

# The Testing Paradigm Applied to Network Structure

*Tom Verhoeff*

Department of Mathematics and Computing Science  
Eindhoven University of Technology  
P.O. Box 513, 5600 MB EINDHOVEN, The Netherlands  
E-mail: wstomv@win.tue.nl

January 1990, Revised February 1994

## **Abstract**

The *testing paradigm* provides a simple framework for comparing networks of processes. To apply the testing paradigm, one needs a suite of *tests* and a *test criterion* expressing when a network passes a test. Two networks are considered *testing equivalent* when they pass the same tests. In all applications of the testing paradigm that we have seen, tests “probe” (some of) the *behavior* of the process network under test. Network *structure*, however, is mostly handled in an ad hoc way.

In this note, we use the testing paradigm to compare structural aspects of process networks. Central to our approach are the following three ingredients: (i) Tests are drawn from the set of process networks, that is, each test is itself just a process network. (ii) A (global) correctness concern, in the form of a predicate, expresses when a network is correct as an autonomous system. (iii) A network passes a test (by another network) when the composition of two networks involved is a correct (autonomous) system.

Our approach has several merits. It allows a uniform treatment of structure and behavior. Structural and behavioral correctness concerns can be varied independently within the same framework. Structural correctness concerns can be made explicit at the very beginning, and need not appear implicitly as an unmotivated afterthought. Several phenomena, such as nondeterminism, can be illustrated solely in terms of structure, without getting bogged down by behavioral complications.

For one particular choice of (structural) correctness concerns, we work out a model in full detail. We briefly investigate alternative correctness concerns.

**Contents**

<b>1</b>	<b>Introduction</b>	<b>1</b>
<b>2</b>	<b>Pre-Abstract Model</b>	<b>2</b>
<b>3</b>	<b>Basic Concepts for a Fully Abstract Model</b>	<b>6</b>
<b>4</b>	<b>Pointwise Analysis of Correctness</b>	<b>8</b>
<b>5</b>	<b>Construction of Fully Abstract Model</b>	<b>14</b>
<b>6</b>	<b>Discussion of Fully Abstract Model</b>	<b>23</b>
<b>7</b>	<b>Alternative Structural Correctness Concerns</b>	<b>27</b>
<b>8</b>	<b>Conclusion</b>	<b>30</b>
<b>A</b>	<b>Countable Bags</b>	<b>30</b>
<b>B</b>	<b>Partial Orders and Complete Lattices</b>	<b>32</b>
	<b>References</b>	<b>34</b>

## 1 Introduction

We study the structure of process networks, ignoring their behavior. Our main goal is to give a uniform treatment of the structural aspects of process networks. We do so by working out one example model in detail.

Why study structure separately? Usually, structural correctness concerns are simply incorporated at the “syntactic” level of a model. In that approach, composition is disallowed when it would yield a composite that is somehow (structurally) undesirable. For instance, for asynchronous circuits it is customary to prohibit the connection of two output ports (see ‘output interference’ in [5]). A disadvantage of this approach is that composition is a partial operator and, consequently, many propositions about composition need to be decorated with ad hoc syntactic preconditions (so-called boiler plates; see ‘connectable’ in [7]). In our approach we do not disallow any compositions, but we formulate our judgment of desirability in an explicit structural correctness concern. The correctness concern can be used to define a notion of testing. Using the testing paradigm of [3, 6], it then gives rise to a refinement and an equivalence relation. The result is a mathematically clean formalism.

It may appear as if the formalism that we develop in this note to deal with network structure is far too heavy for its purpose. Admittedly, it is often much easier to deal with structural correctness than behavioral correctness. However, more complex structural correctness concerns require more powerful methods. Furthermore, it turns out that the methods needed to deal with network behavior are very similar (see [10]). We have made the formalism in this note more general than is strictly necessary, so that behavioral aspects can be incorporated with little effort. This note is an opportunity for the reader to become familiar with the general methods in a context where the results are fairly easy to predict by intuition. However, we urge the reader to make as little use as possible of these intuitions.

### Overview

In Section 2, we present a pre-abstract model. We start the presentation by defining the set  $\mathcal{SYS}$  of all systems (process networks). On  $\mathcal{SYS}$  we then define structural composition *par* and correctness criterion *Correct*. This induces relations *sat* and *equ* on  $\mathcal{SYS}$  in a straightforward way (also see [10]). Relation *sat* captures refinement and *equ* expresses system equivalence. Thus we obtain a pre-abstract model consisting of the algebra  $\langle \mathcal{SYS}; par, sat \rangle$  with congruence *equ*. The model is called pre-abstract because many networks are distinguished in  $\mathcal{SYS}$  that we wish to identify since they are equivalent for all relevant purposes.

We are interested in the quotient algebra  $\langle \mathcal{SYS}; par, sat \rangle / equ$  consisting of the *equ*-congruence classes. In Sections 3, 4, and 5 we develop an isomorphic fully abstract model. The objects of this abstract model are functions on the set  $\Sigma$  of link identifiers satisfying certain properties. In Section 6 we discuss some of the properties of the fully abstract algebra. We look at alternative structural correctness concerns in Section 7. Finally, Section 8 contains concluding remarks. Appendix A defines our notation for count-

able bags and Appendix B summarizes some lattice theory.

## 2 Pre-Abstract Model

The pre-abstract model presented in this section was directly inspired by work of van de Snepscheut [9], Udding [7, 8], and Ebergen [5, 4].

### Alphabets, processes, and systems

As far as structure is concerned, all we care to know about a process are the names of its communication ports and the direction of each port (either input or output). In our model, the communication links that connect ports convey signals only; there is no data transport. That way, a link may be implemented by a single wire in an electronic circuit. Data may be encoded by employing several links. If data communication is to be incorporated on a higher level into the model then each port could also have a data type.

Let  $\Sigma$  be an infinite set of **symbols**, playing the role of port and link identifiers. Typical (distinct) symbols in  $\Sigma$  are  $a$ ,  $a_0$ ,  $a_1$ ,  $b$ , and  $c$ . Variables  $a$ ,  $b$ , and  $c$  range over  $\Sigma$ . An **alphabet** is a subset of  $\Sigma$ .

In this note, a **process** is simply a pair  $(I, O)$  of disjoint alphabets, where  $I$  is the set of **input ports**, or inputs for short, and  $O$  the set of **output ports** (outputs). There is no behavior associated with a process. The set of all processes is denoted by  $\mathcal{PROC}$ . Variables  $P$ ,  $Q$ , and  $R$  range over  $\mathcal{PROC}$ . The projection functions **i** and **o** on processes are defined by

$$P = (\mathbf{i}P, \mathbf{o}P) .$$

A **system** is an countable bag (see Appendix A) of processes. Note 2.3 below motivates our choice for countable bags. The set of all systems is denoted by  $\mathcal{SYS}$ . Variables  $S$ ,  $T$ , and  $U$  range over  $\mathcal{SYS}$ .

A system models (the topology of) a process network as follows. All ports with the same name, say  $a$ , are connected by a single communication **link**, which will also be named  $a$ . Ports with different names are not so connected. Thus, links are implicitly given in a system. Notice that a link never connects a process to itself. A “self-link” must be simulated by introducing a separate process that behaves like a link.

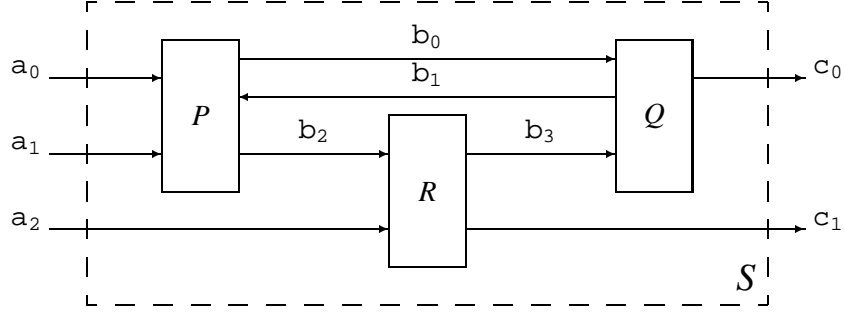
**2.1 Example** Let system  $S$  be defined as  $[P, Q, R]$  ( $[-]$  being the bag constructor, see Appendix A), where

$$\begin{aligned} P &= (\{a_0, a_1, b_1\}, \{b_0, b_2\}) , \\ Q &= (\{b_0, b_3\}, \{b_1, c_0\}) , \text{ and} \\ R &= (\{a_2, b_2\}, \{b_3, c_1\}) . \end{aligned}$$

System  $S$  may be depicted as in Figure 1. Notice that each link in  $S$  connects to at most one input port and one output port. We will come back to this when defining correctness.

■

■

Figure 1: Topology diagram of system  $S$ 

### Composition and correctness

**Structural composition**, or composition for short, is a binary operator on  $\mathcal{S}\mathcal{Y}\mathcal{S}$ , denoted by *par* and defined as bag summation. It is a total operator, is commutative and associative, and has the empty bag as unit. Composition is easy to carry out in terms of electronic circuits: a circuit for  $S \text{ par } T$  is obtained from a circuit for  $S$  and a circuit for  $T$  by fusing wire nets with the same name.

On  $\mathcal{S}\mathcal{Y}\mathcal{S}$  we want to define a predicate *Correct* that captures our (structural) **correctness concerns**. *Correct.S* is intended to express the conditions under which correct **autonomous** operation of system  $S$  is guaranteed. ‘Autonomous’ here means that  $S$  is put to work all by itself, without hooking it up to some “environment”.

The particular requirements that we have chosen for our example model throughout this note derive from an intended implementation of systems by electronic circuitry. Some alternative correctness concerns are discussed in Section 7.

In electronic circuits, there are a number of undesirable situations. Connected outputs may give rise to power shorts when they do not agree in voltage level. Connecting too many inputs together may overload the driving output. Dangling inputs may pick up noise and dangling outputs may emit spurious electromagnetic signals. We formalize these concepts as follows.

Link  $a$  is said to be **conflicting** in system  $S$  when

$$(\exists P, Q : [P, Q] \subseteq S : a \in \mathbf{o}P \cap \mathbf{o}Q),$$

that is, when it connects two output ports. Link  $a$  is called **overloaded** in  $S$  when

$$(\exists P, Q : [P, Q] \subseteq S : a \in \mathbf{i}P \cap \mathbf{i}Q),$$

that is, when it connects two input ports. System  $S$  is called **well-formed** when it has neither conflicting nor overloaded links; otherwise, it is called **malformed**. Formally,  $S$  is well-formed when

$$(\forall P, Q : [P, Q] \subseteq S : \mathbf{i}P \cap \mathbf{i}Q = \emptyset = \mathbf{o}P \cap \mathbf{o}Q).$$

System  $S$  of Example 2.1 is well-formed. Observe that malformedness may be introduced by composition of well-formed systems and that it persists under composition.

In well-formed system  $S$  containing processes  $P$  and  $Q$  there is a directed communication link labeled  $a$  from  $P$  to  $Q$  whenever port  $a$  is an output of  $P$ , i.e.  $a \in \mathbf{o}P$ , and an input of  $Q$ , i.e.  $a \in \mathbf{i}Q$  (hence,  $P \neq Q$ ). Such a link is considered **internal** to the system, in the sense that further connections to it are undesirable, that is, links are output-to-input connections. System  $S$  of Example 2.1 has four such internal links. Merging and forking of signals must be accomplished by incorporating explicit merge and fork processes. Internal links are structurally “visible” by their name, but when incorporating system behavior, communication events along internal links are intended to be hidden, that is, they are unobservable for other processes. In Section 7 we will discuss multi-point connections.

Link  $a$  is said to be **undriven** or a **dangling input** in system  $S$  (viewed as an autonomous system) when

$$(\exists P : P \in S : a \in \mathbf{i}P) \wedge (\forall Q : Q \in S : a \notin \mathbf{o}Q),$$

that is, if it connects to an input port but not to any output port. Link  $a$  is called **unterminated** or a **dangling output** in  $S$  (again, viewed as an autonomous system) when

$$(\exists P : P \in S : a \in \mathbf{o}P) \wedge (\forall Q : Q \in S : a \notin \mathbf{i}Q),$$

that is, if it connects to an output port but not to input ports. System  $S$  is called **closed** when it has no undriven and no unterminated links; otherwise, it is called **open**. Formally,  $S$  is closed when

$$(\bigcup P : P \in S : \mathbf{i}P) = (\bigcup P : P \in S : \mathbf{o}P).$$

A dangling link is considered an **external port** of the system, available for connection to the environment of  $S$ . System  $S$  of Example 2.1 has three external inputs and two external outputs. Observe that composition may introduce internal links, namely when one system has an external input for which the other system has the corresponding external output. Hence, openness may disappear under composition.

Predicate *Correct* on  $S \mathcal{Y} S$  is now defined by

$$\text{Correct}.S \equiv \text{“}S \text{ is well-formed and closed”}.$$

*Correct.S* expresses that  $S$  has output-to-input connections only (is well-formed) and has no dangling inputs or outputs (is closed). That is, each port that occurs in some process of  $S$  occurs exactly once as input and once as output in the processes of  $S$ .

