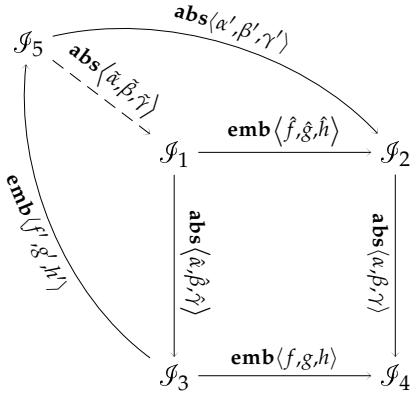


FIGURE 5. Anti-zoom.

CHAPTER 4

Supervision

In this chapter, we are going to investigate supervision algorithms (i.e. control loops). A basic version is presented in the Section 1. Section 2 defines a number of metrics that measure the *competence* of a supervision system (more precisely, of the supervision model in use) on the basis of counters defined in the algorithm from Section 1. The final Section 3 addresses a number of issues concerning hierarchical supervision.

1. Basic Supervision Algorithm

We assume that the system under supervision exposes its internal states and actions to the supervision system, at least in an abstracted form, i.e. the supervision model may be an abstraction of the execution model (or program) of the system under supervision. This information on states and action is expressed in terms of the supervision model.

Supervision now means:

(1) Continuous observation of the current state of the system under supervision and assessment of the admissibility of this state;
(2) If non-admissibility is assessed, some contingency plan, i.e. a pomset executable in the supervision model, is computed. We assume that there is some suitable planning algorithm in place, details have not been investigated yet. There is certainly a relationship to the assessment of the appropriateness or correctness of system behavior, which can be used to select a suitable plan for a given problem situation. Thus planning is related to the assessment of system behavior, which is not yet considered in Work Package 2.
(3) This plan is executed and its effectiveness is assessed.

1.1. Controllability and Observability. The execution and assessment of effectiveness of plans requires that the system under supervision is able to control and to observe the actions that the system under supervision performs, at least to a certain extend.

Let Σ be a distributed alphabet. We define several subsets of action from A_Σ in the following way:

(1) Sets $A_{\Sigma,\mathrm{c}}$ and $A_{\Sigma,\mathrm{uc}}$ of controllable and uncontrollable actions, respectively. We assume that $A_{\Sigma,\mathrm{c}} \cap A_{\Sigma,\mathrm{uc}} = \varnothing$.
(2) Sets $A_{\Sigma,\mathrm{o}}$ and $A_{\Sigma,\mathrm{uo}}$ of observable and unobservable actions, respectively, where $A_{\Sigma,\mathrm{o}} \cap A_{\Sigma,\mathrm{uo}} = \varnothing$.
(3) We suppose that all actions of A_Σ—with the exception of the hidden action τ_Σ—are element of at least one of the sets described above, i. e. $A_\Sigma \setminus \{\tau_\Sigma\} = A_{\Sigma,\mathrm{c}} \cup A_{\Sigma,\mathrm{uc}} \cup A_{\Sigma,\mathrm{o}} \cup A_{\Sigma,\mathrm{uo}}$.

We do not require that all actions that appear in the supervision model are controllable or observable. As the supervision system and the system under supervision are supposed to commit a contract on supervision (which is basically given by the supervision model together with a declaration of observable and controllable actions), insufficient observability or controllability might result in non-effective (but still "contractual") supervision.

1.2. Timing Issues. We need to understand that a concept of time is crucial when performing supervision in a distributed environment. States are usually not "well-defined" in the sense that there are always actions performed by the environment of a system which—although probably not observable—alter states both of the system and its environment. This means, if we observe an action a that occurs at some (abstract) state s, then it is not ensured that this action actually (more precisely, its concrete counterpart) has been performed at some concrete system state $s' \in \beta^{-1}(s)$, where β is of course a state abstraction mapping.

We thus establish the assumptions that there is a notion of time. At the current stage, a global synchronized time is not required. We only stipulate the assumption that it is particularly possible to assign a time stamp $T(a)$ to each occurrence of a monitored action a, and similarly, a time stamp assigned to each state measurement that allows to access the state measured at time t (which is unique only to the execution environment of the action a). Time is linearly ordered (i.e. it is reasonable to take the set $\mathbb{R}$ to express points in time), and since it is measured by a technical process of sampling, we consider a sequence of time stamps $\ldots, t_{-1}, t_0, t_1, \ldots$ such that $t_i - t_{i-1} = c$, where $c \in \mathbb{R}$ is a constant.

