A formal semantics of synchronous interworkings

S. Mauw[a] , M. van Wijk[b] and T. Winter[c]

[a]Dept. of Mathematics and Computing Science, Eindhoven University of Technology,
P.O. Box 513, 5600 MB Eindhoven, The Netherlands

[b]Programming Research Group, University of Amsterdam,
Kruislaan 403, 1098 SJ Amsterdam, The Netherlands

[c]Philips Research Laboratories, P.O. Box 80000, 5600 JA Eindhoven, The Netherlands

Interworkings can be considered as a synchronous variant of Message Sequence Charts. In this paper we present the formal semantics of interworkings in an algebraic framework. We concentrate on the main operators for combining interworkings: the interworking sequencing and the interworking merge.

1. INTRODUCTION

Interworkings are used in the analysis phase of the development process at PKI (Philips Kommunikations Industrie) Nürnberg for describing the message interactions between blocks. The specification of one functional block contains the most common sequences of message interactions with all the neighbour blocks it communicates with. Interworkings are a synchronous variant of message sequence charts mentioned below. They are used, for example, for the specification of radio communication systems. In close cooperation with PKI Nürnberg a project is being performed in which an Interworking Tool Set (ITS) is specified and prototyped that can process, analyse, and combine interworkings.

Message sequence charts have been used for a long time by CCITT Study Groups in their recommendations and within industry, according to different conventions and under various names. Interworkings also play an important role in other areas than the telecommunication industry. Several object-oriented methods have absorbed Message Sequence Charts (ObjectOry [5], Rumbaugh [9]).

Compared to other trace languages interworkings have the advantage of a clear graphical layout and structuring. However, interworkings are only suitable for the description of relatively small parts of the system behaviour. Therefore, in order to describe the behaviour of a system more completely, composition operations on interworkings similar to the composition operations in for example LOTOS are required. In the literature several solutions have already been proposed in this area. In [8] the syntax of message sequence charts is enhanced with the concept of "conditions" providing the means to combine message sequence charts sequentially. The sequential composition operation is only described implicitly in this document. No other composition operations on message sequence charts are defined. In [3] a system is described as a hierarchy of services increasing in detail. The lowest level of services in the hierarchy is described in terms of

message sequence charts. For the services several composition operations are defined (e.g. sequential-, parallel-composition, choice-operator). Although services can be combined the document does not discuss the related composition of the message sequence charts. In [10], a formalisation of message sequence charts is given in terms of set theory. This formalisation has the advantage that it only uses basic constructs but has the disadvantage that the parallel composition operator can not be easily expressed in the formalism.

A collection of interworkings describes the behaviour of a system on a high level of abstraction. Each interworking is a projection of a part of the communication behaviour of a system onto a set of entities. In a telecommunication context where for example SDL is used as description language, an entity may be a process or a set of processes combined into one functional block. Interworkings are event oriented instead of entity oriented and only synchronous communication actions can be modeled. This means that messages can not be delayed as in message sequence charts where communication is asynchronous. Because in practice, this does not seem to be a restriction, we concentrate ourselves in this document on the simpler model of synchronous communication and keep the asynchronous variant for further study.

For the formal definition of interworkings and the composition operators we explore techniques from process algebra [1]. Process algebra is a commonly accepted technique for the description of communicating systems. Algebraic reasoning is very suitable for formal verification activities which concern in our case the consistency checking of interworkings. Additionally, via process algebra, it is easy to generate prototypes implementing the various operators, and a link to other algebraic description methods such as LOTOS can easily be made.

In this paper we first give an introduction to interworking diagrams and operators. After that we present the formal semantics and some useful properties. We will not explain the whole interworking language but we restrict ourselves to the main operators. In [6] syntax and semantics of the complete language is given.

Acknowledgements Thanks are due to many people for giving fruitful comments on the documents. Special thanks go to Jan Bergstra for his constructive advice concerning both the documents and the project, and to Jos Baeten for his comments on process algebra. We also want to mention Lex Augusteijn for his help with regard the Elegant compiler generator tool set which we used for the development of prototype tools and finally Michel Reniers and David Riemens for their support with proving and programming.