**2.2 Example** Notice that both  $[\ ]$  (the empty system) and  $[(\emptyset, \emptyset)]$  (the system consisting of one “empty” process, i.e., a process without ports) are correct systems in our sense. Another example of a correct system is  $[(\{a\}, \{b\}), (\{b\}, \{a\})]$ , having two internal links labeled  $a$  and  $b$ . Examples of incorrect systems are  $[(\{a\}, \{b\}), (\emptyset, \{b\})]$  (malformed because of conflicting link  $b$ ) and  $[(\{a\}, \{b\}), (\{b\}, \emptyset)]$  (well-formed, but not closed because of dangling input  $a$ ).  $\blacksquare$

**2.3 Note** The “empty” process  $(\emptyset, \emptyset)$  is the only process that possibly occurs more than once in a well-formed system. It is not a very interesting process and could be omitted without great loss. Therefore, one could also model a system as a *set*—instead of a *bag*—of processes. There are several reasons for not doing so.

The main reason is that composition is harder to define satisfactorily for sets. Consider well-formed systems  $S = [P, Q]$  and  $T = [P, R]$  where  $P, Q,$  and  $R$  are distinct non-empty processes. Under our definition,  $S \text{ par } T$  equals  $[P, P, Q, R]$  and this composite is malformed—as intended. Simply taking set union as composition would yield  $S \text{ par } T = \{P, Q, R\}$  which could—unintentionally—be well-formed again. One way to overcome this problem is to make composition a partially defined operator, but that is exactly what we intend to avoid.

A second reason is that when process behavior is incorporated, it is well possible that structurally equal processes may have different behaviors associated with them. Therefore, if we model a system as a set of processes, then stripping away process behavior naturally yields a *bag* of (behaviorless) processes.

Finally, a third reason for using bags is that under some alternative correctness concerns (cf. Section 7), well-formed systems possibly have multiple occurrences of non-empty processes.

We have not restricted ourselves to finite systems, because we wanted to investigate some of the problems encountered with infinite networks. The restriction to countable bags is purely pragmatic. ■

### Testing, satisfaction, and equivalence

On the basis of the correctness concern and the composition operator we define (cf. [10]) **testing relation** *pass*, **satisfaction pre-order** *sat*, and **equivalence** *equ* by

$$\begin{aligned} S \text{ pass } T &\equiv \text{Correct.}(S \text{ par } T) , \\ S \text{ sat } T &\equiv (\forall U :: S \text{ pass } U \Leftarrow T \text{ pass } U) , \\ S \text{ equ } T &\equiv (\forall U :: S \text{ pass } U \equiv T \text{ pass } U) . \end{aligned}$$

Recall that  $U$  ranges over  $\mathcal{S}\mathcal{Y}\mathcal{S}$ . Relation *pass* expresses the result of testing  $S$  by (putting it in environment)  $T$ . The *pass*-set of  $S$ , denoted by  $\text{pass}.S$ , is defined by

$$\text{pass}.S = \{U : S \text{ pass } U : U\} .$$

Relation *sat* expresses when one system is at least as “good” as another in the sense of passing at least the same tests:

$$S \text{ sat } T \equiv \text{pass}.S \supseteq \text{pass}.T .$$

It acts as a refinement relation. Relation *equ* expresses that one system is as “good” as another in the sense of passing the same tests:

$$S \text{ equ } T \equiv \text{pass}.S = \text{pass}.T .$$

It may alternatively be defined by

$$S \text{ equ } T \equiv S \text{ sat } T \wedge T \text{ sat } S$$

and it is a congruence relation on  $\langle S\mathcal{Y}S; \text{par}, \text{sat} \rangle$ .

We are interested in the quotient algebra  $\langle S\mathcal{Y}S; \text{par}, \text{sat} \rangle / \text{equ}$ . In the next sections we construct an isomorphic algebra, whose objects have a mathematically simpler structure than the congruence classes.

### 3 Basic Concepts for a Fully Abstract Model

In this section we introduce some fundamental concepts for an abstract algebra.

A **link status** is a member of the six-element set  $\Lambda$  defined by

$$\Lambda = \{\perp, \blacklozenge, ?, !, \diamond, \top\}.$$

Link statuses will be used to indicate how a system “treats” each link. Their interpretation is as follows:

- $\perp$  : abused (conflicting or overloaded),
- $\blacklozenge$  : internal,
- $?$  : external input,
- $!$  : external output,
- $\diamond$  : unused,
- $\top$  : miraculous (compensation for  $\perp$ ).

The presence of  $\top$  will be motivated later in Note 4.3. A **link status function** (LSF for short) is a mapping from  $\Sigma$  to  $\Lambda$ . LSFs will be the objects of the fully abstract model of Section 5. The set of all LSFs is denoted by  $\mathcal{LSF}$ . Variables  $\alpha, \beta$ , and  $\gamma$  range over  $\Lambda$  and variables  $p, q$ , and  $r$  range over  $\mathcal{LSF}$ . For each  $\alpha \in \Lambda$  we define the constant LSF  $\alpha^\Sigma$  by

$$\alpha^\Sigma.a = \alpha.$$

$\parallel$	$\top$	$\diamond$	$!$	$?$	$\blacklozenge$	$\perp$
$\top$	$\top$	$\top$	$\top$	$\top$	$\top$	$\top$
$\diamond$	$\top$	$\diamond$	$!$	$?$	$\blacklozenge$	$\perp$
$!$	$\top$	$!$	$\perp$	$\blacklozenge$	$\perp$	$\perp$
$?$	$\top$	$?$	$\blacklozenge$	$\perp$	$\perp$	$\perp$
$\blacklozenge$	$\top$	$\blacklozenge$	$\perp$	$\perp$	$\perp$	$\perp$
$\perp$	$\top$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$

Table 1: Composition operator  $\parallel$  on  $\Lambda$

**Composition**, denoted  $\parallel$ , is a binary operator on  $\Lambda$  defined in Table 1. For example, input and output merge into internal under composition:  $? \parallel ! = \blacklozenge$ . Input composed with



input yields abuse ( $? \parallel ? = \perp$ ) because connections should be output-to-input. When modeling multi-point connections one could define composition of inputs to yield an input again (see Example 7.4).

**3.1 Note** Defining  $? \parallel ! = \diamond$  would model completely<sup>1</sup> hidden internal links. We will briefly look at that possibility in Section 7. In that case, composition would not be associative (see Example 7.3), which explains our preference for the current definition (see Property 3.2 below). ■

**3.2 Property** Composition operator  $\parallel$  on  $\Lambda$  is commutative, associative, and has  $\diamond$  as unit. Furthermore, it has  $\top$  as zero and there are no zero divisors under  $\parallel$ , that is, we have

$$\alpha \parallel \beta = \top \equiv \alpha = \top \vee \beta = \top .$$

**Proof** Commutativity, the unit and the zero, and the absence of zero divisors are readily verified in Table 1. Regarding associativity, notice that (i) the cases where  $\top$ ,  $\diamond$  (the unit), or  $\perp$  occur are trivial and (ii) any composition of three elements from  $\{\diamond, ?, !\}$  yields  $\perp$ . ■

In view of Property 3.2, we can extend  $\parallel$  to a unary operator on finite bags over  $\Lambda$  (instead of composing just two elements), for example,

$$\begin{aligned} \parallel[ ] &= \diamond , \\ \parallel[\alpha, \beta, \beta] &= \alpha \parallel \beta \parallel \beta . \end{aligned}$$

Composition is not idempotent, but we do have

$$\alpha \parallel \alpha \parallel \alpha = \alpha \parallel \alpha .$$

On account of this we can reduce multiplicities in a bag to at most two when computing  $\parallel$ , without affecting the outcome: for finite bag  $B$  over  $\Lambda$  we have

$$\begin{aligned} \parallel B &= \parallel C, \text{ where} \\ C.\alpha &= \min \{B.\alpha, 2\} . \end{aligned}$$

We define  $\parallel$  for  $\omega$ -bags over  $\Lambda$  as well, by first reducing them to a finite bag as above. We also extend  $\parallel$  to  $\mathcal{LSF}$  by pointwise application, that is,  $p \parallel q$  is defined by

$$(p \parallel q).a = p.a \parallel q.a .$$

Obviously,  $\parallel$  on  $\mathcal{LSF}$  inherits some properties from  $\parallel$  on  $\Lambda$ ; for example, it is also commutative and associative, and has  $\diamond^\Sigma$  as unit and  $\top^\Sigma$  as zero. Note, however, that it does have zero divisors. We also use  $\parallel$  as unary operator on countable bags over  $\mathcal{LSF}$  by defining

$$(\parallel B).a = \parallel[p : B.p : p.a] .$$

---

<sup>1</sup>Not only behaviorally, but also structurally.

### From processes and systems to LSFs

Before we can express the correctness predicate more simply we need to introduce a mapping from  $\mathcal{SYS}$  to  $\mathcal{LSF}$ . First, we define mapping  $\mathbf{l}: \mathcal{PROC} \rightarrow \mathcal{LSF}$  by

$$\mathbf{l}.P.a = \begin{cases} ? & \text{if } a \in \mathbf{i}P \\ ! & \text{if } a \in \mathbf{o}P \\ \diamond & \text{otherwise} \end{cases}$$

Note that this is a proper definition since  $\mathbf{i}P$  and  $\mathbf{o}P$  are disjoint. We call  $\mathbf{l}.P$  the **link status function of process**  $P$ . Since  $\mathbf{l}$  is an injective mapping, one may view it as an embedding of  $\mathcal{PROC}$  in  $\mathcal{LSF}$ . Next, we lift  $\mathbf{l}$  via  $\parallel$  to  $\mathcal{SYS}$  yielding mapping  $\mathbf{L}: \mathcal{SYS} \rightarrow \mathcal{LSF}$  defined by

$$\mathbf{L}.S = \parallel [P : S.P : \mathbf{l}.P] .$$

We call  $\mathbf{L}.S$  the **link status function of system**  $S$ . Note that this definition takes the multiplicity of each process in  $S$  into account, thus, for example,

$$\mathbf{L}.[P, Q, Q].a = \mathbf{l}.P.a \parallel \mathbf{l}.Q.a \parallel \mathbf{l}.Q.a .$$

**3.3 Property** For process  $P$  and systems  $S$  and  $T$  we have

$$\begin{aligned} (\forall a :: \mathbf{l}.P.a \notin \{\perp, \diamond, \top\}) , \\ (\forall a :: \mathbf{L}.S.a \neq \top) , \end{aligned}$$

and

$$\begin{aligned} \mathbf{L}.[ ] &= \diamond^\Sigma , \\ \mathbf{L}.[(\emptyset, \emptyset)] &= \diamond^\Sigma , \\ \mathbf{L}.[P] &= \mathbf{l}.P , \\ \mathbf{L}.(S \text{ par } T) &= \mathbf{L}.S \parallel \mathbf{L}.T . \end{aligned}$$

■

■

We can now express correctness of a system concisely in terms of its LSF.

**3.4 Theorem** For system  $S$  we have

$$\text{Correct}.S \equiv (\forall a :: \mathbf{L}.S.a \notin \{\perp, ?, !\}) .$$

**Proof** Observe that (i)  $S$  is well-formed if and only if  $\perp$  does not occur as  $\mathbf{L}.S$ -image and (ii) if  $S$  is well-formed, then  $S$  is closed if and only if  $?$  and  $!$  do not occur as  $\mathbf{L}.S$ -image. ■

## 4 Pointwise Analysis of Correctness

We will carry out a pointwise analysis of system correctness in this section, that is, by concentrating on the links individually. In the next section we will look at the global aspects of correctness again.

Inspired by Theorem 3.4, let us define correctness predicate  $Correct_\Lambda$  and testing relation  $pass_\Lambda$  on  $\Lambda$  by

$$\begin{aligned} Correct_\Lambda.\alpha &\equiv \alpha \notin \{\perp, ?, !\}, \\ \alpha \text{ pass}_\Lambda \beta &\equiv Correct_\Lambda.(\alpha \parallel \beta). \end{aligned}$$

**4.1 Property** For systems  $S$  and  $T$  we now have

$$\begin{aligned} Correct.S &\equiv (\forall a :: Correct_\Lambda.(L.S.a)), \\ S \text{ pass } T &\equiv (\forall a :: L.S.a \text{ pass}_\Lambda L.T.a). \end{aligned}$$

**Proof** Use Theorem 3.4 and Property 3.3. ■

Notice that  $pass_\Lambda$  is symmetric since  $\parallel$  is commutative. Recall the usual derived concepts:

$$\begin{aligned} pass_\Lambda.\alpha &= \{\gamma : \alpha \text{ pass}_\Lambda \gamma : \gamma\}, \\ \alpha \text{ sat}_\Lambda \beta &\equiv pass_\Lambda.\alpha \supseteq pass_\Lambda.\beta. \end{aligned}$$

The  $pass_\Lambda$ -sets are tabulated in Table 2. Notice that these  $pass_\Lambda$ -sets are unique, that

$\alpha$	$pass_\Lambda.\alpha$
$\top$	$\{\top, \diamond, !, ?, \blacklozenge, \perp\}$
$\diamond$	$\{\top, \diamond, \blacklozenge\}$
$!$	$\{\top, ?\}$
$?$	$\{\top, !\}$
$\blacklozenge$	$\{\top, \diamond\}$
$\perp$	$\{\top\}$

Table 2: The  $pass_\Lambda$ -sets and the Hasse diagram for  $\sqsubseteq_\Lambda$  (converse of  $sat_\Lambda$ )

is,  $\alpha = \beta$  if and only if  $pass_\Lambda.\alpha = pass_\Lambda.\beta$ . Hence, relation  $sat_\Lambda$  induced by  $pass_\Lambda$  is a partial order, also denoted by  $\sqsupseteq_\Lambda$ . The Hasse diagram of  $\sqsubseteq_\Lambda$  is given in Table 2. Obviously,  $\langle \Lambda; \sqsubseteq_\Lambda \rangle$  is a complete lattice. We will leave out subscript  $\Lambda$  when it is clear from the context.