Of course this bears various simplifications and assumptions: Different sensors might define different sampling rates, and sampling rates might not be constant but dependent of uncontrollable (or even unknown) circumstances. Furthermore, by choosing the sequence of time stamps unbounded both into the direction of negative and positive indices, we assume that the supervision process has an "infinite memory" in the sense that it is able to access state measurements that has been recorded arbitrarily in the past. By $S(r)$ we denote the state that the system assumes at time point $r \in \mathbb{R}$. Hence, the states stored by the the supervision systems at time point r are $\{S(t_i) : i \in \mathbb{Z} \wedge t_i \leqslant r\}$. We however have to assume that the sampling rate that our sensors can provide is high enough to register all "relevant" state changes:

$$S(r_1) \neq S(r_2) \quad \Rightarrow \quad |r_1 - r_2| > c$$

The above implication defines an absolute performance limit for supervision. If it is not valid, then effective supervision is logically impossible. Of course, even if the implication is valid, the supervision system might still be to slow to enforce a controllable action in time. Finally, by choosing a *time stamp* that indicates the point in time $T(a)$ where an action a occurs basically means that actions occur instantaneous. This assumption can be made more weak (and realistic) is we use a start time stamp and an end time stamp assigned to action occurrences. This however makes our approach neither more expressive nor more problematic, but adds only complexity to its presentation, we thus refrain from going this way.

By the assumptions presented so far it is however easy to define the validity of an action:

$$a \text{ is valid} \quad \Leftrightarrow_{\mathrm{df}} \quad S(T(a)) \xrightarrow[\mathscr{G}]{a} S(T(a) + c).$$

SUPERVISE()
1 SENSE(s)
2 **if** $\neg\varphi_{\mathcal{J}}(s)$
3 **then compute** $s' \in S_{\mathcal{J}}; u \in$ **Pom**$^{\mathrm{w}}_{\mathcal{J}} : s \xrightarrow[\mathcal{J}]{u} s'$ and $\varphi_{\mathcal{J}}(s')$
4 EXECUTE(u);

FIGURE 1. Basic supervision procedure

1.3. Algorithm. The basic supervision algorithm is presented in the Figures 1 and 2, respectively. The algorithm SUPERVISE makes use of a monitoring procedure SENSE(s) which observes the current state of the system under supervision and detects a corresponding state s in the supervision model $\mathcal{J}$ (thus the procedure SENSE provides us with an elementary state abstraction mechanism β that translate from the "real" system to the abstract system model). If this state is not admissible it calls a planning algorithms which computes a pomset u. The execution of u is supposed to lead the system under supervision back into an admissible state.

4.1. REMARK. The exact definition of the planning algorithm has not been considered yet in the CASCADAS project; this will be done in a later project phase. □

4.2. REMARK. The current formulation of the basic supervision algorithm uses a plan that does not contain choices with respect to variation of the behavior of the system under supervision or its environment. Decision trees are an attractive alternative to those static plans. Formally, those decision trees can be expressed as sets of pomsets that are ordered by the prefix relation $\leqslant$ (compare Def. 2.9.(9)), or—equivalently—by means of *event structures*. Informally, event structures are pomsets that comprise explicit choice points. Behavior that is executed after a choice point is passed in never unified again. Thus event structures expose a tree-like structure with branches defined by choice points. This will also be considered in a later project phase. □

To simplify the presentation of the algorithm shown in Figure 2 we write $a \in \min v$ if there is some $e \in E_v$ such that $\tilde{e} = a$ and $e \in \min_{\leqslant_a} E_v$. Note that by the definition of pomsets $v \in \mathbf{Pom}^{\mathrm{w}}(\Sigma)$ the event e in question is always unique if it exists. For the same reason, the operation

$$v \setminus a \quad =_{\mathrm{df}} \quad v\left[E_v \setminus \left\{e \in E_v \mid e \in \min_{\leqslant_v} E_v \wedge \tilde{e} = a\right\}\right]$$

that removes a single, minimal event e labeled with a from v if $a \in \min v$ is well-defined.