2. INTERWORKINGS

2.1. Interworking diagrams

The basic graphical language of interworking diagrams is very simple and resembles the syntax for basic sequence charts [7]. Vertical lines are used to represent entities within a system and horizontal arrows are used to represent communications between two entities. No global time axis is assumed for one interworking. Along each vertical entity-line the time runs from top to bottom, however, no proper time scale is assumed. Events of different entities are ordered only via messages. Within interworkings, no delay between a message output and the corresponding message input is explicitly modelled. Therefore, an interworking imposes only a partial ordering on the events being contained. In figure 1,

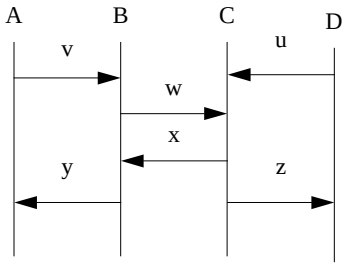

Figure 1. Example interworking diagram

the messages `u` and `v` are not related and no ordering exists between them. This also holds for the messages `y` and `z`.

A collection of interworkings describes the behaviour of a system on a high level of abstraction. Each interworking describes a common sequence of communication actions performed by the contained entities. Note that the interworking does not specify communication actions with entities that are not contained. The behaviour of a single entity is thus only partially described by the events being contained in the interworking. The actual behaviour will be an interleaving with communication events concerning other entities. Between the output of `v` and the input of `y`, entity `A` in figure 1 for example can have interactions with an entity `E` which is not in the diagram. However, it can not have a communication action with either one of the entities `B`, `C`, or `D` between the output of `v` and the input of `y`.

2.2. Interworking sequencing

An interworking diagram can be constructed from atomic communication actions and applications of the interworking sequencing operator (denoted by $\circ_{iw}$). The interworking sequencing, or simply the sequencing of two interworking diagrams is the vertical concatenation of the two diagrams (See figure 2). In this case, where the interworking

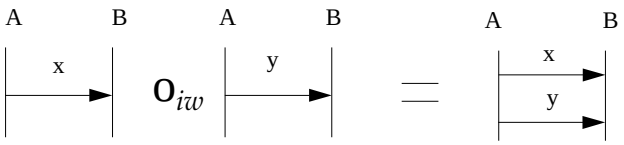

Figure 2. Sequencing

diagrams have all entities in common, the sequencing corresponds to a real sequentialisation in time. The operands need not necessarily have all entities in common. In the next example (figure 3) the two interworkings only have entity `B` in common. If interworkings contain disjoint entities, the behaviour described by means of their sequencing will in most cases not be equal to simple sequential composition of the behaviours. Events that do not involve the same entity or are not related via messages will be unordered in the sequential composition. An example of the sequencing of two interworkings that contain

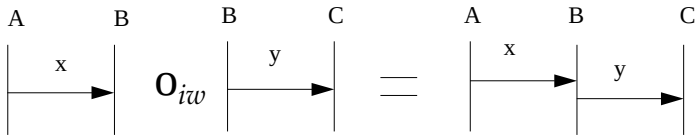

Figure 3. Sequencing interworkings with one entity in common

unrelated messages is given in figure 4. Note that the right-hand side describes a single interworking in which the vertical position of the arrows does not indicate any ordering.

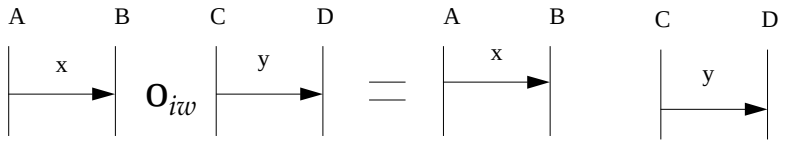

Figure 4. Sequencing interworkings with disjoint entities

2.3. Interworking merge

The interworking merge, or simply called the merge (denoted by $\|_{iw}$), of two interworkings is their interleaved composition with the restriction that the interworkings are forced to synchronise on a set of communication actions. This set consists of the communication actions concerning every pair of entities which the interworkings have in common. In