From Table 2 one can readily infer a number of properties. For instance,  $Correct.\alpha$  is equivalent to  $\alpha \sqsupseteq \blacklozenge$ . Each  $pass$ -set has a minimum under  $\sqsubseteq$ . Furthermore, composition is  $\sqsubseteq$ -monotonic. It is a little harder to verify the stronger statement that composition distributes over  $\sqcap$ . Instead of exploiting our detailed knowledge about  $\Lambda$  and  $\parallel$ , we will prove these properties more generally. The reason for doing so is that one encounters a similar situation when behavior is incorporated. The general results derived here can be carried over directly.

For the remainder of this section (excepting examples) we allow ourselves to use only (i) Property 3.2 about  $\sqcup$ , (ii) the definitions of  $pass$  and  $\sqsubseteq$ , (iii) that  $\langle \Lambda; \sqsubseteq \rangle$  is a complete lattice, and (iv) that each  $pass$ -set has a minimum.

It turns out to be useful to introduce the unary operator  $\smile$ , called **reflection**, on  $\Lambda$  defined by

$$\smile\alpha = \min(pass.\alpha).$$

It is properly defined because each  $pass$ -set has a minimum. The reflection of  $\alpha$  is the “severest” test passed by  $\alpha$ .

**4.2 Property** We have  $\alpha pass \smile\alpha$ .

**Proof** From the definition of  $\smile\alpha$  follows  $\smile\alpha \in pass.\alpha$ . ■

Reflection enables us to give an alternative expression for the  $pass$  relation (Property 4.5), to give an explicit isomorphism between  $\langle \Lambda; \sqsubseteq \rangle$  and  $\langle \Lambda; \supseteq \rangle$  (Corollary 4.11), and to formulate an interesting factorization formula (Property 4.13).

**4.3 Note** Without  $\top$  we could not have defined the reflection of  $\perp$ , for in that case  $pass.\perp = \emptyset$ , whereas now we have  $pass.\perp = \{\top\}$ . This motivates the introduction of  $\top$  (but not our choice for evaluating compositions involving  $\top$ ). We will come back to the role of  $\top$  in Section 6. ■

$\alpha$	$\perp$	$\blacklozenge$	$?$	$!$	$\diamond$	$\top$
$\smile\alpha$	$\top$	$\diamond$	$!$	$?$	$\blacklozenge$	$\perp$

Table 3: Reflection operator  $\smile$  on  $\Lambda$

The effect of reflection is shown in Table 3. From this table one sees that reflection is an involution, that is, its own inverse. But we can also prove this more generally and we will not make further use of the table (again, excepting examples).

### General results for $\Lambda$

We start by observing that  $pass$ -sets are  $\sqsubseteq$ -upward closed.

**4.4 Property** We have

$$\beta \in pass.\alpha \wedge \beta \sqsubseteq \gamma \Rightarrow \gamma \in pass.\alpha.$$

**Proof** We derive

$$\begin{aligned} & \beta \in pass.\alpha \wedge \beta \sqsubseteq \gamma \\ \equiv & \quad \{ \text{symmetry of } pass \text{ and definition of } \sqsubseteq \} \\ & \alpha \in pass.\beta \wedge pass.\beta \sqsubseteq pass.\gamma \\ \Rightarrow & \quad \{ \text{set theory} \} \end{aligned}$$

$$\begin{aligned}
& \alpha \in \text{pass}.\gamma \\
\equiv & \quad \{ \text{symmetry of } \text{pass} \} \\
& \gamma \in \text{pass}.\alpha
\end{aligned}$$

■

■

Relation *pass* is expressible in terms of the order and reflection:

**4.5 Property** We have

$$\alpha \text{ pass } \beta \equiv \alpha \sqsupseteq \smile\beta .$$

**Proof** The implication from left to right follows from the definition of  $\smile\beta$  as the  $\sqsubseteq$ -minimum of *pass*. $\beta$ . The implication from right to left follows from Property 4.4 and  $\smile\beta \in \text{pass}.\beta$ . ■

We can now give a different expression for correctness:

**4.6 Property** We have

$$\text{Correct}.\alpha \equiv \alpha \sqsupseteq \smile\diamond .$$

**Proof** We derive

$$\begin{aligned}
& \text{Correct}.\alpha \\
\equiv & \quad \{ \diamond \text{ is unit of } \parallel \} \\
& \text{Correct}.\alpha \parallel \diamond \\
\equiv & \quad \{ \text{definition of } \text{pass} \} \\
& \alpha \text{ pass } \diamond \\
\equiv & \quad \{ \text{Property 4.5} \} \\
& \alpha \sqsupseteq \smile\diamond
\end{aligned}$$

■

■

**4.7 Note** The appearance of  $\diamond$  in Property 4.6 is not a coincidence as is seen in the proof:  $\diamond$  is the unit of  $\parallel$  on  $\Lambda$ . ■

**4.8 Corollary** We have

$$\alpha \parallel \beta \sqsupseteq \smile\diamond \equiv \alpha \sqsupseteq \smile\beta .$$

**Proof** Use Property 4.6, definition of *pass*, and Property 4.5. ■

Reflection reverses the order:

**4.9 Property** We have

$$\alpha \sqsubseteq \beta \equiv \smile\alpha \sqsupseteq \smile\beta .$$

**Proof** We derive

$$\begin{aligned}
& \alpha \sqsubseteq \beta \\
\equiv & \quad \{ \text{definition of } \sqsubseteq \} \\
& \text{pass}.\alpha \subseteq \text{pass}.\beta \\
\equiv & \quad \{ \text{property of min, } \text{pass}.\beta \text{ is } \sqsubseteq\text{-upward closed (Property 4.4)} \} \\
& \text{min}(\text{pass}.\alpha) \sqsupseteq \text{min}(\text{pass}.\beta) \\
\equiv & \quad \{ \text{definition of } \smile \} \\
& \smile\alpha \sqsupseteq \smile\beta
\end{aligned}$$

■

■

Reflection is an involution:

**4.10 Property** We have  $\smile\smile\alpha = \alpha$ .

**Proof** We derive

$$\begin{aligned}
& \text{true} \\
\equiv & \quad \{ \text{Property 4.2 applied to } \alpha \text{ and } \smile\alpha \} \\
& \alpha \text{ pass } \smile\alpha \wedge \smile\alpha \text{ pass } \smile\smile\alpha \\
\equiv & \quad \{ \text{Property 4.5} \} \\
& \alpha \sqsupseteq \smile\smile\alpha \wedge \smile\alpha \sqsupseteq \smile\smile\smile\alpha \\
\equiv & \quad \{ \text{Property 4.9} \} \\
& \alpha \sqsupseteq \smile\smile\alpha \wedge \alpha \sqsubseteq \smile\smile\alpha \\
\equiv & \quad \{ \text{antisymmetry of } \sqsubseteq \} \\
& \alpha = \smile\smile\alpha
\end{aligned}$$

■

■

**4.11 Corollary** Reflection is an isomorphism between  $\langle \Lambda; \sqsubseteq \rangle$  and  $\langle \Lambda; \sqsupseteq \rangle$ .

**4.12 Note** So far, we have not used associativity of  $\parallel$ .

We now prove a property that enables us to solve inequations of the form  $\alpha \parallel \beta \sqsupseteq \gamma$  for  $\alpha$ . It is called a factorization formula because it shows how  $\beta$  may be factored out of  $\gamma$ . We will come back to this important property in Section 6.

**4.13 Property (Factorization Formula)** We have

$$\alpha \parallel \beta \sqsupseteq \gamma \equiv \alpha \sqsupseteq \smile(\beta \parallel \smile\gamma).$$

**Proof** We derive

$$\alpha \parallel \beta \sqsupseteq \gamma$$

$$\begin{aligned}
&\equiv \{ \text{Corollary 4.8, using } \gamma = \smile\smile\gamma \text{ (Property 4.10)} \} \\
&\quad (\alpha \parallel \beta) \parallel \smile\gamma \supseteq \smile\Diamond \\
&\equiv \{ \text{associativity of } \parallel \} \\
&\quad \alpha \parallel (\beta \parallel \smile\gamma) \supseteq \smile\Diamond \\
&\equiv \{ \text{Corollary 4.8} \} \\
&\quad \alpha \supseteq \smile(\beta \parallel \smile\gamma)
\end{aligned}$$

■

■

The Factorization Formula is a Galois connection. It shows that for each  $\beta$  the functions  $\_ \parallel \beta$  and  $\smile(\beta \parallel \_)$  form a Galois pair.

Now we are in a position to prove

**4.14 Property** Composition operator  $\parallel$  is  $\sqcap$ -continuous (distributes over arbitrary  $\sqcap$ ), that is, for  $W \subseteq \Lambda$  we have

$$\alpha \parallel \sqcap W = \sqcap \{ \beta : \beta \in W : \alpha \parallel \beta \} .$$

**Proof** Let  $W$  be a subset of  $\Lambda$ . It suffices to prove that for all  $\gamma$  we have

$$\alpha \parallel \sqcap W \supseteq \gamma \equiv \sqcap \{ \beta : \beta \in W : \alpha \parallel \beta \} \supseteq \gamma .$$

We derive

$$\begin{aligned}
&\alpha \parallel \sqcap W \supseteq \gamma \\
&\equiv \{ \text{Factorization Formula (Property 4.13)} \} \\
&\quad \sqcap W \supseteq \smile(\alpha \parallel \smile\gamma) \\
&\equiv \{ \text{property of } \sqcap \} \\
&\quad (\forall \beta : \beta \in W : \beta \supseteq \smile(\alpha \parallel \smile\gamma)) \\
&\equiv \{ \text{Factorization Formula} \} \\
&\quad (\forall \beta : \beta \in W : \alpha \parallel \beta \supseteq \gamma) \\
&\equiv \{ \text{property of } \sqcap \} \\
&\quad \sqcap \{ \beta : \beta \in W : \alpha \parallel \beta \} \supseteq \gamma
\end{aligned}$$

■

■

**4.15 Corollary** Composition operator  $\parallel$  is  $\sqsubseteq$ -monotonic. ■

**4.16 Example** Composition operator  $\parallel$  does not distribute over  $\sqcup$ , as is seen in

$$\Diamond \parallel (? \sqcup !) = \Diamond \parallel \top = \top \neq \perp = \perp \sqcup \perp = (\Diamond \parallel ?) \sqcup (\Diamond \parallel !) .$$

Neither  $\sqcap$  nor  $\sqcup$  distributes over  $\parallel$ . Here is a counterexample for  $\sqcap$ :

$$\Diamond \sqcap (? \parallel !) = \Diamond \sqcap \Diamond = \Diamond \neq \perp = \perp \parallel \perp = (\Diamond \sqcap ?) \parallel (\Diamond \sqcap !) .$$

The same choice of operands provides a counterexample for  $\sqcup$ .

Finally, reflection does not distribute over  $\parallel$ . For if this were the case then all link statuses of the form  $\alpha \parallel \smile\alpha$  would be self-dual since  $\parallel$  is commutative and  $\smile$  is an involutions. But there are no self-dual link statuses in  $\Lambda$  at all (see Table 3). ■

### Pointwise extension to $\mathcal{LSF}$

We extend  $\sqsubseteq$  and  $\smile$  to  $\mathcal{LSF}$  by pointwise application. Hence,  $\langle \mathcal{LSF}; \sqsubseteq \rangle$  is also a complete lattice and it is isomorphic to its converse via  $\smile$ . Obviously, the Factorization Formula also applies to composition  $\parallel$  on  $\mathcal{LSF}$  and this composition is also  $\sqcap$ -continuous. We can now reformulate Property 4.1, giving alternative expressions for the correctness predicate and the testing relation on  $\mathcal{SYS}$ . Observe that in the expression for *pass* we profit again from the presence of  $\top$  in  $\Lambda$ , which made reflection possible.

**4.17 Theorem** For systems  $S$  and  $T$  we have

$$\begin{aligned} \text{Correct}.S &\equiv \mathbf{L}.S \sqsubseteq \smile \diamond^\Sigma, \\ S \text{ pass } T &\equiv \mathbf{L}.S \sqsubseteq \smile \mathbf{L}.T. \end{aligned}$$

**Proof** The second equivalence follows from Properties 4.1 and 4.5. The first equivalence follows from the second and Property 3.3 by observing

$$\text{Correct}.S \equiv S \text{ pass } [\ ] .$$

In fact, we no longer need to analyze *Correct* directly since by now we know so much about *pass*. ■

**4.18 Corollary** For systems  $S$  and  $T$  we have

$$\begin{aligned} \mathbf{L}.S \sqsubseteq \mathbf{L}.T &\Rightarrow S \text{ sat } T, \\ \mathbf{L}.S = \mathbf{L}.T &\Rightarrow S \text{ equ } T. \end{aligned}$$

■

**4.19 Note** In the proof of the preceding corollary, both transitivity and antisymmetry of  $\sqsubseteq$  are of importance. ■

On account of Corollary 4.18 and Property 3.3,  $\mathbf{L}$  may be viewed as an *equ*-respecting abstraction function, because  $\cong_{\mathbf{L}}$  is a congruence relation on  $\langle \mathcal{SYS}; \text{par}, \text{sat} \rangle$  with  $\cong_{\mathbf{L}} \subseteq \text{equ}$ . But it is not a *full* abstraction because the converse implications of the corollary do not hold in general.