The procedure used a number of subroutines:

(1) VALIDATE(a) validates the execution of the action a in the system under supervision;
(2) ENFORCE(a) executed the action a in the system under supervision;

The action set *Exp* of expected actions contains those controllable and observable actions which have been enforced by the supervision system but have not been monitored yes. A timer is used to prevent from "starvation" in the case the that none of the expected events happen in the system under supervision.

```
EXECUTE(v)
 1  u : Pom^w(Σ) ← ϵ
 2  pending : Bool ← false
 3  t : timer ← ...
 4  Exp : Set of A_Σ ← ∅
 5  N ← T ← E ← A ← 0;
 6  while v ≠ ϵ ∧ pending
 7  do choice
 8       alt monitor a ⇒
 9                 N ← N + 1;
10                 if a ∈ A_Σ,o
11                   then if VALIDATE(a) = NIL
12                          then error "action not appropriate";
13                               A ← A + 1;
14                        if a ∈ min v
15                          then v ← v \ a
16                               u ← u ∘_Σ a
17                               if a ∈ A_Σ,c
18                                 then if a ∈ Exp
19                                        then Exp ← Exp \ {a}
20                               if a D_Σ b  for some  b ∈ Exp
21                                 then error "not in time";
22                                      T ← T + 1
23                          else error "unexpected event";
24                               A ← A + 1
25                   else error "unknown event";
26                        A ← A + 1
27       alt select a ∈ min v : a ∈ A_Σ,c ⇒
28                 N ← N + 1;
29                 if ENFORCE(a) = NIL
30                   then error "action enforcement not effective";
31                        E ← E + 1;
32                 v ← v \ a
33                 if a ∉ A_Σ,o
34                   then u ← u ∘_Σ a
35                   else Exp ← Exp ∪ {a}
36       alt timeout t ⇒
37                 N ← N + 1; error "no progress"
38                 A ← A + 1;
39    pending ← Exp ≠ ∅
40  if pending
41    then error "unable to enforce actions"
42         E ← E + 1;
43  SENSE(s)
44  if ¬φ_J(s)
45    then error "supervision not appropriate"
46         A ← A + 1;
```

FIGURE 2. Procedure EXECUTE.

2. Metrics for Self-assessment

The algorithm presented in Figure 2 also makes use of a number of error notifications which can be used to determine the "competence" of the supervision algorithm with respect to its supervision tasks. The algorithm contains a number of counters that determine the number of trials and failures to monitor or to enforce a certain actions. The counter N contains the total number of those trials. We furthermore consider the following metrics:

2.1. Effectiveness $1 - E/N$**.** Refers to the ability of a supervision system to enforce countermeasures at all. In the algorithm of Figure 2, the counter E in increased in the following situations:

(1) Line 30: The enforcement of an actions fails, and the failure is directly observable by the supervision system.
(2) Line 41: Some of the observable actions that have been enforced by the supervision system could not observed.

2.2. Timeliness $1 - T/N$**.** Is the ability to react in time. Timeliness can only be detected indirectly by the perception of an event a that impacts the enabledness of those events that are expected (the counter T is increased in line 22). In our terminology, this is an event that is dependent of one of the expected events in the set Exp. Note that the occurrence of a does not necessarily prohibit from the execution of v in the supervised system but would lead to a pomset which is not executable in the supervision model.

2.3. Appropriateness $1 - A/N$**.** Means that the system has the capabilities to determine a sequence of actions that leads to the desired results. Non-appropriateness is detected in several ways:

(1) Line 13: The validation of a monitored actions fails, i.e. the execution of the action leads to a result that is different from that specified in the supervision model.
(2) Line 23: The observed event is—although specified in the model—not expected at the current state.
(3) Line 25: The observed event is not specified in the model at all.
(4) Line 37: The system under supervision does not produce any event within a given time frame (a timeout occurs), although the supervision model predicts such events.
(5) Line 45: The system under supervision ended up in a non-admissible state after the execution of the plan v.

The metrics defined above are obviously not useful only for self-assessment of the competence of the supervision system, but also for self-optimization. This topic has however not yet been investigated.

3. Hierarchical Supervision

We are now turn to the issue of hierarchical supervision.

3.1. Model Trees. We now assume that supervision is not performed by means of a "flat" interpretation but by means of a hierarchical organized models, i.e. a tree of interpretations.