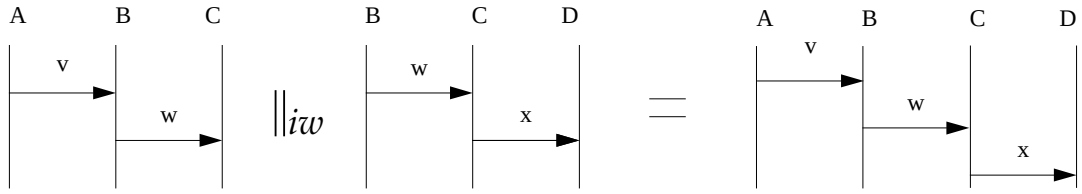

Figure 5. Merge

figure 5 two interworkings are merged, which have entities `B` and `C` in common. In those cases where the interworking diagrams describe disjoint entities, the merge is equal to real parallel composition. An example of the merge of two interworking diagrams in which disjoint entities are described is given in figure 6. Note also that the merge of these two interworking diagrams is equal to the sequencing of these diagrams (See figure 4). An interworking does not specify communication actions with entities that are not contained. The behaviour of a single entity is only partially described by the events contained in an interworking. The merge of two interworkings may return an interworking in which the sequence of events for a single entity is the result of interleaving the sequences of events in

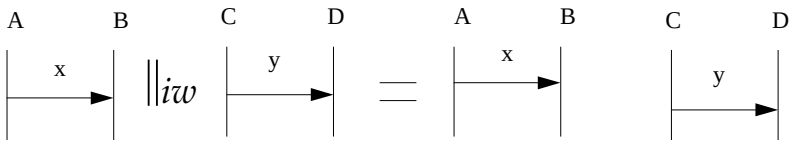

Figure 6. Merging interworkings with disjoint entities

the original interworkings. As an example, figure 7 shows the merge of two interworking diagrams that have only one entity (B) in common. For the interworking resulting from

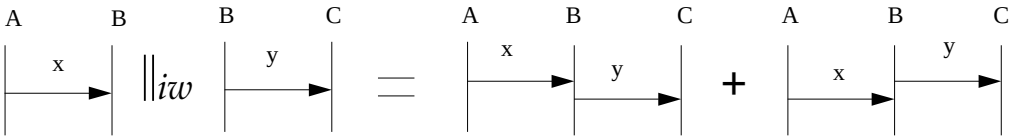

Figure 7. Merge with interleaving of events

the merge in figure 7, nothing can be said about the order of the events for entity B. The input of x can equally well occur before or after the output of y. Note that there is no single interworking diagram that describes this merge. In this case we need a collection of two different interworking diagrams.

2.4. Consistency

If two interworkings are merged, the interworkings have to synchronise on all communication actions between entities which they have in common. In case this synchronisation succeeds, we will call the two interworkings *merge-consistent*. However, if two interworkings have multiple entities in common but do not synchronise on all the communication actions between these entities then the interworkings are not merge-consistent and the merge of the interworkings will contain so called deadlock or inaction. The two interworking diagrams in figure 8 for example are not merge-consistent.

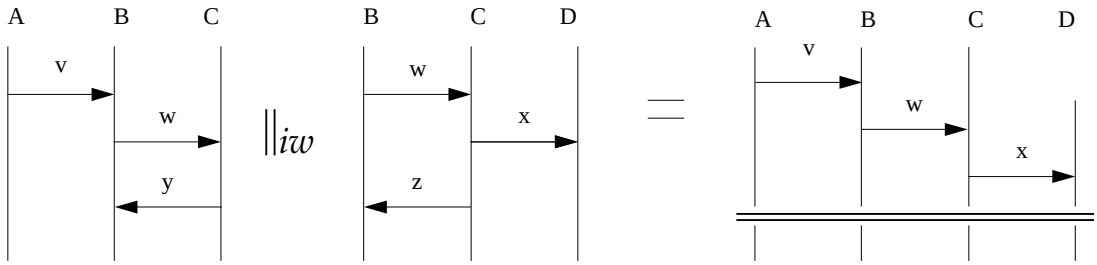

Figure 8. Inconsistent interworkings

The textual and graphical syntax for interworking diagrams does not contain a construct to represent deadlock. In figure 8, the deadlock is depicted by means of a double horizontal

line. It should be noted that deadlock is not related to any contained entity in particular. Deadlock is a property of the interworking as a whole.