**4.20 Example** Consider processes  $P$  and  $Q$ , and systems  $S$  and  $T$  defined by

$$\begin{aligned} P &= (\emptyset, \{a\}), \\ Q &= (\emptyset, \{b\}), \\ S &= [P, P], \\ T &= [Q, Q]. \end{aligned}$$

On the one hand we have  $S \text{ equ } T$  because the *pass*-sets of both  $S$  and  $T$  are empty due to output conflicts. On the other hand we have  $\mathbf{L}.S \neq \mathbf{L}.T$  because, for instance,  $\mathbf{L}.S.a = \perp$  and  $\mathbf{L}.T.a = \diamond$ . ■



## 5 Construction of Fully Abstract Model

So far we have looked at pointwise aspects of correctness only. In this section we will tie these aspects together and develop them into a fully abstract model (cf. Theorem 5.19).

Under a full abstraction, all equivalent systems should be identified, i.e., mapped into the same object. For these fully abstract objects we intend to use certain members of  $\mathcal{LSF}$ . At the end of the preceding section we observed that  $\mathbf{L}$  is an abstraction function but not a full abstraction. All malformed systems fail every test and, hence, are equivalent. Nevertheless  $\mathbf{L}$ -images of malformed systems may differ. It turns out that failure to identify malformed systems is the only deficiency that keeps  $\mathbf{L}$  from being a full abstraction.

Therefore let us consider mapping  $\llbracket \_ \rrbracket : \mathcal{SYS} \rightarrow \mathcal{LSF}$  defined by

$$\llbracket S \rrbracket = \begin{cases} \perp^\Sigma & \text{if } (\exists a :: \mathbf{L}.S.a = \perp) \\ \mathbf{L}.S & \text{otherwise} \end{cases}$$

as candidate for a full abstraction. It identifies all malformed systems by mapping them into  $\perp^\Sigma$ . We intend to take  $\sqsupseteq$  as fully abstract counterpart of *sat*. This requires us to show

$$S \text{ sat } T \equiv \llbracket S \rrbracket \sqsupseteq \llbracket T \rrbracket .$$

Furthermore, we need to define a fully abstract counterpart of *par* on the image space of  $\mathcal{SYS}$  under  $\llbracket \_ \rrbracket$ . We postpone composition for a while and concentrate on the first obligation concerning satisfaction.

### Satisfaction

The definition of  $\llbracket \_ \rrbracket$  can be rewritten in a way that facilitates generalization. We define the subset  $\mathcal{LSF}_\perp$  of  $\mathcal{LSF}$  by

$$p \in \mathcal{LSF}_\perp \equiv (\forall a, b : p.a = \perp : p.b = \perp) .$$

Notice that the defining predicate on the right-hand side is equivalent to

$$(\exists a :: p.a = \perp) \Rightarrow p = \perp^\Sigma .$$

Let  $\lfloor \_ \rfloor$  be the downward projection induced by  $\mathcal{LSF}_\perp$  in  $\langle \mathcal{LSF}, \sqsupseteq \rangle$  (see Appendix B), that is,

$$\lfloor p \rfloor = \sqcup \{ r : r \in \mathcal{LSF}_\perp \wedge r \sqsupseteq p : r \} .$$

**5.1 Property** For  $p \in \mathcal{LSF}$  we have

$$\lfloor p \rfloor = \begin{cases} \perp^\Sigma & \text{if } (\exists a :: p.a = \perp) \\ p & \text{otherwise} \end{cases}$$

**Proof** If  $(\exists a :: p.a = \perp)$  then  $\{ r : r \in \mathcal{LSF}_\perp \wedge r \sqsupseteq p : r \} = \{ \perp^\Sigma \}$  and, hence,  $\lfloor p \rfloor = \perp^\Sigma$  in this case; otherwise,  $p \in \mathcal{LSF}_\perp$  and, hence,  $\lfloor p \rfloor = p$ .  $\blacksquare$

**5.2 Corollary** For system  $S$  we have  $\llbracket S \rrbracket = \llbracket \mathbf{L}.S \rrbracket$ . ■

From now on we refer to Corollary 5.2 as definition of  $\llbracket \_ \rrbracket$ .

In this section we will work backwards, that is, we state important theorems early and in their proofs we make forward references to lemmata proved later. This way we can directly motivate our interest in certain properties of  $\mathcal{LSF}_\perp$  and  $\llbracket \_ \rrbracket$ .

Predicate *Correct* and relations *pass*, *sat*, and *equ* may be characterized in terms of  $\llbracket \_ \rrbracket$ .

**5.3 Theorem** For systems  $S$  and  $T$  we have

$$\begin{aligned} \text{Correct}.S &\equiv \llbracket S \rrbracket \sqsupseteq \sim \diamond^\Sigma, \\ S \text{ pass } T &\equiv \llbracket S \rrbracket \sqsupseteq \sim \llbracket T \rrbracket, \\ S \text{ sat } T &\equiv \llbracket S \rrbracket \sqsupseteq \llbracket T \rrbracket, \\ S \text{ equ } T &\equiv \llbracket S \rrbracket = \llbracket T \rrbracket. \end{aligned}$$

**Proof** We derive the first equivalence.

$$\begin{aligned} &\text{Correct}.S \\ \equiv & \quad \{ \text{Theorem 4.17} \} \\ &\mathbf{L}.S \sqsupseteq \sim \diamond^\Sigma \\ \equiv & \quad \{ \text{property of } \llbracket \_ \rrbracket \text{ using } \sim \diamond^\Sigma \in \mathcal{LSF}_\perp \} \\ &\llbracket \mathbf{L}.S \rrbracket \sqsupseteq \sim \diamond^\Sigma \\ \equiv & \quad \{ \text{definition of } \llbracket S \rrbracket \} \\ &\llbracket S \rrbracket \sqsupseteq \sim \diamond^\Sigma \end{aligned}$$

We derive the second equivalence.

$$\begin{aligned} &S \text{ pass } T \\ \equiv & \quad \{ \text{Theorem 4.17} \} \\ &\mathbf{L}.S \sqsupseteq \sim \mathbf{L}.T \\ \equiv & \quad \{ \text{property of } \llbracket \_ \rrbracket \text{ using } \sim \mathbf{L}.T \in \mathcal{LSF}_\perp \text{ on account of Lemma 5.4 below} \} \\ &\llbracket \mathbf{L}.S \rrbracket \sqsupseteq \sim \mathbf{L}.T \\ \equiv & \quad \{ \text{reflection reverses the order} \} \\ &\sim \llbracket \mathbf{L}.S \rrbracket \sqsubseteq \mathbf{L}.T \\ \equiv & \quad \{ \text{property of } \llbracket \_ \rrbracket \text{ using } \sim \llbracket \mathbf{L}.S \rrbracket \in \mathcal{LSF}_\perp \text{ on account of Lemma 5.5 below} \} \\ &\sim \llbracket \mathbf{L}.S \rrbracket \sqsubseteq \llbracket \mathbf{L}.T \rrbracket \\ \equiv & \quad \{ \text{reflection reverses the order and definition of } \llbracket \_ \rrbracket \} \\ &\llbracket S \rrbracket \sqsupseteq \sim \llbracket T \rrbracket \end{aligned}$$

We derive the third equivalence.

$$S \text{ sat } T$$

$$\begin{aligned}
&\equiv \{ \text{definition of } sat \} \\
&\quad (\forall U : T \text{ pass } U : S \text{ pass } U) \\
&\equiv \{ \text{second equivalence} \} \\
&\quad (\forall U : \llbracket T \rrbracket \sqsupseteq \sim \llbracket U \rrbracket : \llbracket S \rrbracket \sqsupseteq \sim \llbracket U \rrbracket) \\
&\equiv \{ \text{Note below for '}\Rightarrow\text{'}; \text{transitivity of } \sqsubseteq \text{ for '}\Leftarrow\text{' } \} \\
&\quad \llbracket S \rrbracket \sqsupseteq \llbracket T \rrbracket
\end{aligned}$$

*Note:* If  $\llbracket T \rrbracket = \perp^\Sigma$  then we are done because  $\perp^\Sigma$  is the least element in  $\mathcal{LSF}$ . If  $\llbracket T \rrbracket \neq \perp^\Sigma$  then, on account of Lemma 5.6 below, we can instantiate  $U$  such that  $\llbracket U \rrbracket = \sim \llbracket T \rrbracket$ . The desired result now is a consequence of  $\sim \sim p = p$  and reflexivity of  $\sqsubseteq$ .

The fourth equivalence follows from the third and antisymmetry of  $\sqsubseteq$ .  $\blacksquare$

On account of this theorem,  $\llbracket \_ \rrbracket$  may be viewed as a full abstraction. However, we still have three proof obligations to take care of. The first one is to show that for all systems  $T$  we have  $\sim \mathbf{L}.T \in \mathcal{LSF}_\perp$ . Let us define  $\mathcal{LSF}_\top$  as the subset of  $\mathcal{LSF}$  satisfying

$$p \in \mathcal{LSF}_\top \equiv (\forall a, b : p.a = \top : p.b = \top).$$

Note that  $\mathcal{LSF}_\perp$  and  $\mathcal{LSF}_\top$  are each other's dual in the sense that

$$p \in \mathcal{LSF}_\top \equiv \sim p \in \mathcal{LSF}_\perp.$$

We now prove

**5.4 Lemma** For system  $S$  we have  $\mathbf{L}.S \in \mathcal{LSF}_\top$ .

**Proof** From Property 3.3 follows  $\mathbf{L}.S.a \neq \top$  for any  $a$  and, hence,  $\mathbf{L}.S \in \mathcal{LSF}_\top$ .  $\blacksquare$

After dualization, the second obligation is to show  $\llbracket \mathbf{L}.S \rrbracket \in \mathcal{LSF}_\top$ . We prove

**5.5 Lemma** For LSF  $p$  in  $\mathcal{LSF}_\top$  we have  $\llbracket p \rrbracket \in \mathcal{LSF}_\top$ . Hence, for system  $S$  we have  $\llbracket S \rrbracket \in \mathcal{LSF}_\top$ .

**Proof** Assuming  $p \in \mathcal{LSF}_\top$  we derive

$$\begin{aligned}
&\llbracket p \rrbracket.a = \top \\
&\Rightarrow \{ \llbracket p \rrbracket \sqsubseteq p \text{ and } \top = \max \Lambda \} \\
&\quad p.a = \top \\
&\Rightarrow \{ p \in \mathcal{LSF}_\top \} \\
&\quad p = \top^\Sigma \\
&\Rightarrow \{ \top^\Sigma \in \mathcal{LSF}_\top, \text{property of } \llbracket \_ \rrbracket \} \\
&\quad \llbracket p \rrbracket = \top^\Sigma
\end{aligned}$$

The second part follows from the first and Lemma 5.4.  $\blacksquare$

Our third obligation is to show that for each system  $T$  with  $\llbracket T \rrbracket \neq \perp^\Sigma$  there exists a system  $U$  such that  $\llbracket U \rrbracket = \neg\llbracket T \rrbracket$ . This may be expressed concisely as

$$\neg(\llbracket \mathcal{SYS} \rrbracket \setminus \{\perp^\Sigma\}) \subseteq \llbracket \mathcal{SYS} \rrbracket,$$

where  $\neg$  and  $\llbracket \_ \rrbracket$  applied to a set of LSFs yields the set of all images of its members. In fact, we can show the following stronger result. Define  $\mathcal{LSF}'$  and  $\mathcal{LSF}''$  by

$$\begin{aligned} \mathcal{LSF}' &= (\mathcal{LSF}_\perp \cap \mathcal{LSF}_\top), \\ \mathcal{LSF}'' &= \mathcal{LSF}' \setminus \{\top^\Sigma\}. \end{aligned}$$

**5.6 Lemma** We have  $\llbracket \mathcal{SYS} \rrbracket = \mathcal{LSF}''$ .

**Proof** We infer

$$\begin{aligned} \llbracket \mathcal{SYS} \rrbracket &\subseteq \mathcal{LSF}_\top, \\ \llbracket \mathcal{SYS} \rrbracket &\subseteq \mathcal{LSF}_\perp, \\ \llbracket \mathcal{SYS} \rrbracket &\not\subseteq \top^\Sigma, \text{ and} \\ \llbracket \mathcal{SYS} \rrbracket &\supseteq \mathcal{LSF}'' . \end{aligned}$$

from Lemmata 5.4, 5.7, 5.8, and 5.9 respectively (the latter three occur below).  $\blacksquare$

**5.7 Lemma**  $\mathcal{LSF}_\top$  is  $\sqcap$ -complete in  $\langle \mathcal{LSF}; \sqsubseteq \rangle$  (cf. App. A) and, hence,  $\mathcal{LSF}_\perp$  is  $\sqcup$ -complete. Consequently,  $\llbracket \_ \rrbracket$  maps into  $\mathcal{LSF}_\perp$  and, hence,  $\llbracket \_ \rrbracket$  also.

**Proof** Let  $W$  be a subset of  $\mathcal{LSF}_\top$ . We derive for symbols  $a$  and  $b$ :

$$\begin{aligned} &(\sqcap W).a = \top \\ \equiv &\quad \{ \sqcap \text{ taken pointwise} \} \\ &\sqcap \{ p : p \in W : p.a = \top \} \\ \equiv &\quad \{ \text{property of } \sqcap \text{ using } \top = \max \Lambda \} \\ &(\forall p : p \in W : p.a = \top) \\ \Rightarrow &\quad \{ W \subseteq \mathcal{LSF}_\top \} \\ &(\forall p : p \in W : p.b = \top) \\ \equiv &\quad \{ \text{roll back} \} \\ &(\sqcap W).b = \top \end{aligned}$$

Hence,  $\sqcap W \in \mathcal{LSF}_\top$ .  $\blacksquare$

**5.8 Lemma** For LSF  $p$  we have

$$\llbracket p \rrbracket = \top^\Sigma \equiv p = \top^\Sigma.$$

For system  $S$  we have  $\llbracket S \rrbracket \neq \top^\Sigma$ .