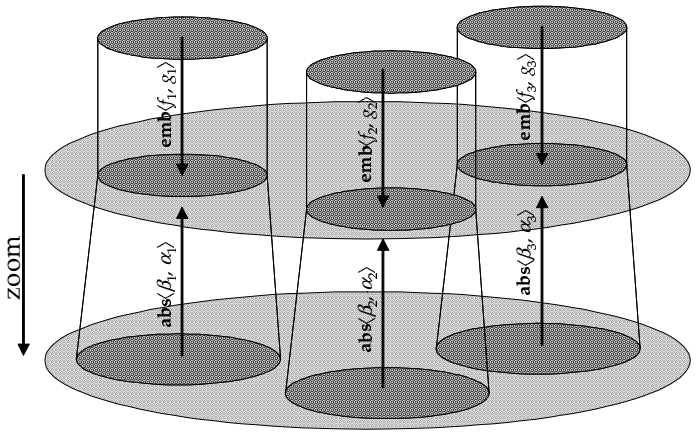


FIGURE 3. Illustration of the concept of compositions. Shown are three interpretations that are embedded into some environment (upper gray ellipsoid). Additional abstractions provide a means to embed more concrete versions of these interpretations into the environment (i.e. to zoom into it), thus producing a more concrete overall interpretation (lower gray ellipsoid).

A *composition* $\mathscr{C} = \langle J, \{\mathscr{I}_i\}_{i\in J}, \mathscr{E}, \mathscr{F}\rangle$ is given by a finite set of indices J, a finite family $\{\mathscr{I}_i\}_{i\in J}$ of interpretations (called *components* and defined over suitable distributed alphabets) and an interpretation $\mathscr{E}$ (called *environment*, also defined over a suitable distributed alphabet) such that there is a set of embeddings $\mathscr{F} = \{\mathbf{emb}\,\langle f_i, g_i, h_i\rangle : \mathscr{I}_i \to \mathscr{E}\}_{i\in J}$. With other words, all "abstract" systems that we consider are given by a number of components that communicate with each other by means of their environment.

A *model tree* is a given by a set $\boldsymbol{C}$ of compositions together with a *refinement relation* $\to \subseteq \boldsymbol{C} \times \boldsymbol{C}$ that forms a tree (i.e. there is exactly one composition that has no predecessor w.r.t. $\to$, while any other interpretation in $\boldsymbol{C}$ has exactly one predecessor w.r.t. $\to$), and moreover the following is true:

(1) Each composition $\langle J, \{\mathscr{I}_i\}_{i\in J}, \mathscr{E}, \mathscr{F}\rangle$ has exactly $|J|$ successors w.r.t $\to$, such that we can define a mapping $\mathrm{succ}_i : \langle J, \{\mathscr{I}_i\}_{i\in J}, \mathscr{E}, \mathscr{F}\rangle \mapsto \langle J_i, \{\mathscr{I}_j\}_{j\in J_i}, \mathscr{E}_i, \mathscr{F}_i\rangle$ that assigns to a composition $\langle J, \{\mathscr{I}_i\}_{i\in J}, \mathscr{E}, \mathscr{F}\rangle$ its i^{th} successor $\langle J_i, \{\mathscr{I}_j\}_{j\in J_i}, \mathscr{E}_i, \mathscr{F}_i\rangle$ for each $i \in J$;

(2) If $\langle J', \{\mathscr{I}'_i\}_{i\in J'}, \mathscr{E}', \mathscr{F}'\rangle = \mathrm{succ}_i(\langle J, \mathscr{C}, \mathscr{E}, \mathscr{F}\rangle)$, then there is an abstraction $\mathbf{abs}\,\langle \alpha_i, \beta_i, \gamma_i\rangle : \mathscr{I}_i \to \mathscr{E}'$.

By the terminology provided in Section 2.3 the pairs $\langle \mathbf{emb}\,\langle f_i, g_i, h_i\rangle, \mathbf{abs}\,\langle \alpha_i, \beta_i, \gamma_i\rangle\rangle$ form a zooms for each "change of level" in the model tree; the refined model is then given by

$$\langle \mathbf{emb}\,\langle f_i, g_i, h_i\rangle, \mathbf{abs}\,\langle \alpha_i, \beta_i, \gamma_i\rangle\rangle\,(\mathscr{I}, \mathrm{succ}_i(\mathscr{I}), \mathscr{E}).$$

Thus if a problem is detected in a system that corresponds to some subtree of the model tree, then we are able to produce a dedicated refinement that considers "unproblematic" system parts from a top level perspective, while effected parts can be arbitrarily refined.