3. FORMAL SEMANTICS

In this section we will first give an introduction to BPA, which is a basic theory for process description. Then we define the two interworking operators and the class of interworkings. Finally we give some useful properties of interworkings.

3.1. Basic Process Algebra

BPA (Basic Process Algebra) is an algebraic theory for the description of process behaviour [1, 2]. We consider two basic notions: *atomic actions* and *processes*. An atomic action is an indivisible unit of behaviour, such as the insertion of a coin in a coffee dispenser or the communication of some piece of information between two agents. A process is the description of the (possible) behaviour of a system. Atomic actions will be denoted by $a, b, \ldots$ and processes by $x, y, \ldots$.

Every atomic action will be considered as a process. Processes can be constructed from simpler processes with the use of two operators. The *sequential composition* of processes x and y (notation $x \cdot y$ or xy for short) is the process that first executes x, and upon completion of x starts y. The *alternative composition* of x and y (notation $x + y$) is the process that either executes x, or executes y (but not both). We will not specify how this choice is made. These operators are defined by the equations from table 1.

Table 1
Basic Process Algebra

x + y = y + x
(x + y) + z = x + (y + z)
x + x = x
(x + y)z = xz + yz
(xy)z = x(yz)

The equations state that the alternative composition is commutative, associative and idempotent, and that the sequential composition is associative only. Note that the $\cdot$ only distributes over a +-operator on the left-hand side.

In order to describe processes which may not terminate successfully, we extend BPA with features for unsuccessful termination. We use the term *deadlock* for unsuccessful termination. The special atomic action δ denotes a deadlocked process. Furthermore we introduce the encapsulation function, which renames atomic actions from a given set into δ. It is denoted by ∂_H, where H is a set of atomic actions. The equations in table 2 define deadlock and encapsulation. The first equation says that no deadlock will ever occur as long as there is an alternative that can proceed. The second equation states that after a deadlock has occurred, no other actions can possibly follow. The predicate *isdelta* determines whether a process equals δ, and the predicate *df* determines whether a process is deadlock-free or not. The equations from tables 1 and 2 define the theory $BPA_{\delta,\partial_H,df}$.

Table 2
Deadlock

$x + \delta = x$			
$\delta x = \delta$			
$\partial_H(a) = a$	if $a \notin H$	$\neg isdelta(a)$	if $a \not\equiv \delta$
$\partial_H(a) = \delta$	if $a \in H$	$isdelta(\delta)$	
$\partial_H(x + y) = \partial_H(x) + \partial_H(y)$		$isdelta(ax) = isdelta(a)$	
$\partial_H(xy) = \partial_H(x) \cdot \partial_H(y)$		$isdelta(x + y) = isdelta(x) \wedge isdelta(y)$	
$\neg df(\delta)$			
$df(a)$	if $a \not\equiv \delta$		
$df(ax) = df(x)$	if $a \not\equiv \delta$		
$df(x + y) = df(x) \wedge df(y)$	if $\neg isdelta(x)$ and $\neg isdelta(y)$		

3.2. Interworking operators

3.2.1. Basics

The collection of atomic actions can be considered as a parameter of BPA. In the setting of interworkings, we will only consider communication actions of a fixed format and some primitive actions, so we specialise the set of atomic actions.

Let EID and MID be finite sets, containing entity identifiers and message identifiers. Define the following collection of communication actions.

$$A = \{c(p, q, m) | p, q \in EID, m \in MID\}$$

Action $c(p, q, m)$ means that entity p sends message m to entity q. If we extend this collection of atomic actions with δ we get A_δ. Unless stated differently, all variables $a, b, \ldots$ range over A_δ.

Furthermore, we need an auxiliary function E, which determines the entities involved in an action. This function is defined in table 3.

Table 3
The Entity function

$E(\delta) = \emptyset$	
$E(c(p, q, m)) = \{p, q\}$	
$E(ax) = E(a) \cup E(x)$	if $a \not\equiv \delta$
$E(x + y) = E(x) \cup E(y)$	

3.2.2. Interworking Sequencing

The interworking sequencing of two processes (notation $\circ_{iw}$) is their interleaved composition with the restriction that actions involving common entities are performed first

by the left-hand process and then by the right-hand process. First we define the set of actions generated by the entities of a given process.