**Proof** We derive the first part

$$\begin{aligned}
& \llbracket p \rrbracket = \top^\Sigma \\
& \equiv \{ \top^\Sigma = \max \mathcal{LSF} \} \\
& \llbracket p \rrbracket \sqsupseteq \top^\Sigma \\
& \equiv \{ \text{property of } \llbracket \_ \rrbracket, \text{ using } \top^\Sigma \in \mathcal{LSF}_\perp \} \\
& p \sqsupseteq \top^\Sigma \\
& \equiv \{ \top^\Sigma = \max \mathcal{LSF} \} \\
& p = \top^\Sigma
\end{aligned}$$

The second part now follows from the first and Property 3.3, which implies  $\mathbf{L}.S \neq \top^\Sigma$ . ■

**5.9 Lemma** For all LSFs  $p$  in  $\mathcal{LSF}''$  there exists a system  $S$  such that  $\llbracket S \rrbracket = p$ .

**Proof** We construct mapping  $inv: \mathcal{LSF}'' \rightarrow \mathcal{SYS}$  such that  $\llbracket inv.p \rrbracket = p$  for  $p \in \mathcal{LSF}''$ .

Let  $p \in \mathcal{LSF}''$ . Therefore, for all symbols  $a$ , we have  $p.a \neq \top$ . Define system  $inv.p$  as  $[P, Q]$  where processes  $P$  and  $Q$  are given by

$$\begin{aligned}
P &= (\{a : p.a = ? : a\}, \{a : p.a \in \{!, \blacklozenge, \perp\} : a\}), \\
Q &= (\{a : p.a = \blacklozenge : a\}, \{a : p.a = \perp : a\}).
\end{aligned}$$

Thus  $P$  supplies external inputs and outputs, and outputs for internal and conflicting links, whereas  $Q$  supplies inputs for internal links and outputs for conflicting links. On account of  $p \in \mathcal{LSF}''$  we have  $\mathbf{L}.P \parallel \mathbf{L}.Q = p$  and, hence,  $\mathbf{L}.(inv.p) = p$ . Since  $p \in \mathcal{LSF}_\perp$  as well, we have  $\llbracket p \rrbracket = p$  and therefore  $\llbracket inv.p \rrbracket = p$ . ■

**5.10 Note** In the above proof there are many choices for system  $S$  such that  $\llbracket S \rrbracket = p$ . If  $p$  has neither conflicting nor internal links, then process  $Q$  as defined above equals  $(\emptyset, \emptyset)$  and may be omitted; otherwise, at least two processes are required in  $S$ .

The construction given in the above proof may be applied to arbitrary LSFs, thereby extending  $inv$ . For  $p \in \mathcal{LSF}_\top \setminus \{\top^\Sigma\}$  we then have  $\mathbf{L}.(inv.p) = p$ . This is no longer the case when behavior is included. ■

**5.11 Corollary** For system  $S$  we have  $S \text{ equ } inv.\llbracket S \rrbracket$ .

**Proof** We derive

$$\begin{aligned}
& inv.\llbracket S \rrbracket \text{ equ } S \\
& \equiv \{ \text{Theorem 5.3} \} \\
& \llbracket inv.\llbracket S \rrbracket \rrbracket = \llbracket S \rrbracket \\
& \equiv \{ \text{Lemma 5.9} \} \\
& \text{true}
\end{aligned}$$

■ ■

We have now fulfilled our proof obligations concerning satisfaction. Next we consider composition.

### Composition

Given Lemma 5.9, it is straightforward to give a definition for the fully abstract counterpart of  $par$  on  $\mathcal{LSF}''$ : define binary operator  $\parallel$  on  $\mathcal{LSF}''$  by

$$p \parallel q = \llbracket inv.p \ par \ inv.q \rrbracket .$$

**5.12 Lemma**  $\langle \mathcal{LSF}''; \parallel, \sqsupseteq \rangle$  is an algebra with the same signature as  $\langle \mathcal{SYS}; par, sat \rangle$ . Furthermore, mapping  $\llbracket \_ \rrbracket$  is compatible with composition ( $par$  and  $\parallel$ ).

**Proof** All that is left to check for the first proposition is that  $\parallel$  is an operator on  $\mathcal{LSF}''$ , which it obviously is. Next we derive compatibility of  $\llbracket \_ \rrbracket$  with composition:

$$\begin{aligned} \llbracket S \ par \ T \rrbracket &= \llbracket S \rrbracket \parallel \llbracket T \rrbracket \\ &\equiv \{ \text{definition of } \parallel \} \\ \llbracket S \ par \ T \rrbracket &= \llbracket inv.\llbracket S \rrbracket \ par \ inv.\llbracket T \rrbracket \rrbracket \\ &\equiv \{ \text{Theorem 5.3} \} \\ S \ par \ T &equ \ inv.\llbracket S \rrbracket \ par \ inv.\llbracket T \rrbracket \\ &\equiv \{ \text{Corollary 5.11, } equ \text{ is congruence w.r.t. } par \} \\ &true \end{aligned}$$

■

■

The main result of this section (Theorem 5.19 below) may now be proven and the reader can skip the remainder of this subsection, which presents an alternative definition of  $\parallel$ .

Our current definition of  $\parallel$  is rather cumbersome since it works via  $\mathcal{SYS}$ . We can rewrite it as follows:

$$\begin{aligned} &\llbracket inv.p \ par \ inv.q \rrbracket \\ &= \{ \text{definition of } \llbracket \_ \rrbracket \} \\ &\llbracket \mathbf{L}.(inv.p \ par \ inv.q) \rrbracket \\ &= \{ \text{Property 3.3} \} \\ &\llbracket \mathbf{L}.(inv.p) \parallel \mathbf{L}.(inv.q) \rrbracket \\ &= \{ \text{see proof of Lemma 5.9} \} \\ &\llbracket p \parallel q \rrbracket \end{aligned}$$

For arbitrary LSFs  $p$  and  $q$ , we now define  $p \parallel q$  by

$$p \parallel q = \llbracket p \parallel q \rrbracket .$$

We need to show that the restriction of  $\parallel$  to  $\mathcal{LSF}''$  is an operator on the latter and that  $\llbracket \_ \rrbracket$  is compatible with it. We show a little more. (The role of  $\mathcal{LSF}'$  will be explained in Section 6.)

**5.13 Lemma**  $\langle \mathcal{LSF}'; \parallel, \sqsupseteq \rangle$  and  $\langle \mathcal{LSF}''; \parallel, \sqsupseteq \rangle$  are algebras with the same signature as  $\langle \mathcal{SYS}; par, sat \rangle$ .

**Proof** All we need to show is that  $\mathcal{LSF}'$  and  $\mathcal{LSF}''$  are closed under  $\parallel$ .

Let  $p$  and  $q$  be LSFs in  $\mathcal{LSF}'$  and, hence, in  $\mathcal{LSF}_\top$ . Lemma 5.14 below implies  $p \parallel q \in \mathcal{LSF}_\top$ . From Lemmata 5.5 and 5.7 we infer  $\llbracket \mathcal{LSF}_\top \rrbracket \subseteq \mathcal{LSF}'$  and, hence,  $p \parallel q \in \mathcal{LSF}'$ .

Let  $p$  and  $q$  be LSFs in  $\mathcal{LSF}''$ . In view of the preceding, all that is left to show is  $p \parallel q \neq \top^\Sigma$ , which follows from Lemmata 5.8 and 5.14.  $\blacksquare$

**5.14 Lemma** For  $B$  an countable bag over  $\mathcal{LSF}_\top$  we have  $\parallel B \in \mathcal{LSF}_\top$  and, furthermore,

$$\parallel B = \top^\Sigma \quad \equiv \quad \top^\Sigma \in B .$$

**Proof** Let  $B$  be an countable bag over  $\mathcal{LSF}_\top$ . We derive for symbol  $a$ :

$$\begin{aligned} & (\parallel B).a = \top \\ = & \quad \{ \text{definition of } \parallel \text{ for bags over } \mathcal{LSF} \} \\ & \parallel [p : B.p : p.a] = \top \\ = & \quad \{ \text{Property 3.2: } \parallel \text{ on } \Lambda \text{ has no zero divisors } \} \\ & (\exists p : p \in B : p.a = \top) \end{aligned}$$

Both propositions now follow from the fact that  $B$  is a bag over  $\mathcal{LSF}_\top$ .  $\blacksquare$

**5.15 Lemma** Mapping  $\llbracket \_ \rrbracket$  is compatible with composition (*par* and  $\parallel$ ), that is,

$$\llbracket [S \text{ par } T] \rrbracket = \llbracket [S] \rrbracket \parallel \llbracket [T] \rrbracket .$$

**Proof** For systems  $S$  and  $T$  we derive

$$\begin{aligned} & \llbracket [S \text{ par } T] \rrbracket \\ = & \quad \{ \text{definition of } \llbracket \_ \rrbracket \} \\ & \llbracket [\mathbf{L}.(S \text{ par } T)] \rrbracket \\ = & \quad \{ \text{Property 3.3} \} \\ & \llbracket [\mathbf{L}.S \parallel \mathbf{L}.T] \rrbracket \\ = & \quad \{ \text{definition of } \parallel \} \\ & \mathbf{L}.S \parallel \mathbf{L}.T \\ = & \quad \{ \text{Lemma 5.16 below, using Lemmata 5.4 and 5.5} \} \\ & \llbracket [\mathbf{L}.S] \rrbracket \parallel \llbracket [\mathbf{L}.T] \rrbracket \\ = & \quad \{ \text{definition of } \llbracket \_ \rrbracket \} \\ & \llbracket [S] \rrbracket \parallel \llbracket [T] \rrbracket \end{aligned}$$

$\blacksquare$

$\blacksquare$

**5.16 Lemma** For LSFs  $p$  and  $q$  with  $q \in \mathcal{LSF}_\top$  we have

$$p \parallel q = \llbracket [p] \rrbracket \parallel q .$$

**Proof** On account of the definition of  $\parallel$  we need to show

$$\lfloor p \parallel q \rfloor = \lfloor \lfloor p \rfloor \parallel q \rfloor.$$

For  $r \in \mathcal{LSF}_\perp$  we derive

$$\begin{aligned} & p \parallel q \supseteq r \\ \equiv & \quad \{ \text{Factorization Formula applied pointwise (Property 4.13)} \} \\ & p \supseteq \vee(q \parallel \vee r) \\ \equiv & \quad \{ \text{property of } \lfloor \_ \rfloor \text{ using } \vee(q \parallel \vee r) \in \mathcal{LSF}_\perp \text{ on account of } q \in \mathcal{LSF}_\top, r \in \\ & \quad \mathcal{LSF}_\perp, \text{ and Lemma 5.14} \} \\ & \lfloor p \rfloor \supseteq \vee(q \parallel \vee r) \\ \equiv & \quad \{ \text{Factorization Formula applied pointwise} \} \\ & \lfloor p \rfloor \parallel q \supseteq r \end{aligned}$$

Application of the definition of  $\lfloor \_ \rfloor$  completes the proof.  $\blacksquare$

**5.17 Corollary** Composition operator  $\parallel$  is associative on  $\mathcal{LSF}_\top$ .

**Proof** We derive

$$\begin{aligned} & (p \parallel q) \parallel r \\ = & \quad \{ \text{definition of } \parallel \} \\ & \lfloor \lfloor p \parallel q \rfloor \parallel r \rfloor \\ = & \quad \{ \text{Lemma 5.16 using } r \in \mathcal{LSF}_\top \} \\ & \lfloor (p \parallel q) \parallel r \rfloor \end{aligned}$$

Associativity of  $\parallel$  now follows from associativity of  $\lfloor \_ \rfloor$ .  $\blacksquare$

**5.18 Example** The condition  $q \in \mathcal{LSF}_\top$  in Lemma 5.16 and Corollary 5.17 is crucial. Consider LSFs  $p$  and  $q$  defined by

$$\begin{aligned} p.a &= \begin{cases} \perp & \text{if } a = a \\ \diamond & \text{otherwise} \end{cases} \\ q.a &= \begin{cases} \top & \text{if } a = a \\ \diamond & \text{otherwise} \end{cases} \end{aligned}$$

Then we have  $p \in \mathcal{LSF}_\top$ ,  $q \notin \mathcal{LSF}_\top$ , and

$$p \parallel q = \lfloor p \parallel q \rfloor = \lfloor q \rfloor = q \neq \perp^\Sigma = \lfloor \perp^\Sigma \parallel q \rfloor = \perp^\Sigma \parallel q = \lfloor p \rfloor \parallel q.$$

$\blacksquare$

$\blacksquare$



### Fully abstract model and summary of construction

We now have all the ingredients for a fully abstract model.

**5.19 Theorem** Algebras  $\langle \mathcal{SYS}; sat, par \rangle / equ$  and  $\langle \mathcal{LSF}''; \sqsupseteq, \parallel \rangle$  are isomorphic.

**Proof**  $\langle \mathcal{LSF}''; \parallel, \sqsupseteq \rangle$  is an algebra according to Lemma 5.12 (or 5.13). On account of Theorem 5.3 and Lemma 5.12 (or 5.15), mapping  $\llbracket \_ \rrbracket$  is a homomorphism from  $\langle \mathcal{SYS}; par, sat \rangle$  to  $\langle \mathcal{LSF}''; \parallel, \sqsupseteq \rangle$ . On account of Lemma 5.9 it is a surjection. From Theorem 5.3 also follows

$$equ = \cong_{\llbracket \_ \rrbracket}.$$

Now we can apply the Homomorphism Theorem to complete the proof.  $\blacksquare$

Let us summarize the key ingredients of the construction.