It has to be noticed that—although a model tree separates different subtrees from each other—the environments of the compositions within different subtrees that build the model tree may be related to the same part of the concrete underlying system. They may "share"

actions as well as state information. Thus the term *hierarchical supervision* does not actually assume that the system under supervision is organized in a hierarchical, tree-like fashion. It is the chosen structure of abstractions and embeddings which defines the tree.

3.2. Hierarchical Planning. We now address the question how model trees can be used to perform hierarchical planning, i.e. how to refine a plan recursively according to the tree structure. Of course it is not desirable to "flatten" the whole tree. We aim on producing a model that is just as concrete as it needs to be.

As before we assume that there is a planning procedure working on interpretations. Assume that $\langle \boldsymbol{C}, \rightarrow \rangle$ is a model tree and let u be a pomset. A *distribution* of u is a mapping $\Delta : \mathscr{C} \mapsto \langle s, v \rangle$ for each composition $\mathscr{C} = \langle J, \{\mathscr{I}_i\}_{i \in J}, \mathscr{E}, \mathscr{F} \rangle \in \boldsymbol{C}$, such that $s \in S_{\mathscr{E}}$ and $v \in \mathbf{Pom}^{\mathrm{w}}_{\mathscr{E}}(s)$, and moreover, if $\langle v, s \rangle = \Delta(\mathscr{C})$, and $\Delta(\mathrm{succ}_i(\mathscr{C})) = \langle v', s' \rangle$, then $v = f_i^*(v')$ and $s = \beta_i(s')$, where $\mathbf{abs}\,\langle f_i, g_i \rangle : \mathscr{E} \rightarrow \mathbf{img}\,\langle f_i, g_i \rangle\,(\mathscr{I}_i)$ is the canonical abstraction (recall Def. 3.16), and f_i^* is the extension of the mapping f_i to pomsets (recall Def. 3.6).

This somewhat complicated looking definition just expresses the idea that on each level of the model tree, an appropriate concretization of the plan that is assigned to the root of the tree is used.

3.3. Hierarchical Supervision Algorithm. We will not give an programmatic formulation of this algorithm, as this formulation is pretty obvious. Each node of the model tree is now "equipped" with a separate instance of the supervision algorithm, and all this instances run in parallel. Planning in a flat model is now replaced by a plan distribution. Each "supervision node" receives the plan that is assigned to its node in the model tree. All these plans are executed concurrently.

Supervision activities should start at the leaves of the supervision tree. If a problem is not solvable on the most concrete level, the process has to be repeated on a higher level of abstraction. Here, not only the actions of the local component where an non-admissible state occurred can be taken into account, but also the actions of "neighbored" components that cannot be controlled by the faulty component in question (and not by the local supervision system). Thus supervision activities move upward the model tree (and the level of abstraction) until either the problem is solved (i.e. the system is in an admissible state), or the root of the model tree is reached, and all supervision activities have failed.

CHAPTER 5

Summary and Further Work

We have presented a framework for model-based hierarchical supervision that bases on an abstract notions of the terms “system”, “behavior”, “abstraction”, and “composition”. The framework have been presented in a “syntax-free” form, i.e. only with the weakest assumption on the modeling formalisms that is used for concrete supervision models. The approach is intended as a foundation and preparation of the further work on contract-based supervision which will be performed in the CASCADAS Work Package 2.

Since the material we presented is far from being completed, a number of open issues can be identified:

Firstly, at the current stage, supervision models are static in the sense that there is no mechanism in place that adapts a model to according to observations of the system under supervision and its environment. For that, we propose:

(1) The use of more advanced value structures that provide not only an assessment of the suitability of states, but also an assessment of actions according to their effectiveness, success probability and their costs.
(2) The use of the metrics introduced in Section 4.2 for the adjustment quantitative adjustment of probability and cost parameters of the advanced value structure.
(3) Investigation of ways of qualitative adjustment of the underlying supervision model according to observations.

The notion of a hierarchical supervision algorithm is still a bit fuzzy. Of course, it is strongly related to the hierarchical set-up of the Viable System Model (VSM) that is used as a reference functional architecture of supervision pervasions. The precise justification of the relationship of the VSM an the notion of model-based supervision will be investigated in the further course of the CASCADAS project.