$\alpha_E(x) = \{a \in A | E(a) \cap E(x) \neq \emptyset\}$

The definition of the interworking sequencing in table 4 resembles the definition of the communication free merge from [1]. We use the auxiliary operators left sequencing ($\mathbf{L}\circ_{iw}$) and right sequencing ($\mathbf{R}\circ_{iw}$) which have the following intuitive meaning. The left sequencing of two processes means that the left operand is forced to do the first step. The right sequencing of two processes means that the right operand has to do the first step, but it may only execute this step if it is not blocked by some action from the left operand which involves the same entity. The encapsulation operator (∂_H) is used for blocking unwanted actions.

Table 4
Interworking Sequencing

$x \circ_{iw} y = x \mathbf{L}\circ_{iw} y \ + \ x \mathbf{R}\circ_{iw} y$	
$a \mathbf{L}\circ_{iw} x = ax$	$x \mathbf{R}\circ_{iw} a = \partial_{\alpha_{E(x)}}(a) \cdot x$
$ax \mathbf{L}\circ_{iw} y = a(x \circ_{iw} y)$	$x \mathbf{R}\circ_{iw} ay = \partial_{\alpha_{E(x)}}(a) \cdot (x \circ_{iw} y)$
$(x + y) \mathbf{L}\circ_{iw} z = x \mathbf{L}\circ_{iw} z \ + \ y \mathbf{L}\circ_{iw} z$	$x \mathbf{R}\circ_{iw} (y + z) = x \mathbf{R}\circ_{iw} y \ + \ x \mathbf{R}\circ_{iw} z$

Example 1 *Assuming that p, q, r, s and t are all different entities, we have the following derivation.*

$c(p,q,m) \circ_{iw} (c(p,r,n) \circ_{iw} c(s,t,l)) =$
$c(p,q,m) \circ_{iw} (c(p,r,n) \cdot c(s,t,l) + c(s,t,l) \cdot c(p,r,n)) =$
$c(p,q,m) \cdot (c(p,r,n) \cdot c(s,t,l) + c(s,t,l) \cdot c(p,r,n)) \ + c(s,t,l) \cdot c(p,q,m) \cdot c(p,r,n)$

3.2.3. Interworking Merge

The interworking merge of two processes (notation $\|_{iw}$) is their interleaved composition, except that the processes are forced to synchronise on a set of atomic actions (see table 5). This set consists of the actions concerning every pair of entities which the processes have in common. We define the function α_{CE}, which determines the actions performed by common entities as follows.

$\alpha_{CE}(x, y) = \{c(p,q,m) \in A | p, q \in E(x) \cap E(y)\}$

First we define the S-interworking merge (notation $\|^S_{iw}$) for a subset S of A. As in the definition of the parallel composition in ACP (the algebra of communicating processes, [1]) we need two auxiliary operators: the left interworking merge ($\lfloor\!\lfloor^S_{iw}$) and the synchronisation interworking merge ($|^S_{iw}$). The left interworking merge forces that the first step is taken from the left argument, provided that this action is not an element of the set S. The synchronisation interworking merge can only execute an action if both operands can perform this same action and this action is an element of the set S. The function γ_S determines whether two actions have to synchronise.

Table 5
Interworking Merge

$$x \parallel_{iw} y = x \parallel_{iw}^{\alpha_{CE}(x,y)} y$$

$$x \parallel_{iw}^{S} y = x \lfloor\!\lfloor_{iw}^{S} y + y \lfloor\!\lfloor_{iw}^{S} x + x|_{iw}^{S} y \qquad a|_{iw}^{S} b = \gamma_S(a,b)$$

$$a \lfloor\!\lfloor_{iw}^{S} x = \partial_S(ax) \qquad ax|_{iw}^{S} b = \gamma_S(a,b) \cdot \partial_S(x)$$

$$ax \lfloor\!\lfloor_{iw}^{S} y = \partial_S(a) \cdot (x \parallel_{iw}^{S} y) \qquad a|_{iw}^{S} bx = \gamma_S(a,b) \cdot \partial_S(x)$$

