Empty interworkings and refinement semantics of interworkings revised

Citation for published version (APA):

Document status and date:
Published: 01/01/1995

Document Version:
Publisher’s PDF, also known as Version of Record (includes final page, issue and volume numbers)

Please check the document version of this publication:
• A submitted manuscript is the version of the article upon submission and before peer-review. There can be important differences between the submitted version and the official published version of record. People interested in the research are advised to contact the author for the final version of the publication, or visit the DOI to the publisher’s website.
• The final author version and the galley proof are versions of the publication after peer review.
• The final published version features the final layout of the paper including the volume, issue and page numbers.

Link to publication

General rights
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

• Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
• You may not further distribute the material or use it for any profit-making activity or commercial gain
• You may freely distribute the URL identifying the publication in the public portal.

If the publication is distributed under the terms of Article 25fa of the Dutch Copyright Act, indicated by the “Taverne” license above, please follow below link for the End User Agreement:
www.tue.nl/taverne

Take down policy
If you believe that this document breaches copyright please contact us at:
openaccess@tue.nl
providing details and we will investigate your claim.

Download date: 03. May. 2019
Empty Interworkings and Refinement
Semantics of Interworkings Revised

by

S. Mauw and M.A. Reniers

95/12
Empty Interworkings and Refinement
Semantics of Interworkings Revised

S. Mauw, M.A. Reniers*
Department of Mathematics and Computing Science,
Eindhoven University of Technology,
P.O. Box 513, NL–5600 MB Eindhoven, The Netherlands.
sjouke@win.tue.nl, michelr@win.tue.nl

Abstract
The semantics for Interworkings from [MvWW93] does not give a proper meaning to empty
terms and empty Interworkings. Furthermore, the process algebra considered has to be
extended in order to define refinement of Interworkings. For these purposes we give a revision
and extension of the semantics.

1 Introduction
Interworkings are used for the graphical presentation of system traces. An Interworking describes
the communication behaviour of system components. Interworkings are similar to Message Se­
quence Charts ([IT94]), which are standardized by the International Telecommunication Union
(ITU). The main difference is that Interworkings describe synchronous communication, whereas
Message Sequence Charts describe asynchronous communication.

A first proposal for the syntax and semantics of Interworkings is given in [MvWW93] and
[MW93] contains a description of a tool set for Interworkings. The semantics are given via a
translation into process algebra ([BK84a, BK84b, BV95, BW90]). Communications are translated
into atomic actions and two composition operators are defined: the interworking sequencing (qw)
for vertical composition and the interworking merge (||iw) for horizontal composition. In [vBG95]
Van den Brink and Griffioen describe an extension of Interworkings with discrete absolute time
features by labelling actions with a time stamp and by labelling actions with a discrete time
interval.

An Interworking consists of entities, represented by vertical axes, and messages, represented
by horizontal arrows. The intuition behind the semantics is as follows. If two messages share
an entity, the highest drawn message is executed first. Two messages which are not ordered in
this way directly (i.e. via a shared entity) or indirectly (via a number of communications) may be
executed in any order (see e.g. Figure 2D in which only one execution order is allowed: m1, m2,
m3). This is expressed formally in the definition of the interworking sequencing operator. The
interworking merge operator is explained below.

Although the semantics in [MvWW93] are consistent, we are not completely satisfied with it,
especially with respect to empty entities and empty Interworkings. These notions imply introduc­
tion of the so-called empty process into the process algebra. This extended process algebra is also
needed if we define refinement as introduced in [MvWW92]. The problem encountered in the se­
mantics of [MvWW93] with respect to empty entities is the following. Consider the Interworkings
from Figure 1. The difference between the Interworkings A and B is that B contains an entity
s while A does not. In fact Interworking B specifies that there can be no communication with
entity s, whereas Interworking A does not say anything about the interaction with s.

*Part of this research is funded by Philips Research Laboratories Eindhoven.
If we merge Interworkings A and C (from Figure 1), we first calculate the set of common entities of A and C, which is \( \{q, r\} \). The result is such that all communications between entities within this set are present in both A and C. The result is Interworking D shown in Figure 2.

In Interworking B entity s is present, although not participating in any communication action. If we merge Interworkings B and C, the common entities from B and C are \( \{q, r, s\} \). Thus B and C have to comply also with respect to the communication of m3 from r to s. Since B does not have this communication, a deadlock results as depicted by two horizontal bars in the Interworking E from Figure 2.