The notions of abstractions, embeddings, and zooms provide us with a framework to understand model hierarchies and changes of the level of abstraction. For contract-based supervision, we however need a mechanism to construct a common supervision model from the operational models of the members of an ensemble of system components (either strongly organized within an ACE or weakly set up as a self-organized community of ACEs). Of course, a “magic” operation that gives us the right supervision model layered in a suitable hierarchy of abstractions is very unlikely to be available. We however foresee the possibility to work with standard mechanisms that solve the problem for a wide range of applications.

Bibliography

[1] S. Abramsky and A. Jung. Domain theory. In S. Abramsky, D. M. Gabbay, and T. S. E. Maibaum, editors, *Handbook of Logic in Computer Science*, volume 2 — Semantic Structures, chapter 1, pages 1 – 168. Clarendon Prss, Oxford, 1994.

[2] M. Arbib and E. Manes. *Arrows, Structures, and Functors: The Categorical Imperative*. Academic Press, New York, 1975.

[3] E. Badouel and P. Darondeau. Theory of regions. In W. Reisig and G. Rozenberg, editors, *Lectures on Petri Nets I: Basic Modells*, number 1491 in Lecture Notes in Computer Science, pages 529 – 586. Springer-Verlag, 1998. Lecture Notes of the 3rd Advanced Course on Petri Nets, Dagstuhl (1996).

[4] L. Bernardinello. Synthesis of net systems. In *Proc. of the Int. Conf. on Application and Theory of Petri Nets (ICATPN'93)*, number 691 in Lecture Notes in Computer Science, pages 89 – 105. Springer-Verlag, 1993.

[5] E. Best and C. Fernández. *Nonsequential Processes*, volume 13 of *EATCS Monographs on Theoretical Computer Science Series*. Springer-Verlag, 1988.

[6] G. Birkhoff. *Lattice Theory*. American Mathematical Society, Providence, Rhode Island, 3rd edition, 1967.

[7] G. Birkhoff. *Lattice Theory*, volume 25 of *Colloquium Publications*. American Mathematical Society, Providence, Rhode Island, 3 edition, 1967.

[8] P. Deussen. Algorithmic aspects of concurrent automata. In H.-D. Burkhard, L. Czaja, and P. Starke, editors, *Workshop on Concurrency, Specification & Programming '98*, number 110 in Informatik-Berichte, pages 39–50, Berlin, 1998. Humboldt Univ. zu Berlin.

[9] P. Deussen. Concurrent automata. Technical Report 1-05/1998, Brandenburg Tech. Univ. Cottbus, 1998.

[10] P. Deussen. Improvements of concurrent automata generation. Technical Report I-08/1998, Brandenburg Tech. Univ. Cottbus, 1999.

[11] P. H. Deussen. *Analyse verteilter Systeme mit Hilfe von Prozessautomaten*. PhD thesis, Brandenburg Technical Univ. of Cottbus, 2001. In German.

[12] R. P. Dillworth and P. Crawley. *Algebraic Theory of Lattices*. Prentice-Hall, 1973.

[13] M. Droste. Concurrency, automata and domains. In M. S. Paterson, editor, *Automata, Languages and Programming*, number 443 in Lecture Notes in Computer Science, pages 195 – 202. Springer-Verlag, 1990.

[14] A. Ehrenfeucht and G. Rozenberg. Partial 2-structures; part ii: State space of concurrent systems:. *Acta Informatica*, 27:348 – 368, 1990.

[15] R. Goldblatt. *Topoi, the Categorial Analysis of Logic*. North Holland Publishing Company, 1979.

[16] J. Grabowski. On partial languages. *Fund. Inform*, 4(2):427–498, 1981.

[17] G. Grätzer. *General Lattice Theory*. Birkhäuser Verlag, Basel, 1978.

[18] E. Kindler. *Modularer Entwurf verteilter Systeme mit Petrinetzen*. Dieter Belz Verlag, Berlin, 1995.

[19] S. M. Lande. *Categories for the Working Mathematics*, volume 5 of *Graduate Texts in Mathematics*. Springer Verlag, New York, 1977.

[20] S. Mauw and M. A. Reniers. Operational semantics for msc'96. In A. Cavalli and D. Vincent, editors, *Tutorials of the Eighth SDL Forum SDI'97: Time for Testing - SDL, MSC and Trends*, pages 135–152, Evry, France, 1997. Institut national des tlcommunications.