First we introduce the “mini” algebra  $\langle \Lambda; \parallel, \sqsupseteq \rangle$  and derive a number of general properties. Next we consider the function space  $\mathcal{LSF} = \Sigma \rightarrow \Lambda$  and turn it into the algebra  $\langle \mathcal{LSF}; \parallel, \sqsupseteq \rangle$  by pointwise extension. It inherits many properties from the mini algebra.  $PROC$  and  $SYS$  are mapped into  $\mathcal{LSF}$  via  $\mathbf{I}$  and  $\mathbf{L}$  respectively, translating the abstraction problem to  $\mathcal{LSF}$ .

The set  $\mathcal{LSF}$  has to be reduced. We consider the predicate

$$p.a = \top \Rightarrow p.b = \top,$$

whose universal closure over  $a$  and  $b$  defines the subset  $\mathcal{LSF}_{\top}$ .  $\mathcal{LSF}'$  is the intersection of  $\mathcal{LSF}_{\top}$  and its reflection  $\mathcal{LSF}_{\perp}$ . Downward projection  $\lfloor \_ \rfloor$  onto  $\mathcal{LSF}_{\perp}$  and full abstraction  $\llbracket \_ \rrbracket$  are defined. The following properties of  $\mathcal{LSF}_{\top}$  are proved. For process  $P$ , countable bag  $B$  over  $\mathcal{LSF}_{\top}$ , subset  $W$  of  $\mathcal{LSF}_{\top}$ , LSF  $p$  in  $\mathcal{LSF}_{\top}$ , and LSF  $q$  in  $\mathcal{LSF}' \setminus \{\top^{\Sigma}\}$  we have

- |     |   |                                 |
|-----|---|---------------------------------|
| (0) | $\top^{\Sigma} \in \mathcal{LSF}'$                                | { used in Lemmata 5.5 and 5.8 } |
| (1) | $\diamond^{\Sigma} \in \mathcal{LSF}'$                            | { used in Theorem 5.3 }         |
| (2) | $\mathbf{I}.P \in \mathcal{LSF}_{\top}$                           | { Property 3.3 }                |
| (3) | $\mathbf{L}.P \neq \top^{\Sigma}$                                 | { Property 3.3 }                |
| (4) | $\parallel B \in \mathcal{LSF}_{\top}$                            | { Lemma 5.14 }                  |
| (5) | $\parallel B = \top^{\Sigma} \equiv \top^{\Sigma} \in B$          | { Lemma 5.14 }                  |
| (6) | $\sqcap W \in \mathcal{LSF}_{\top}$                               | { Lemma 5.7 }                   |
| (7) | $\lfloor p \rfloor \in \mathcal{LSF}_{\top}$                      | { Lemma 5.5 }                   |
| (8) | $(\exists S : S \in \mathcal{SYS} : \llbracket S \rrbracket = q)$ | { Lemma 5.9 }                   |

The proofs rely on the specific form of the defining predicate for  $\mathcal{LSF}_{\top}$ . They need to be redone for each particular application, for example, when using other structural correctness concerns or when incorporating behavior. The following are general consequences

of the definitions and the above properties; they need not be redone for other applications. For LSF  $p$  in  $\mathcal{LSF}_\top$  and system  $S$  we have

- |      |  |   |
|------|--|---|
| (9)  | $\mathbf{L}.S \in \mathcal{LSF}_\top$  | { (2) and (4) above, def. $\mathbf{L}.S$ }            |
| (10) | $\mathbf{L}.S \neq \top^\Sigma$  | { (3) and (5) above, def. $\mathbf{L}.S$ }            |
| (11) | $\lfloor p \rfloor \in \mathcal{LSF}'$   | { (6) and (7) above, def. $\lfloor p \rfloor$ }       |
| (12) | $\lfloor p \rfloor = \top^\Sigma \equiv p = \top^\Sigma$                                   | { Lemma 5.8, uses (0) above }                         |
| (13) | $\llbracket S \rrbracket \in \mathcal{LSF}'$   | { (9) and (11), and def. $\llbracket S \rrbracket$ }  |
| (14) | $\llbracket S \rrbracket \neq \top^\Sigma$   | { (10) and (12), and def. $\llbracket S \rrbracket$ } |
| (15) | $\llbracket \mathcal{S} \mathcal{S} \rrbracket = \mathcal{LSF}' \setminus \{\top^\Sigma\}$ | { (8) above, (13), and (14) }                         |

The fully abstract version of composition in  $\mathcal{LSF}'$ , i.e.  $\parallel$ , is defined by

$$p \parallel q = \lfloor p \parallel q \rfloor.$$

Property (1) above is also needed to show that  $\diamond^\Sigma$  is the unit of  $\parallel$  in  $\mathcal{LSF}'$ .

## 6 Discussion of Fully Abstract Model

In this section we investigate the fully abstract model. We list a couple of important properties enjoyed by this model and we discuss the Factorization Formula and interpretations of greatest lower bounds.

### Important properties

We have attempted to restrict ourselves to properties that do not mention the internal mathematical structure of the objects in the fully abstract model ( $\mathcal{LSF}''$ ), that is, their being mappings from  $\Sigma$  to  $\Lambda$ . These properties may be used as the beginning of an axiomatic characterization. We have not looked for a complete axiomatic characterization.

LSF  $\top^\Sigma$  is included again, because the resulting algebra is much richer than  $\langle \mathcal{LSF}''; \parallel, \sqsupseteq \rangle$ ; that is, we investigate the algebra  $\langle \mathcal{LSF}'; \parallel, \sqsupseteq, \smile \rangle$ . Keep in mind, however, that  $\top^\Sigma$  has no concrete counterpart in  $\mathcal{S} \mathcal{S}$ . In this section, members of  $\mathcal{LSF}'$  will be called **abstract processes**, or processes for short.

**6.1 Property**  $\langle \mathcal{LSF}'; \sqsupseteq \rangle$  is a complete lattice. There exist unique abstract processes  $e$  and  $z$  such that for all processes  $p, q$ , and  $r$ , and all sets  $W$  of processes we have:

- |      |   |  |
|------|---|--|
| (0)  | $p \parallel q = q \parallel p$   | ( $\parallel$ is commutative)                            |
| (1)  | $(p \parallel q) \parallel r = p \parallel (q \parallel r)$                     | ( $\parallel$ is associative)                            |
| (2)  | $p \parallel e = p$   | ( $e$ is unit of $\parallel$ )                           |
| (3)  | $p \parallel q = z \equiv p = z \vee q = z$                                     | ( $z$ is zero of $\parallel$ , no zero divisors)         |
| (4)  | $p \neq z \Rightarrow p \parallel \smile z = \smile z$                          | ( $\smile z$ is pseudo zero of $\parallel$ )             |
| (5)  | $p \sqsupseteq q \equiv p \parallel \smile q \sqsupseteq \smile e$              | (relationship between $\parallel, \sqsupseteq, \smile$ ) |
| (6)  | $p \parallel \sqcap' W = \sqcap' \{q : q \in W : p \parallel q\}$               | ( $\parallel$ is $\sqcap'$ -continuous)                  |
| (7)  | $\smile \smile p = p$   | ( $\smile$ is self-inverse)                              |
| (8)  | $p \sqsupseteq q \equiv \smile p \sqsupseteq \smile q$                          | ( $\smile$ reverses $\sqsupseteq$ )                      |
| (9)  | $z = \sqcap \emptyset$  | ( $z$ is maximum)  |
| (10) | $p \parallel q \sqsupseteq r \equiv p \sqsupseteq \smile(q \parallel \smile r)$ | (Factorization Formula)                                  |

**Proof**  $\mathcal{LSF}_\perp$  is  $\sqcup$ -complete and  $\mathcal{LSF}_\top$  is  $\sqcap$ -complete in  $\langle \mathcal{LSF}; \sqsubseteq \rangle$  (Lemma 5.7). Furthermore, we have

$$(\forall p : p \in \mathcal{LSF}_\top : \lfloor p \rfloor \in \mathcal{LSF}_\top)$$

on account of Lemma 5.5. Hence (cf. theorem on complete lattices in Appendix A),  $\langle \mathcal{LSF}; \sqsubseteq \rangle$  is a complete lattice in which greatest lower bounds ( $\sqcap$ ) are computed as follows:

$$\sqcap' W = \lfloor \sqcap W \rfloor .$$

We take  $e = \diamond^\Sigma$  and  $z = \top^\Sigma$ . Unicity follows from 2 and 3. We derive  $\sqcap'$ -continuity of  $\parallel$ :

$$\begin{aligned} & p \parallel \sqcap' W \\ = & \quad \{ \text{definition of } \parallel, \text{ property of } \sqcap' \} \\ & \lfloor p \parallel \sqcap W \rfloor \\ = & \quad \{ \text{Lemma 5.16} \} \\ & \lfloor p \parallel \sqcap W \rfloor \\ = & \quad \{ \parallel \text{ is } \sqcap\text{-continuous (cf. Property 4.14)} \} \\ & \lfloor \sqcap \{ q : q \in W : p \parallel q \} \rfloor \\ = & \quad \{ \text{Lemma 6.2 below} \} \\ & \lfloor \sqcap \{ q : q \in W : \lfloor p \parallel q \rfloor \} \rfloor \\ = & \quad \{ \text{property of } \sqcap', \text{ definition of } \parallel \} \\ & \sqcap' \{ q : q \in W : p \parallel q \} \end{aligned}$$

Verification of the other properties is left as an exercise to the reader.  $\blacksquare$

**6.2 Lemma** For subset  $W$  of  $\mathcal{LSF}$  we have

$$\lfloor \sqcap W \rfloor = \sqcap \{ p : p \in W : \lfloor p \rfloor \} .$$

**Proof** For  $r \in \mathcal{LSF}_\perp$  we derive:

$$\begin{aligned} & \sqcap W \sqsupseteq r \\ \equiv & \quad \{ \text{property of } \sqcap \} \\ & (\forall p : p \in W : p \sqsupseteq r) \\ \equiv & \quad \{ \text{property of } \lfloor \_ \rfloor, \text{ using } r \in \mathcal{LSF}_\perp \} \\ & (\forall p : p \in W : \lfloor p \rfloor \sqsupseteq r) \\ \equiv & \quad \{ \text{property of } \sqcap \} \\ & \sqcap \{ p : p \in W : \lfloor p \rfloor \} \sqsupseteq r \end{aligned}$$

Application of the definition of  $\lfloor \_ \rfloor$  completes the proof.  $\blacksquare$

The properties listed above do not characterize the algebra completely (i.e. up to isomorphism), nor are they independent.

### Factorization Formula

We now briefly discuss some aspects of the Factorization Formula. One application is as follows. Given to be implemented is some specification  $r$ . The implementor decides to attempt an implementation composing a known process  $q$  (being an educated guess) with some unknown process  $p$  that still needs to be found. What would be an appropriate specification for  $p$ , given  $q$  and  $r$ ? The Factorization Formula yields  $\smile(q \parallel \smile r)$  as specification for  $p$ . In fact, this is the weakest specification for such a  $p$ . Any acceptable solution  $p$  will satisfy it. Thus, by using the Factorization Formula one does not exclude any solutions.

One may wonder what happens if the choice of  $q$  was inappropriate for the given  $r$ . An inappropriate choice of  $q$  leaves less room for choosing  $p$ . In the worst case  $q \parallel \smile r$  equals  $\perp^\Sigma$ . According to the Factorization Formula the specification for  $p$  then is  $\top^\Sigma$ . Equation  $p: p \sqsupseteq \top^\Sigma$  has  $\top^\Sigma$  as only solution. So, what happens is that this choice of  $q$  requires  $p$  to be a “miracle”. Thus, the presence of  $\top^\Sigma$  in the abstract algebra is useful in that it enables us to express unsolvability of certain equations *within the model*.

Finally, it is worth to note that choosing  $p$  equal to  $\smile(q \parallel \smile r)$  does not necessarily imply that  $p \parallel q$  then equals  $r$ . Reflection is not an inverse of composition in the usual sense. The best we can say is that  $(\smile(q \parallel \smile r)) \parallel q$  is at least as good as  $r$ , but it may be strictly better. It is quite possible that we have chosen a process  $q$  which is “too good” for the purpose, and that no  $p$  is able to “cancel” this abundance of goodness (e.g., consider the choice of  $\top^\Sigma$  for  $q$ ). In the case of system structure (not paying attention to behavior) this is actually the only way in which  $q$  may be “irreversibly” good. When behavior is incorporated there are more subtle examples (see [10]).

### Interpreting greatest lower bounds

Taking the greatest lower bound ( $\sqcap$ ) may be interpreted as “*demonic non-deterministic choice*” when occurring in an implementation or as expressing *implementation freedom* when occurring in a specification. The idea behind the first interpretation is that when confronted with  $p \sqcap q$  all one knows is that it is either  $p$  or  $q$ , but there is no way of knowing which one it is in advance. It has as formal basis:

$$p \sqcap q \sqsupseteq r \Rightarrow p \sqsupseteq r \wedge q \sqsupseteq r, \quad (1)$$

which may be paraphrased as: in order for  $p \sqcap q$  to implement  $r$ , it is necessary that both  $p$  and  $q$  individually implement  $r$ . That is, when using  $p \sqcap q$  as implementation (for  $r$ ), one had better be prepared for the worst (as chosen by the demon). By the way, we have an equivalence in (1), that is, in order for  $p \sqcap q$  to implement  $r$ , it is also sufficient that both  $p$  and  $q$  individually implement  $r$ .