$$(x+y) \lfloor\!\lfloor_{iw}^{S} z = x \lfloor\!\lfloor_{iw}^{S} z + y \lfloor\!\lfloor_{iw}^{S} z \qquad ax|_{iw}^{S} by = \gamma_S(a,b) \cdot (x \parallel_{iw}^{S} y)$$

$$\gamma_S(a,b) = \begin{cases} a & \text{if } a = b \wedge a \in S \\ \delta & \text{otherwise} \end{cases} \qquad (x+y)|_{iw}^{S} z = x|_{iw}^{S} z + y|_{iw}^{S} z$$

$$x|_{iw}^{S} (y+z) = x|_{iw}^{S} y + x|_{iw}^{S} z$$

Example 2 *Both processes in the following merge have to synchronise on* $c(q,r,l)$.

$$c(p,q,k) \cdot c(q,r,l) \parallel_{iw} c(q,r,l) \cdot c(r,s,m) = c(p,q,k) \cdot c(q,r,l) \cdot c(r,s,m)$$

3.3. Interworkings

The theory $BPA_{\delta,\partial_H,df}$ with the two operators for interworking merge and interworking sequencing will be called BPA_{iw}. An *interworking* is a process which can be constructed only from atomic actions and applications of the interworking sequencing and the interworking merge operator. The set of interworkings is denoted by IW. The set of deadlock free interworkings is denoted by IW_{df}. A process which is only constructed of atomic actions and the interworking sequencing operator is called a locally deterministic interworking. The set of locally deterministic interworkings is called IW^{ld}. Likewise, IW_{df}^{ld} is used for deadlock free locally deterministic interworkings.

It is not always the case that the merge of two deadlock free interworkings is again deadlock free as was shown in figure 8. We say that two interworkings are *consistent* if their interworking merge is deadlock free. This notion of consistency plays an important role when developing a specification via stepwise refinement. The tool set can be used to automatically check consistency of interworkings.

Example 3 *The following example shows two interworkings which don't agree on the communications with respect to entities* q *and* r.

$$c(p,q,k) \cdot c(q,r,l) \parallel_{iw} c(s,t,m) \cdot c(r,q,n) =$$
$$c(p,q,k) \cdot c(s,t,m) \cdot \delta + c(s,t,m) \cdot c(p,q,k) \cdot \delta$$

3.4. Properties

In this section we present some useful properties about BPA_{iw}. In general x, y and z are finite process expressions in BPA_{iw} and S is a set of actions, The use of the first proposition is that most theoretical results for BPA can be transferred to BPA_{iw}.

Proposition 1 (Elimination)
(1) Every process in BPA_{iw} *is equal to a process which has one of the following forms:*

$$\delta,\ a,\ a \cdot x,\ x + y$$

(2) BPA_{iw} is a conservative extension of BPA.

Proof Term rewrite analysis as in [1]. □

The operators $\|^S_{iw}$ and $\|_{iw}$ are commutative, while the arguments of the $\circ_{iw}$ operator may only be interchanged if they have no entities in common. The first rule, for example, can be used to derive that in figure 6 x and y are not ordered.

Proposition 2

$$E(x) \cap E(y) = \emptyset \Rightarrow \quad x \circ_{iw} y = y \circ_{iw} x \tag{1}$$
$$x \|^S_{iw} y = y \|^S_{iw} x \tag{2}$$
$$x \|_{iw} y = y \|_{iw} x \tag{3}$$

Proof For (1) use $E(x) \cap E(y) = \emptyset \Rightarrow x \,\mathbf{L}\circ_{iw}\, y = y \,\mathbf{R}\circ_{iw}\, x$
(2) and (3) are clear by symmetry of the definitions. □

The operators $\|^S_{iw}$ and $\circ_{iw}$ are associative, while associativity for the $\|_{iw}$ operator only holds for pairwise consistent interworkings. The proof of proposition 5 consists of a complex simultaneous induction.

Proposition 3 $(x \circ_{iw} y) \circ_{iw} z = x \circ_{iw} (y \circ_{iw} z)$

Proof Simultaneous induction on the total number of symbols in x, y and z of the following four propositions.