We conclude that intuitively the merge of A and C is different from the merge of B and C. In the semantics of [MvWW93], however, A and B have exactly the same interpretation. The empty entity s is simply neglected, so \( A \equiv_{\text{hu}} C \) and \( B \equiv_{\text{hu}} C \) are equal. We solve this difference in intuition and semantics by enriching the process which is the interpretation of a given Interworking with explicit information about the entities which are present.

The second reason for revising the semantics of Interworkings is that we give a formal definition of the notion of refinement. An Interworking is a refinement of another Interworking if they have the same behaviour after aggregation of a number of entities into one single entity. This implementation relation is very useful for expressing levels of abstraction and thus allows for a top-down specification style.

This paper is organized as follows. In Section 2 we give a formal definition of the interworking operators. Section 3 contains several properties of the interworking sequencing and interworking merge operators. The Interworking refinement is defined in Section 4.

Acknowledgements We would like to thank Thijs Winter (Philips Hilversum) and Mark van Wijk for cooperating on a preliminary, although never published, version of Interworkings with refinement. Furthermore, we thank Jos Baeten, Twan Basten and Hans Mulder (all Eindhoven University of Technology) for their criticism. Their comments have been very helpful in obtaining the results we did. The anonymous reviewers and Jan Gerben Wijnstra (Philips Research Laboratories Eindhoven) are acknowledged for their comments on a preliminary version of this paper.
We are grateful to Loe Feijs (Philips Research Laboratories Eindhoven) for his comments on the refinement of Interworkings.

2 Process Algebra for Interworkings

In this section we will extend the process algebra $BPA_{I}(A, EID, E)$ from [MvW93] with the empty process $(e)$. The parameters $A$, $EID$ and $E$ are the set of atomic actions, a universe of entity identifiers and a mapping from atomic actions to entity identifiers, respectively. We start by giving the process algebra $BPA_{I}(A)$ from [BW90, BV95]. This process algebra is extended to the process algebra $IWD_{I}(A, EID, E)$, i.e., *Process Algebra for Interworking Diagrams*, with the operator interworking sequencing ($q_w$) and some auxiliary operators ($L_{tw}, R_{tw}$, and $\sqrt{\cdot}$). Then we extend $IWD_{I}(A, EID, E)$ with the $E$-interworking merge ($||_{E}$) and some auxiliary operators ($L_{tw}^{E}, R_{tw}^{E}$). The resulting process algebra is called $IW_{I}(A, EID, E)$. Finally, we extend the processes representing Interworkings with an additional label representing the entities from which the Interworking is built. This process algebra will be called $IWE_{I}(A, EID, E)$, i.e. *Process Algebra for Interworkings with (empty) entities*. We give for each of the process algebras a structured operational semantics in the style of Plotkin [Plo81, Plo83].

In the case of Interworkings, the three parameters of the algebraic theories are instantiated as follows: $A = \{c(p,q,m) | p,q \in EID, m \in MID\}$, where MID is some set of message identifiers, EID is some set of entity identifiers and $E$ is a function which associates to each atomic action from $A$ a set of entity identifiers: $E(c(p,q,m)) = \{p,q\}$. In fact, with Interworkings, there are two parameters: the set of entity identifiers $EID$ and the set of message identifiers $MID$, and a constructor function for the atomic actions $c : EID \times EID \times MID \rightarrow A$. We have chosen for the approach with three parameters to cover applications where an entity function must be defined explicitly, because it can not be obtained from the atomic actions.

2.1 Basic Process Algebra with Deadlock and Empty Process

We will give a brief introduction to the process algebra $BPA_{I}(A)$ [BV95, BW90]. This process algebra will be our starting point towards the more complex algebras which are introduced in the following sections. The parameter $A$ of the process algebra represents the set of atomic actions. Besides the atomic actions from the set $A$ the process algebra has the additional constants $\delta$ and $e$, which represent deadlock and the empty process respectively. The process deadlock is incapable of executing any actions and can moreover not terminate successfully. The empty process can also execute no actions, but it terminates successfully. The set of all constants of the process algebra is denoted by $A_{c,e}$.