[21] A. Mazurkiewicz. Concurrent program schemes ant their interpretations. Technical Report DAIMI PB. 78, Aarhus University, Aarhus, 1977.

[22] A. Mazurkiewicz. Introduction to trace theory. In V. Diekert and G. Rozenberg, editors, *The Book of Traces*, chapter 1, pages 3 – 42. World Scientific, Singapore — New Jersey — London — Hong Kong, 1995.

[23] M. Nielsen, G. Plotkin, and G. Winskel. Petri nets, event structures and domains, Part I. *Theoretical Computer Science*, 13:85–108, 1981.

[24] M. Nielsen, G. Rozenberg, and P. S. Thiagarajan. Behavioural notions for elementary net systems. *Distributed Computing*, 4:45 – 57, 1990.

[25] M. Nielsen, G. Rozenberg, and P. S. Thiagarajan. Elementary transition systems. *Theoretical Computer Science*, 96:3 – 33, 1992.

[26] M. Nielsen and G. Winskel. Models of concurrency. In S. Abramsky, D. M. Gabbay, and T. S. E. Maibaum, editors, *Handbook of Logic in Computer Science*, volume 4 — Semantic Modelling, chapter 1, pages 1 – 148. Clarendon Press, Oxford, 1995.

[27] W. Penzek and R. Kuiper. Traces and logic. In V. Diekert and G. Rozenberg, editors, *The Book of Traces*, chapter 1, pages 307 – 390. World Scientific, Singapore — New Jersey — London — Hong Kong, 1995.

[28] V. Pratt. Modelling concurrency with partial orders. *International Journal of Parallel Programming*, 15(1):33–71, 1986.

[29] V. Pratt. Debate'90: An electronic discussion on true concurrency. In D. A. Peled, V. Pratt, and G. J. Holzmann, editors, *Proc. DIMACS Workshop on Partial Order Methods in Verification*, volume 29 of *DIMACS—Series on Discrete Mathematics and Theoretical Computer Science*, pages 359–403. American Mathematical Society, 1996.

[30] D. K. Probst and H. F. Li. Modelling reactive processes using partial orders. In *Semantics for Concurrency*, Workshops in Coputing, pages 324 – 343, Leicester, 1990. Springer-Verlag.

[31] D. K. Probst and H. F. Li. Partial-order model checking: A guide for the perplexed. In *Proc. of CAV'91*, number 575 in Lecture Notes in Computer Science, pages 322 – 331. Springer-Verlag, 1991.

[32] G. Rozenberg. Behaviour of elementary net systems. In W. Brauer, editor, *Petri Nets: Central Models and their Properties; Advances in Petri Nets; Proc. of an Advanced Curse, Vol. 1*, number 254 in Lecture Notes in Computer Science, pages 60–94, Berlin-Heidelberg-New York, 1986. Springer-Verlag.

[33] G. Rozenberg and J. Engelfriet. Elementary net systems. In W. Reisig and G. Rozenberg, editors, *Lectures on Petri Nets I: Basic Modells*, number 1491 in Lecture Notes in Computer Science, pages 12 – 121. Springer-Verlag, 1998. Lecture Notes of the 3rd Advanced Course on Petri Nets, Dagstuhl (1996).

[34] P. H. Starke. Processes in Petri nets. *J. Inf. Process. Cybern. EIK*, 17(8/9):389–416, 1981.

[35] P. H. Starke. Graph grammars for Petri net processes. *J. Inf. Process. Cybern. EIK*, 19(4/5):199–233, 1983.

[36] P. H. Starke. Multiprocessor systems and their concurrency. *J. Inf. Process. Cybern. EIK*, 20(4):207–427, 1984.

[37] P. H. Starke. *Analyse von Petri-Netz-Modellen*. G. B. Teubner, Stuttgart, 1990.

[38] P. S. Thiagarajan and J. G. Henriksen. Distributed versions of linear time temporal logic: A trace perspective. In W. Reisig and G. Rozenberg, editors, *Lectures on Petri Nets I: Basic Modells*, number 1491 in Lecture Notes in Computer Science, pages 643 – 679. Springer-Verlag, 1998. Lecture Notes of the 3rd Advanced Course on Petri Nets, Dagstuhl (1996).

[39] W. Vogler. *Modular construction and partial order semantics of Petri nets*. Number 625 in Lecture Notes in Computer Science. Springer-Verlag, 1992.