$(x \,\mathbf{L}\circ_{iw}\, y) \,\mathbf{L}\circ_{iw}\, z = x \,\mathbf{L}\circ_{iw}\, (y \circ_{iw} z)$
$(x \,\mathbf{R}\circ_{iw}\, y) \,\mathbf{L}\circ_{iw}\, z = x \,\mathbf{R}\circ_{iw}\, (y \,\mathbf{L}\circ_{iw}\, z)$
$(x \circ_{iw} y) \,\mathbf{R}\circ_{iw}\, z = x \,\mathbf{R}\circ_{iw}\, (y \,\mathbf{R}\circ_{iw}\, z)$
$(x \circ_{iw} y) \circ_{iw} z = x \circ_{iw} (y \circ_{iw} z)$ □

Proposition 4 $(x \|^S_{iw} y) \|^S_{iw} z = x \|^S_{iw} (y \|^S_{iw} z)$

Proof Simultaneous induction on the total number of symbols in x, y and z of the following four propositions.

$(x \lfloor\!\lfloor^S_{iw} y) \lfloor\!\lfloor^S_{iw} z = x \lfloor\!\lfloor^S_{iw} (y \|^S_{iw} z)$
$(x|^S_{iw} y)|^S_{iw} z = x|^S_{iw} (y|^S_{iw} z)$
$(x|^S_{iw} y) \lfloor\!\lfloor^S_{iw} z = x|^S_{iw} (y \lfloor\!\lfloor^S_{iw} z)$
$(x \|^S_{iw} y) \|^S_{iw} z = x \|^S_{iw} (y \|^S_{iw} z)$ □

Proposition 5 *Let x, y and z be interworkings, let x be of the form x', $x' \cdot \delta$ or δ, y be of the form y', $y' \cdot \delta$ or δ and z be of the form z', $z' \cdot \delta$ or δ, such that x', y' and z' are pairwise consistent interworkings, then we have*

$(x \|_{iw} y) \|_{iw} z = x \|_{iw} (y \|_{iw} z)$

Example 4 *The fact that this proposition requires that the interworkings have to be consistent is shown by the following calculations.*

$$c(p,q,m) \parallel_{iw} (c(p,q,n) \parallel_{iw} c(p,q,n)) = c(p,q,m) \parallel_{iw} c(p,q,n) = \delta$$
$$(c(p,q,m) \parallel_{iw} c(p,q,n)) \parallel_{iw} c(p,q,n) = \delta \parallel_{iw} c(p,q,n) = c(p,q,n) \cdot \delta$$

Proposition 6

$$x \circ_{iw} \delta = x \cdot \delta \quad (1)$$
$$\delta \circ_{iw} x = x \cdot \delta \quad (2)$$
$$x \parallel_{iw} \delta = x \cdot \delta \quad (3)$$

Proof (1) Simultaneous induction with $x \mathbf{L}\circ_{iw} \delta = x \cdot \delta$

(2) follows from (1) and proposition 2.

(3) Simultaneous induction of $x \lfloor\!\lfloor^{\emptyset}_{iw} \delta = x \cdot \delta$ and $x \parallel^{\emptyset}_{iw} \delta = x \cdot \delta$. □

The following proposition implies that a deadlock in an interworking is *global* and will be reached only if all admissible actions are executed. It can be proven with simultaneous induction.

Proposition 7 *Every interworking is deadlock free, equal to δ or of the form $p \cdot \delta$ where p is a deadlock free interworking.*

For the following properties we need the notion of a *trace* of a process. A trace is an element of $A^+ \cup A^* \ \delta \cup \{\delta\}$. So a trace is a non-empty sequence of atomic actions, with the restriction that only the last action may be a δ. Concatenation of traces is denoted by placing the traces next to each other. We define the set of traces of a process as follows.

$$tr(a) = \{a\}$$
$$tr(ax) = \{at | t \in tr(x)\} \quad \text{if } a \neq \delta$$
$$tr(x+y) = tr(x) \cup tr(y) \quad \text{if } x \neq \delta \wedge y \neq \delta$$

A process x is *deterministic* if it is of the following form.

$$\sum_{i \in I} a_i \cdot x_i + \sum_{j \in J} a_j$$

where we require that all a_k are different for $k \in I \cup J$ and all x_i are deterministic. It follows directly from the definitions of the interworking operators that every interworking is deterministic. This implies that every interworking is completely determined by its trace set. See [4] for a proof that this proposition holds for all deterministic processes.