From these constants more complex processes can be built by using the operators $+$ and $\cdot$. The $+$ is called alternative composition and $\cdot$ is called sequential composition. The process $x + y$ can execute either process $x$ or process $y$, but not both. The process $x \cdot y$ starts executing process $x$, and upon termination thereof starts the execution of process $y$. These operators are axiomatized by the axioms from Table 1. In these axioms the variables $x$, $y$ and $z$ denote arbitrary processes.

<table>
<thead>
<tr>
<th>Axiom</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x + y = y + z$</td>
<td>A1 $\delta + z = x$ A6</td>
</tr>
<tr>
<td>$(x + y) + z = x + (y + z)$</td>
<td>A2 $\delta \cdot z = \delta$ A7</td>
</tr>
<tr>
<td>$x + z = x$</td>
<td>A3 $x \cdot e = x$ A8</td>
</tr>
<tr>
<td>$(x + y) \cdot z = x \cdot (y + z)$</td>
<td>A4 $e \cdot x = x$ A9</td>
</tr>
<tr>
<td>$(x \cdot y) \cdot z = x \cdot (y \cdot z)$</td>
<td>A5</td>
</tr>
</tbody>
</table>

Table 1: Axioms of $BPA_{I}(A)$
In order to reduce the number of brackets in processes we have the following priorities on operators: \( \cdot \) binds stronger than all other operators and \(+\) binds weaker than all other operators.

To the process algebra \( BPA_{h,c}(A) \) we associate a structured operational semantics in the form of the term deduction system \( T(BPA_{h,c}(A)) \) in Table 2. For the deduction rules in this table we require that \( a \in A \) and that \( x, y, \) and \( z \) are arbitrary processes. A deduction rule is of the form \( \frac{H}{C} \) where \( H \) is a set of hypotheses and \( C \) is the conclusion. The formula \( x \overset{a}{\rightarrow} x' \) expresses that the process \( x \) can perform an action \( a \) and thereby evolves into the process \( x' \) and the formula \( x \perp \) expresses that process \( x \) has an option to terminate immediately and successfully. For a formal definition of term deduction systems we refer to [BV93].

<table>
<thead>
<tr>
<th>( \varepsilon \perp )</th>
<th>( x \perp )</th>
<th>( y \perp )</th>
<th>( x \perp, y \perp )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( x + y \perp )</td>
<td>( x + y \perp )</td>
<td>( x \cdot y \perp )</td>
<td></td>
</tr>
<tr>
<td>( a \overset{a}{\rightarrow} \varepsilon )</td>
<td>( x + y \overset{a}{\rightarrow} x' )</td>
<td>( x + y \overset{a}{\rightarrow} y' )</td>
<td>( x \cdot y \overset{a}{\rightarrow} y' )</td>
</tr>
</tbody>
</table>

Table 2: Structured Operational Semantics of \( BPA_{h,c}(A) \)

Finally we would like to mention the following well-known result from literature (e.g. [BW90]): The process algebra \( BPA_{h,c}(A) \) is a sound and complete axiomatization of bisimulation equivalence, notation \( \equiv \), on the closed \( BPA_{h,c}(A) \) terms. This result will be used in the following sections when relating the extended process algebras to \( BPA_{h,c}(A) \).

2.2 Axiomatization of Interworking Sequencing

In this section we will extend the process algebra \( BPA_{h,c}(A) \) from the previous section with an entity function on the atomic actions from \( A \) and the interworking sequencing operator \( q_w \). The resulting process algebra is denoted by \( IWD_{h,c}(A, EID, E) \). The entity function \( E : A \rightarrow \mathcal{P}(EID) \) associates to every atomic action from the set \( A \) a set of entity identifiers from \( EID \). Intuitively the entities of the atomic action \( a \) are the functional blocks on which the atomic action is defined/executed; these entities are called active entities. The universe of entity identifiers \( EID \) and the entity function \( E \) are considered parameters of the process algebra.

Based on the entity function \( E \) on atomic actions we can associate to every closed term over the signature of the process algebra a set of entities; these are the entities to which the atomic actions of the process refer. This is done by extending the entity function on atomic actions to an entity function \( E \) on process terms. This extension is, for \( a \in A \) and \( x \) and \( y \) arbitrary processes, defined in Table 3. For the atomic actions from \( A \), viewed as a process, the entities are given already by the entity function on atomic actions. For the atomic actions \( \varepsilon \) and \( \delta \) we take \( E(\varepsilon) = \emptyset \) and \( E(\delta) = \emptyset \). For more complex processes the entities are obtained from the atomic actions the process is built from.