**6.3 Note** The above kind of non-determinism does not have anything to do with choice present in process behavior: we are dealing with structure only here. ■

The formal basis for the second interpretation may be found in:

$$p \sqsupseteq q \sqcap r \Leftarrow p \sqsupseteq q \vee p \sqsupseteq r, \quad (2)$$

which may be paraphrased as: in order for  $p$  to satisfy specification  $q \sqcap r$ , it suffices that  $p$  satisfies either  $q$  or  $r$ . In this case, however, we do not have an equivalence. One may at times get away with an implementation  $p$  of  $q \sqcap r$  that neither implements  $q$  nor  $r$ , as is shown in the following example.

**6.4 Example** Let  $a$  and  $b$  be distinct symbols in  $\Sigma$ . Consider processes  $q$  and  $r$  given by

$$\begin{aligned} q.c &= \begin{cases} \diamond & \text{if } c = a \\ \blacklozenge & \text{otherwise} \end{cases} \\ r.c &= \begin{cases} \diamond & \text{if } c = b \\ \blacklozenge & \text{otherwise} \end{cases} \end{aligned}$$

and take  $p = \blacklozenge^\Sigma$ . Then we have  $p = q \sqcap r$ , hence  $p \sqsupseteq q \sqcap r$ , that is,  $p$  satisfies specification  $q \sqcap r$ . But we have neither  $p \sqsupseteq q$  nor  $p \sqsupseteq r$ , in fact  $p \sqsubset q$  and  $p \sqsubset r$ . ■

Likewise, one can interpret  $\sqcup$  as “angelic non-deterministic choice” when occurring in an implementation:

$$p \sqcup q \sqsupseteq r \iff p \sqsupseteq r \vee q \sqsupseteq r \quad (3)$$

or as expressing *implementation restriction* when occurring in a specification:

$$p \sqsupseteq q \sqcup r \implies p \sqsupseteq q \wedge p \sqsupseteq r. \quad (4)$$

(Of the latter the converse also holds.)

## 7 Alternative Structural Correctness Concerns

In this section we briefly look at some alternative structural correctness concerns for systems.

**7.1 Example** First we weaken the correctness concern that we have used so far in that there should be output-to-input connections only, but there may be dangling inputs or outputs (the system need not be closed) for autonomous operation. This corresponds to defining  $Correct_0$  on  $\Lambda$  by

$$Correct_0.\alpha \equiv \alpha \neq \perp.$$

The resulting  $pass_0$ -sets are tabulated in Table 4. Notice that reflection is not affected by this change in correctness concern. Also notice that  $Correct_0.\alpha$  is equivalent to  $\alpha \sqsupseteq_0 \blacklozenge$ . Finally, notice that  $\langle \Lambda; \sqsupseteq_0 \rangle$  is a complete lattice. The remainder of the analysis of this correctness concern is not carried out. ■

**7.2 Example** Now we weaken the correctness concern a little less in that there should be output-to-input connections only and no dangling inputs, but there may be dangling outputs for autonomous operation. This corresponds to defining  $Correct_1$  on  $\Lambda$  by

$$Correct_1.\alpha \equiv \alpha \notin \{\perp, ?\}.$$

$\alpha$	$pass_0.\alpha$						$\sim_0\alpha$
$\top$	$\top$	$\diamond$	$!$	$?$	$\blacklozenge$	$\perp$	$\perp$
$\diamond$	$\top$	$\diamond$	$!$	$?$	$\blacklozenge$		$\blacklozenge$
$!$	$\top$	$\diamond$		$?$			$?$
$?$	$\top$	$\diamond$	$!$				$!$
$\blacklozenge$	$\top$	$\diamond$					$\diamond$
$\perp$	$\top$						$\top$

Table 4: The  $pass_0$ -sets and duals for  $\Lambda$ , and the Hasse diagram for  $\sqsubseteq_0$ 

$\alpha$	$pass_1.\alpha$						$\sim_1\alpha$
$\top$	$\top$	$\diamond$	$!$	$?$	$\blacklozenge$	$\perp$	$\perp$
$\diamond$	$\top$	$\diamond$	$!$		$\blacklozenge$		$\blacklozenge$
$!$	$\top$	$\diamond$		$?$			$?$
$?$	$\top$		$!$				$!$
$\blacklozenge$	$\top$	$\diamond$					$\diamond$
$\perp$	$\top$						$\top$

Table 5: The  $pass_1$ -sets and duals for  $\Lambda$ , and the Hasse diagram for  $\sqsubseteq_1$ 

The resulting  $pass_1$ -sets are tabulated in Table 5. Notice that reflection is again not affected by this change in correctness concern. Also notice that again  $Correct_1.\alpha$  is equivalent to  $\alpha \sqsubseteq_1 \blacklozenge$ . Finally, notice that  $(\Lambda; \sqsubseteq_1)$  is again a complete lattice. ■

**7.3 Example** If one wants to model complete hiding of internal links—also structurally—then  $\parallel$  needs to be redefined and  $\blacklozenge$  becomes superfluous in the model. Let  $\Lambda_2$  be the five-element set defined by

$$\Lambda_2 = \{\perp, ?, !, \diamond, \top\}.$$

Composition operator  $\parallel_2$  on  $\Lambda_2$  is defined in Table 6. The main difference with  $\parallel$  is that

$\parallel_2$	$\perp$	$?$	$!$	$\diamond$	$\top$
$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\top$
$?$	$\perp$	$\perp$	$\diamond$	$?$	$\top$
$!$	$\perp$	$\diamond$	$\perp$	$!$	$\top$
$\diamond$	$\perp$	$?$	$!$	$\diamond$	$\top$
$\top$	$\top$	$\top$	$\top$	$\top$	$\top$

Table 6: Composition operator  $\parallel_2$  on  $\Lambda_2$ 

here we have  $! \parallel_2 ? = \diamond$ . Notice that  $\parallel_2$  is not associative:

$$(! \parallel_2 !) \parallel_2 ? = \perp \parallel_2 ? = \perp \neq ! \equiv ! \parallel_2 \diamond = ! \parallel_2 (! \parallel_2 ?).$$

As structural correctness concern we demand output-to-input connections and absence of dangling inputs and outputs. This is captured by defining  $Correct_2$  on  $\Lambda_2$  by

$$Correct_2.\alpha \equiv \alpha \notin \{\perp, ?, !\}.$$

The resulting  $pass_2$ -sets are tabulated in Table 7. Notice that reflection is a little bit

$\alpha$	$pass_2.\alpha$					$\sim_2\alpha$
$\top$	$\top$	$\diamond$	$!$	$?$	$\perp$	$\perp$
$\diamond$	$\top$	$\diamond$				$\diamond$
$!$	$\top$			$?$		$?$
$?$	$\top$		$!$			$!$
$\perp$	$\top$					$\top$

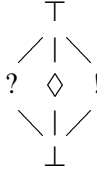


Table 7: The  $pass_2$ -sets for  $\Lambda_2$ , and the Hasse diagram for  $\sqsubseteq_2$

different this time:  $\diamond$  is self-dual. Also notice that  $Correct_2.\alpha$  is equivalent to  $\alpha \sqsubseteq_2 \diamond$ . Finally, notice that  $(\Lambda; \sqsubseteq_2)$  is again a complete lattice. In spite of the non-associativity of  $\parallel_2$  one can again prove the Factorization Formula (cf. Property 4.13). Basically, this works out because  $\parallel_2$  is associative under the restriction that at most one of each  $?$  and  $!$  occurs.

If one would change the correctness concern—still using  $\parallel_2$ —by allowing dangling inputs and/or outputs, then the reflection of  $\diamond$  turns out to be undefined. Consequently, our approach does not work directly. It may well be possible to extend  $\Lambda_2$  and  $\parallel_2$  in such a way that reflection is suitably definable. But we don't have a recipe for that. ■

**7.4 Example** In this example we look at modeling a form of multi-point connections, where each link may be driven by at most one output but may hook up to an unlimited number of inputs. Therefore,  $\blacklozenge$  (modeling a “saturated” link) is no longer needed. So we will work with  $\Lambda_2$  from the previous example. Composition operator  $\parallel_3$  on  $\Lambda_2$  is defined in Table 8. The main difference with  $\parallel$  is that here we have  $! \parallel_3 ? = !$  and  $? \parallel_3 ? = ?$ .

$\parallel_3$	$\perp$	$?$	$!$	$\diamond$	$\top$
$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\top$
$?$	$\perp$	$?$	$!$	$?$	$\top$
$!$	$\perp$	$!$	$\perp$	$!$	$\top$
$\diamond$	$\perp$	$?$	$!$	$\diamond$	$\top$
$\top$	$\top$	$\top$	$\top$	$\top$	$\top$

Table 8: Composition operator  $\parallel_3$  on  $\Lambda_2$

Composition operator  $\parallel_3$  is associative. As structural correctness concern we demand absence of output conflicts. This is captured by defining  $Correct_3$  on  $\Lambda_2$  by

$$Correct_3.\alpha \equiv \alpha \neq \perp.$$

The resulting  $pass_3$ -sets are tabulated in Table 9. Notice that the  $pass_3$ -sets are not unique:

$\alpha$	$pass_3\alpha$					$\sim_3\alpha$	
$\top$	$\top$	$\diamond$	$!$	$?$	$\perp$	$\perp$	$\top$
$\diamond$	$\top$	$\diamond$	$!$	$?$		$!$	$\diamond \cong ?$
$!$	$\top$	$\diamond$		$?$		$?, \diamond$	$ $
$?$	$\top$	$\diamond$	$!$	$?$		$!$	$ $
$\perp$	$\top$					$\top$	$ $
							$\perp$

Table 9: The  $pass_3$ -sets and duals for  $\Lambda_2$ , and the Hasse diagram for  $\sqsubseteq_3$ 

$\diamond$  and  $?$  are indistinguishable. The incorporation of behavior into the model may render them distinguishable again; for instance, only communications over input channels can cause computation interference.

When requiring absence of dangling inputs, i.e. each link should be driven by some output, then the reflection of  $\diamond$  turns out to be undefined.  $\blacksquare$

It is a nice exercise for the reader to model, for instance, the structural restriction that each link is either completely unused or exactly one output drives at most two inputs.

## 8 Conclusion

We carried out an extensive case study in the treatment of network structure using the testing paradigm. We started with a pre-abstract model  $\langle \mathcal{SYS}; par, sat \rangle$ , where  $\mathcal{SYS}$  is a set of systems (process networks),  $opc$  is a (structural) composition operator on  $\mathcal{SYS}$ , and  $sat$  is a refinement relation on  $\mathcal{SYS}$  defined in terms of testing based on a global (structural) correctness concern *Correct*. The model is pre-abstract since equivalent systems (that refine each other) are not necessarily equal.

We then developed a fully abstract model  $\langle \mathcal{LSF}''; \parallel, \sqsupseteq \rangle$  isomorphic to the pre-abstract model modulo equivalence. The objects in the fully abstract model are (link status) functions (LSFs) from the set  $\Sigma$  of link identifiers to the simple algebra  $\langle \Lambda; \parallel, \sqsupseteq \rangle$  of link statuses. The composition and order of this link status algebra are lifted to the function space by pointwise extension. Equivalent functions are identified by projection on a suitable subspace. The result is a smooth theory.

System structure is usually less subtle than system behavior. Nevertheless many of the results that we have obtained for structure carry over to behavior. The reason for simplicity in our case here is that  $\Sigma$ , the domain of LSFs, has no internal mathematical structure. In [10] behavior is incorporated and the behavioral component of a fully abstract model can be given as a function space involving  $\Sigma^*$ , that is, finite-length sequences over  $\Sigma$ , as domain. The added mathematical structure of the domain accounts for an increase in complexity, but the main ideas remain the same.

There are several noteworthy points. The set  $\mathcal{PROC}$  of processes is not so rich that every system has a singleton equivalent. The reason is that processes have no internal links. The composition operator is associative because of the way internal links are



treated. The abstract model is rich enough to allow for a unique reflection operator yielding the “severest” test passed. Note, however, that reflection is not defined for systems. Reflection is the key operator in the Factorization Formula (Properties 4.13 and 6.1), which expresses pseudo-inversion of composition in terms of composition and reflection.

It may be advisable to add a “concrete” counterpart of  $\top^\Sigma$  to the pre-abstract model. It is useful in the abstract model for several reasons. (i) It has an acceptable interpretation as “miracle”. (ii) With it the abstract algebra is isomorphic to its reflection. (iii) It allows one to express unsolvability within the algebra. (iv) Without it all sorts of preconditions are required in theorems (e.g. Factorization Formula).

Lattice theoretic concepts were used extensively. Lattice completeness played an important role and may also be useful for recursion (to define certain structurally infinite systems by an equation). Greatest lower bounds have a natural interpretation that does not involve (behavioral) choice. The framework that we set up is flexible enough to accommodate other structural correctness concerns.

## A Countable Bags

Let  $U$  be a set and let  $\omega'$  be the set of natural numbers extended with  $\omega$  (the least infinite cardinal), also known as  $\omega \cup \{\omega\}$ . A **countable bag**  $B$  **over**  $U$ , or bag for short, is a mapping from  $U$  to  $\omega'$  such that

$$\left(\sum z : z \in U : B.z\right) \leq \omega,$$

where summation is carried out in terms of cardinal arithmetic. We call  $\left(\sum z : z \in U : B.z\right)$  the **size** of the bag. For bag  $B$  and  $x \in U$ , the (cardinal) number  $B.x$  (function  $B$  applied to argument  $x$ ) is the **occurrence count** or **multiplicity** of  $x$  in  $B$ .