Proposition 8 *For interworkings x and y*

$$tr(x) = tr(y) \Rightarrow x = y$$

By $\tilde{x}$ we denote the process x where all sequential operators are replaced by interworking sequencing operators.

$\tilde{a} = a$
$\widetilde{a \cdot x} = a \circ_{iw} \tilde{x}$
$\widetilde{x + y} = \tilde{x} + \tilde{y}$

The class IW is closed under this operation. Moreover we have $x = \tilde{x}$ for every interworking x.

This operation can also be defined on traces.

$\tilde{a} = a$
$\widetilde{ax} = a \circ_{iw} \tilde{x} \quad$ if $a \neq \delta$

We conclude with a proposition which states that every interworking is completely determined by a set of locally deterministic interworkings. The application of this proposition is in the fact that every interworking, in particular interworkings as in figure 7 can be represented by a set of interworking diagrams. So the user of the tool set does not have to know about the $\cdot$ and the $+$-operator. Input and output consist of expressions containing only sequencing and merge operators.

Proposition 9 *For every interworking x*

$$tr(x) = \bigcup_{t \in tr(x)} tr(\tilde{t})$$

4. CONCLUSIONS

The problem of the so called horizontal and vertical composition of interaction diagrams as used for Interworkings or Message Sequence Charts has been tackled successfully by introducing the sequencing and the merge operator. Apart from the clear semantics, the algebraic approach shows some more advantages. The main point is that it enables the formal algebraic verification of useful properties such as consistency and refinement (which is the implementation relation between two interworking specifications as described in [6]). The second point is that, when interpreted as a term rewriting system, the equations provide a straightforward prototyping strategy for tool builders. This easy prototyping has helped in evaluating the use of interworkings in the aforementioned Interworking project.

In [6] we give a semantics of the full Interworking language, which also includes macros, timers and refinement. Research is going on to provide other features from the field of MSCs with algebraic semantics. These are asynchronous communication, message overtaking and conditions. Furthermore, the possibility is being studied to relax the requirement that in a merge the components have to synchronise on all common actions.

REFERENCES

1. J.C.M. Baeten & W.P. Weijland, Process algebra, Cambridge Tracts in Theoretical Computer Science 18, Cambridge University Press, ISBN 0 521 40043 0, 1990.

2. J.A. Bergstra & J.W. Klop, Process algebra for synchronous communication, Inf. & Control 60, pp. 109-137, 1984.
3. V. Encontre, e.a., Combining Services, Message Sequence Charts and SDL: Formalism, Method and Tools, SDL '91, Evolving Methods, Elsevier Science Publishers, ISBN 0 444 88976 0, 1991.
4. J. Engelfriet, Determinacy → (observation equivalence = trace equivalence), TCS 36(1), pp.21-25, 1985.
5. I. Jacobson, a.o., Object-Oriented Software Engineering, A Use Case Driven Approach, Addison Wesley, ISBN 0 201 54435 0, 1992.
6. S. Mauw, M. van Wijk & T. Winter, Syntax and semantics of synchronous interworkings, Philips IST technical report RWB-508-RE-92436, 1992.
7. E. Rudolph, Syntax and Semantics of Basic Sequence Charts, Contribution to the Study Group X, WP X/3, Q8, CCITT meeting, Geneva 1992.
8. E. Rudolph, P. Graubmann, J. Grabowski, Towards an SDL-Design-Methodology Using Sequence Charts Segments, SDL '91, Evolving Methods, page 237-252, Elsevier Science Publishers, ISBN 0 444 88976 0, 1991.
9. J. Rumbaugh, a.o., Object-Oriented Modeling and Design, Prentice Hall International, ISBN 13 630054 5, 1991.
10. P.A.J. Tilanus, A formalisation of Message Sequence Charts, SDL '91, Evolving Methods, page 273-288, Elsevier Science Publishers, ISBN 0 444 88976 0, 1991.