<table>
<thead>
<tr>
<th>( E(\varepsilon) )</th>
<th>( E(\delta) )</th>
<th>( E(a \cdot x) )</th>
<th>( E(x + y) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \emptyset )</td>
<td>( \emptyset )</td>
<td>( E(a) \cup E(x) )</td>
<td>( E(x) \cup E(y) )</td>
</tr>
</tbody>
</table>

Table 3: Active Entities of an Interworking
The interworking sequencing of two processes \( x \) and \( y \) is their parallel execution with the restriction that the right-hand side process may execute an action only if the entities of that action are disjoint from the entities of the left-hand side process. The interworking sequencing operator is similar to the weak sequential composition operator from [RW94]. As a starting point for the axiomatization we take the axioms for the interworking sequencing as they are given in [MvWW93]. To obtain an axiomatization of the interworking sequencing operator in the context of empty Interworkings an approach is followed which compares easily with the step from the free merge in a setting without the empty process to a free merge in a setting with the empty process [BW90].

The axiomatization of \( q_w \) as presented in [MvWW93] uses the two auxiliary operators \( L_q \) and \( R_q \). The process \( x L_q y \) behaves like the process \( x q_w y \) with the restriction that the first action to be executed must originate from process \( x \). The process \( x R_q y \) also behaves like the process \( x q_w y \) but this time with the restriction that the first action to be executed must be from process \( y \).

Intuitively, we want \( t: \) to be a unit for the interworking sequencing, i.e. \( t:q_w x = xq_w t: = x \). In particular we also want to have that \( t:q_w \delta = \delta \). The interpretation of \( x q_w \) is that the process \( x \) is forced to do the first step. Since \( \delta \) is unable to do any step, it seems plausible to define \( \varepsilon := q_w \delta \). Consequently, we also define \( \varepsilon := q_w \delta \). If we apply this in the definition of the sequencing operator as given in [MvWW93] we get \( \delta := q_w \delta + \delta \). This is not what we want and therefore we need the additional operator \( ..j \) as given in Table 4. This operator has also been used by Baeten and Weijland [BW90] in axiomatizing the free merge in a process algebra containing the empty process. The definition of the interworking sequencing operator is given in Table 4.

\[
\begin{align*}
x q_w y &= x L_q y + x R_q y + \sqrt(x) \cdot \sqrt(y) & S1 \\
\varepsilon L_q x &= \delta & S1 \\
\delta L_q x &= \delta & S2 \\
a \cdot x L_q y &= a \cdot (x q_w y) & S3 \\
(x + y) L_q z &= x L_q z + y L_q z & S4 \\
x R_q \varepsilon &= \delta & S5 \\
x R_q \delta &= \delta & S5 \\
x R_q a \cdot y &= a \cdot (x q_w y) & S6 \\
x R_q a \cdot y &= \delta & S6 \\
x R_q (y + z) &= x R_q y + x R_q z & S7 \\
\sqrt(\varepsilon) &= \varepsilon & T1 \\
\sqrt(\delta) &= \delta & T2 \\
\sqrt(a \cdot x) &= \delta & T3 \\
\sqrt(x + y) &= \sqrt(x) + \sqrt(y) & T4
\end{align*}
\]

Table 4: Axioms for interworking sequencing

The structured operational semantics of the interworking sequencing and of the auxiliary operators is given in Table 5. The term deduction system \( T(IWD, (A, EID, E)) \) consists of the deduction rules of \( T(BPA, (A)) \) and the deduction rules of Table 5.

Next, we will formulate some interesting theorems concerning this process algebra. These theorems relate the process algebra \( IWD, (A, EID, E) \) to the process algebra \( BPA, (A) \) and to the structured operational semantics as given by the term deduction systems.

**Theorem 2.2.1 (Congruence)** Bisimulation equivalence is a congruence for the function symbols in the signature of \( IWD, (A, EID, E) \).