A finite bag, that is, a bag of finite size, can be defined by enumerating its members between square brackets as often as they should occur (the order is irrelevant). For example,  $[\ ]$  is the **empty bag** (without members), that is,  $[\ ].x = 0$  for all  $x \in U$ . And  $[a, b, b, c, b]$  is the bag in which  $a$  and  $c$  occur once, and  $b$  occurs three times.

We now introduce a special notation for **bag comprehension**, inspired by set comprehension. Recall that for set  $V$  defined by

$$V = \{z : P.z : E.z\},$$

where  $P$  is some predicate and  $E$  is some expression, we have

$$x \in V \equiv (\exists z : E.z = x : P.z).$$

This is generalized to bags as follows. We define bag  $[z : F.z : E.z]$ , where  $F$  is an  $\omega'$ -valued function and  $E$  is a  $U$ -valued function, by

$$[z : F.z : E.z].x = \left(\sum z : E.z = x : F.z\right).$$

For example, we have

$$B = [z : B.z : z].$$

Bag  $B$  over the natural numbers defined by

$$B = [n : n : n]$$

contains each natural number  $n$  exactly  $n$  times. Bag  $C$  over the natural numbers defined by

$$C = [n : 1 : n \bmod 2]$$

contains the numbers 0 and 1 exactly  $\omega$  times and the others exactly 0 times.

The following relations ( $\in$ ,  $\subseteq$ ,  $=$ ),  $\omega'$ -valued function ( $\#$ ), and operators ( $+$ ,  $\cup$ ,  $\cap$ ) are defined for bags. Let  $B$  and  $C$  be bags and let  $x \in U$ .

Name	Notation	Definition
membership	$x \in B$	$B.x > 0$
subbag	$B \subseteq C$	$(\forall z : z \in U : B.z \leq C.z)$
equality	$B = C$	$B \subseteq C \wedge C \subseteq B$
size	$\#B$	$(\sum z : z \in U : B.z)$
summation	$B + C$	$[z : B.z + C.z : z]$
union	$B \cup C$	$[z : B.z \max C.z : z]$
intersection	$B \cap C$	$[z : B.z \min C.z : z]$

*Note:*  $\sum$ ,  $+$ ,  $\max$ , and  $\min$  in the rightmost column are cardinal arithmetic operations. They are commutative, they are defined as usual for natural arguments, and for all  $n \in \omega'$  we have

$$\begin{aligned} n + \omega &= \omega, \\ n \max \omega &= \omega, \\ n \min \omega &= n. \end{aligned}$$

## B Partial Orders and Complete Lattices

In this appendix we briefly list the basic definitions and theorems from lattice theory. Some nonstandard concepts, in particular, projection on a  $\sqcup$ -complete subset of a lattice, are introduced in a separate subsection. For more details the reader is referred to [1, 2].

### Partial orders

A relation is called a **partial order** when it is reflexive, antisymmetric, and transitive. For partial order  $\sqsubseteq$  on  $V$  we call  $\langle V; \sqsubseteq \rangle$  a **poset**. The **converse** of  $\sqsubseteq$ , denoted by  $\supseteq$ , is defined by

$$u \supseteq v \equiv v \sqsubseteq u$$

for all  $u$  and  $v$  in  $V$ . *Theorem:*  $\langle V; \sqsubseteq \rangle$  is a poset if and only if  $\langle V; \supseteq \rangle$  is a poset. Partial order  $\sqsubseteq$  is called **total** when

$$u \sqsubseteq v \vee v \sqsubseteq u$$

for all  $u$  and  $v$  in  $V$ . We write  $u \sqsubset v$  for  $u \sqsubseteq v \wedge u \neq v$ .

Let  $\langle V; \sqsubseteq \rangle$  be a poset and  $U$  a subset of  $V$ . *Theorem:*  $\langle U; \sqsubseteq' \rangle$ , where  $\sqsubseteq'$  is the restriction of  $\sqsubseteq$  to  $U$ , is also a poset. *Note:* It is customary to denote the restricted order  $\sqsubseteq'$  again by  $\sqsubseteq$ .

Let  $v$  and  $w$  members of  $V$ . We call  $v$  a **lower bound** of  $U$  when

$$(\forall u : u \in U : v \sqsubseteq u) .$$

We abbreviate this to  $v \sqsubseteq U$  when confusion is unlikely (keep in mind that  $U$  may also be a member of  $V$ ). *Theorem:*  $v \sqsubseteq \emptyset$  for all  $v$ . Dually,  $v$  is called an **upper bound** of  $U$  when

$$(\forall u : u \in U : u \sqsubseteq v) ,$$

abbreviated to  $U \sqsubseteq v$ . We call  $v$  **least** in  $U$  or **minimum** of  $U$  when  $v \in U$  and  $v \sqsubseteq U$ . Dually,  $v$  is called **greatest** in  $U$  or **maximum** of  $U$  when  $v \in U$  and  $U \sqsubseteq v$ . *Theorem:* If a minimum (cq. maximum) of  $U$  exists then it is unique. The minimum (cq. maximum) of  $U$ —if it exists—is denoted by  $\min U$  (cq.  $\max U$ ). *Theorem:*  $\min \{v\} = \max \{v\} = v$ .

We call  $v$  **minimal** in  $U$  when  $v \in U$  and

$$(\forall u : u \in U \wedge u \sqsubseteq v : u = v) .$$

Dually, we call  $v$  **maximal** in  $U$  when  $v \in U$  and

$$(\forall u : u \in U \wedge u \sqsupseteq v : u = v) .$$

*Theorem:* If  $v$  is least in  $U$  then  $v$  is minimal in  $U$ . The converse does not hold generally.

We call  $v$  **greatest lower bound** of  $U$  when  $v$  is greatest in the set of lower bounds of  $U$ . If a greatest lower bound of  $U$  exists then it is unique and we denote it by  $\sqcap U$ . We also write  $v \sqcap w$  for  $\sqcap \{v, w\}$ . Dually,  $v$  is called **least upper bound** of  $U$  when  $v$  is least in the set of upper bounds of  $U$ . It is denoted by  $\sqcup U$  when it exists, and we also write  $v \sqcup w$  for  $\sqcup \{v, w\}$ . *Theorem:* If  $\min U$  (cq.  $\max U$ ) exists then  $\sqcap U$  (cq.  $\sqcup U$ ) also exists and they are equal. *Theorem:* We have

$$\begin{aligned} \sqcap \emptyset &= \max V , \\ \sqcup \emptyset &= \min V , \\ \sqcap V &= \min V , \\ \sqcup V &= \max V , \end{aligned}$$

that is, the left-hand side exists if only if the right-hand side exists, and if both exist then they are equal. *Theorem:*  $v$  is the greatest lower bound of  $U$  if and only if

$$(\forall w : w \in V : w \sqsubseteq v \equiv w \sqsubseteq U) .$$

Dually,  $v$  is the least upper bound of  $U$  if and only if

$$(\forall w : w \in V : v \sqsubseteq w \equiv U \sqsubseteq w) .$$

*Theorem:*  $\sqcap$  and  $\sqcup$  as binary operators on  $V$  are commutative, associative, and idempotent.

### Complete lattices

Let  $\langle V; \sqsubseteq \rangle$  be a poset. We call  $\langle V; \sqsubseteq \rangle$  a **complete lattice** when for every subset  $U$  of  $V$  both  $\sqcap U$  and  $\sqcup U$  exist. *Theorem:*  $\langle V; \sqsubseteq \rangle$  is a complete lattice if and only if for every subset  $U$  of  $V$ ,  $\sqcap U$  exists. *Theorem:* For finite  $V$ ,  $\langle V; \sqsubseteq \rangle$  is a complete lattice if and only if  $\sqcap \emptyset$  (i.e.  $\max V$ ) exists and for all  $v$  and  $w$  in  $V$ ,  $v \sqcap w$  exists.

Let  $\langle V_0; \sqsubseteq_0 \rangle$  and  $\langle V_1; \sqsubseteq_1 \rangle$  be complete lattices. Define relation  $\sqsubseteq$  on  $V_0 \times V_1$  by

$$(v_0, v_1) \sqsubseteq (w_0, w_1) \equiv v_0 \sqsubseteq_0 w_0 \wedge v_1 \sqsubseteq_1 w_1,$$

that is, by componentwise reduction to  $\sqsubseteq_0$  and  $\sqsubseteq_1$ . *Theorem:*  $\langle V_0 \times V_1, \sqsubseteq \rangle$  is a complete lattice, in which greatest lower bounds (cq. least upper bounds) are taken componentwise: for example,

$$\sqcap U = (\sqcap_0 \{u_0, u_1 : (u_0, u_1) \in U : u_0\}, \sqcap_1 \{u_0, u_1 : (u_0, u_1) \in U : u_1\}).$$

Let  $\langle V; \sqsubseteq \rangle$  be a complete lattice. Let  $D \rightarrow V$  be the set of all mappings from some set  $D$  to  $V$  and let  $\sqsubseteq'$  be the pointwise extension of  $\sqsubseteq$  to  $D \rightarrow V$ , that is,

$$f \sqsubseteq' g \equiv (\forall d : d \in D : f.d \sqsubseteq g.d)$$

for all  $f$  and  $g$  in  $D \rightarrow V$ . *Theorem:*  $\langle D \rightarrow V; \sqsubseteq' \rangle$  is a complete lattice, in which greatest lower bounds (cq. least upper bounds) are taken pointwise: for example,

$$(\sqcap' U).d = \sqcap \{f : f \in U : f.d\}.$$

*Note:* It is customary to denote the pointwise extended order  $\sqsubseteq'$  again by  $\sqsubseteq$ .

### Some nonstandard concepts

Let  $\langle V; \sqsubseteq \rangle$  be a complete lattice and let  $W$  be a subset of  $V$ . The function  $[-]_W : V \rightarrow V$  is defined by

$$[v]_W = \sqcup \{w : w \in W \wedge w \sqsubseteq v : w\}.$$

We leave out the subscript  $W$  when it is clear from the context. *Theorem:* For  $u$  and  $v$  in  $V$ , and  $w$  in  $W$  we have

$$\begin{aligned} [v] &\sqsubseteq v, \\ w \sqsubseteq [v] &\equiv w \sqsubseteq v, \\ [v] &= [[v]], \\ v = [v] &\Leftarrow v \in W, \\ [u] \sqsubseteq [v] &\Leftarrow u \sqsubseteq v. \end{aligned}$$

Consequently,  $[-]$  is called the **downward projection** induced by  $W$ . Dually, upward projection  $[-]_W$  is defined. Not necessarily  $[v] \in W$ . We call  $W$   **$\sqcup$ -complete** (in  $\langle V; \sqsubseteq \rangle$ ) when

$$(\forall U : U \subseteq W : \sqcup U \in W).$$

Dually,  $\sqcap$ -complete is defined. Let  $W$  be  $\sqcup$ -complete. *Theorem:* For  $v \in V$  we have  $\lfloor v \rfloor \in W$ . Hence,  $\lfloor \_ \rfloor$  projects onto  $W$ . Let  $\sqsubseteq'$  be the restriction of  $\sqsubseteq$  to  $W$ . *Theorem:*  $\langle W; \sqsubseteq' \rangle$  is a complete lattice, in which least upper bounds ( $\sqcup'$ ) and greatest lower bounds ( $\sqcap'$ ) may be computed as follows:

$$\begin{aligned}\sqcup' U &= \sqcup U, \\ \sqcap' U &= \lfloor \sqcap U \rfloor.\end{aligned}$$

Let  $X \subseteq V$  be  $\sqcap$ -complete in  $\langle V; \sqsubseteq \rangle$ . *Theorem:* If

$$(\forall x : x \in X : \lfloor x \rfloor_W \in X),$$

then  $\langle W \cap X; \sqsubseteq \rangle$  is a complete lattice, in which greatest lower bounds coincide with those taken in  $\langle W; \sqsubseteq \rangle$  and least upper bounds with those in  $\langle X; \sqsubseteq \rangle$ .

This concludes our overview of lattice theory.

## References

- [1] G. Birkhoff. *Lattice Theory*, volume 25 of *Colloquium Publications*. American Mathematical Society, Providence, RI, third edition, 1984.
- [2] B. A. Davey and H. A. Priestley. *Introduction to Lattices and Order*. Cambridge University Press, 1990.
- [3] R. de Nicola and M. Hennessy. Testing equivalences for processes. *Theoretical Comput. Sci.*, 34:83–133, 1983.
- [4] J. C. Ebergen. A formal approach to designing delay-insensitive circuits. Computing Science Notes 88/10, Dept. of Math. and C.S., Eindhoven Univ. of Technology, May 1988.
- [5] J. C. Ebergen. *Translating Programs into Delay-Insensitive Circuits*, volume 56 of *CWI Tract*. Centre for Mathematics and Computer Science, 1989.
- [6] M. Hennessy. *Algebraic Theory of Processes*. Series in Foundations of Computing. The MIT Press, Cambridge, Mass., 1988.
- [7] J. T. Udding. *Classification and Composition of Delay-Insensitive Circuits*. PhD thesis, Dept. of Math. and C.S., Eindhoven Univ. of Technology, 1984.
- [8] J. T. Udding. A formal model for defining and classifying delay-insensitive circuits. *Distributed Computing*, 1(4):197–204, 1986.
- [9] J. L. A. van de Snepscheut. *Trace Theory and VLSI Design*, volume 200 of *Lecture Notes in Computer Science*. Springer-Verlag, 1985.
- [10] T. Verhoeff. *A Theory of Delay-Insensitive Systems*. PhD thesis, Dept. of Math. and C.S., Eindhoven Univ. of Technology, May 1994.