**Proof** It is straightforward to verify that the deduction rules of the term deduction system which
Table 5: Structured Operational Semantics of interworking sequencing and auxiliary operators

| x ⊨ y ⊨ x qw y ⊨ x Lqw y ⊢ x' qw y ⊨ x ↝ x' ⊨ y ↝ y', E(a) ∩ E(x) = ∅ ⊨ x ↝ x' qw y ⊨ z ↝ z qw y' |

Theorem 2.2.2 (Soundness) The process algebra IWDₜ(A, EID, E) is a sound axiomatization of bisimulation equivalence on closed IWDₜ(A, EID, E) terms.

Proof Due to the fact that bisimulation is a congruence, we only have to verify the soundness of each axiom. For the axioms S1, LS3,4, and RS3,5 we relate the left-hand side to the right-hand side and we add the diagonal (i.e. we relate each term to itself). For the other axioms we only relate the left-hand side to the right-hand side.

Theorem 2.2.3 (Conservativity) The process algebra IWDₜ(A, EID, E) is a conservative extension of the process algebra BPAₖₜₜ(A).

Proof The proof of this theorem uses the approach of Verhoef [Ver94]. The theorem follows from the following observations:

1) Bisimulation is definable in terms of predicate and relation symbols only,

2) the process algebra BPAₖₜₜ(A) is a complete axiomatization of bisimulation equivalence on closed BPAₖₜₜ(A) terms (see [BV95, BW90]),

3) the process algebra IWDₜ(A, EID, E) is a sound axiomatization of bisimulation equivalence on closed IWDₜ(A, EID, E) terms (see Theorem 2.2.2),

4) the term deduction system T(BPAₖₜₜ(A)) is pure¹, well-founded² and in path format, and

5) the term deduction system T(IWDₜ(A, EID, E)) is in path format³.

Theorem 2.2.4 (Elimination) The process algebra IWDₜ(A, EID, E) has the elimination property for the process algebra BPAₖₜₜ(A).

Proof The term rewrite system associated with the axioms A3-A7 from Table 1 and the axioms from Tables 3 and 4 and the additional rewriting rules from Table 6 is strongly normalizing. This can be proven with the method of the lexicographical path ordering [KL80, Klo92]. Note that the additional rewriting rules are, for closed terms, derivable from the axioms of IWDₜ(A, EID, E).

If we additionally show that every normal form of the closed terms is in fact a closed BPAₖₜₜ(A) term then the theorem follows easily. Thereto, suppose that s is a normal form with respect to the term rewrite system and suppose that s is not a closed BPAₖₜₜ(A) term. Then s must contain at

¹For a definition of pure term deduction systems see [BV93].
²For a definition of well-founded term deduction systems see [BV93].
³For a definition of the path format see [BV93].
least one occurrence of the operators q_{w}, L_{qw}, R_{qw}, or √. Take a smallest subterm of s which is headed by one of these operators. In any case it follows that the operands of this operator are closed BPA_{E}(A) terms. From that it is easily seen that a rewrite rule must be applicable, which contradicts the assumptions. Therefore, we conclude that every normal form of a closed IWD_{E}(A, E, E) term is a closed BPA_{E}(A) term.

Theorem 2.2.5 (Completeness) The process algebra IWD_{E}(A, E, E) is a complete axiomatization of bisimulation equivalence on closed IWD_{E}(A, E, E) terms.

Proof By the General Completeness Theorem of Verhoef [Ver94], the completeness of the process algebra IWD_{E}(A, E, E) follows immediately from the propositions which are used in the proof of Theorem 2.2.3 and the fact that IWD_{E}(A, E, E) has the elimination property for BPA_{E}(A) (see Theorem 2.2.4).

2.3 Axiomatization of Interworking Merge
In this section we will extend the process algebra IWD_{E}(A, E, E) from the previous section with the interworking merge operator (||_E). The resulting process algebra is called IWE_{E}(A, E, E). Thereto, we first describe IW_{E}(A, E, E), the extension of IWD_{E}(A, E, E) with the E-interworking merge operator (||_E). Technically speaking, we can axiomatize the interworking merge without using the E-interworking merge. But, to stay as close as possible to the existing axiomatization of the interworking merge, we use the E-interworking merge. After that we introduce tuples of process terms and entity sets. On this new structure we define the interworking merge operator.

As we have shown in the introduction there is a problem with axiomatizing the interworking merge operator. From an expression representing an Interworking it is not possible to determine the empty entities, since the Interworking can have empty entities which are not represented in the atomic actions describing the Interworking. In this section we will solve this problem by associating to every closed term of the process algebra a label denoting the entities which are present. With that additional information it is straightforward to give an axiomatization of the interworking merge. As was done in [MvWW93] the interworking merge is expressed in terms of the E-interworking merge operator and the common entities of the operands.

The axiomatization of the S-interworking merge as presented in [MvWW93] uses the auxiliary operators left S-interworking merge ||_L_{S} and synchronization interworking merge ||_S_{w} with S a set of atomic actions. We will use similar auxiliary operators only now labelled with a set of entities in stead of a set of atomic actions. This set represents the entities on which communication actions must synchronize. The process x ||_{E}^{S} y is the parallel execution of the processes x and y with the restriction that the processes must synchronize on all atomic actions which are defined on entities from the set E. The process x ||_{E}^{S} y behaves like the process x ||_{E}^{w} y with the restriction that the first action must come from process x and that action does not have to synchronize with an action from y. The process x ||_{E}^{S} y behaves as the process x ||_{E}^{w} y with the restriction that the first action to be executed must be a synchronization.
Since we have defined the entities of the empty process to be the empty set, it seems plausible that we define $x I_i \omega < \delta$ and $\epsilon I_i \omega x = x I_i \omega \epsilon = \delta$. The interpretation of $x I_i \omega y$ and $x I_i \omega y$, however, make it reasonable to define $\epsilon I_i \omega x = \delta$ and $\epsilon I_i \omega x = x I_i \omega \epsilon = \delta$. With respect to the main axiom for the interworking merge from [MvWW93]: $x I_i \omega y = x I_i \omega x + x I_i \omega y$, we derive $\epsilon I_i \omega x = \epsilon I_i \omega x + \epsilon I_i \omega x = \delta + \delta = \delta$. In terms of Interworkings this means that two empty Interworkings are not consistent which is not what we want. Therefore, we need to redefine the interworking merge operator in a way similar to the definition of the sequencing operator. The new axioms for the $\epsilon$-interworking merge operator are given in Table 7. Recall that the axioms for the termination operator are given in Table 4.

$$x I_i \omega y = x I_i \omega y + x I_i \omega y + \sqrt{x} \cdot \sqrt{y}$$

Table 7: Axioms of $\epsilon$-interworking merge

Next, we present a structured operational semantics for the operators which are introduced in this section. The term deduction system $T(IW,(A, EID, E))$ consists of the deduction rules of $T(IW,(A, EID, E))$ and the deduction rules of Table 8.

Table 8: Structured Operational Semantics of $\epsilon$-interworking merge and auxiliary operators

Before we turn to the interworking merge operator we list results similar to those of the previous section for the process algebra $IW_e(A, EID, E)$. The proofs are omitted.

Theorem 2.3.1 (Congruence) Bisimulation equivalence is a congruence for the function symbols in the signature of $IW_e(A, EID, E)$. 

8
Theorem 2.3.2 (Soundness) The process algebra $IW_e(A, EID, E)$ is a sound axiomatization of bisimulation equivalence on closed $IW_e(A, EID, E)$ terms.

Theorem 2.3.3 (Conservativity) The process algebra $IW_e(A, EID, E)$ is a conservative extension of the process algebra $IWD_e(A, EID, E)$.

Theorem 2.3.4 (Elimination) The process algebra $IW_e(A, EID, E)$ has the elimination property for the process algebra $IWD_e(A, EID, E)$.

Theorem 2.3.5 (Completeness) The process algebra $IW_e(A, EID, E)$ is a complete axiomatization of bisimulation equivalence on closed $IW_e(A, EID, E)$ terms.

Now that we have given the axioms and structured operational semantics of the $E$-interworking merge we will discuss the interworking merge operator. The interworking merge of two processes is their parallel execution with the restriction that the processes must synchronize on all atomic actions which are defined on the common entities of the processes. For the interworking merge operator it is necessary to determine the common entities of the operands. The entities of an operand cannot be obtained from the process term representing it, since empty entities are not represented in the process term. Therefore, we label every process term by a set of entity names over $EID$. For an Interworking $x$, this set together with the active entities of $x$ (i.e. $E(x)$) represents the entities of the Interworking (including the empty entities). An interworking with a dynamical behaviour denoted by $x$ over the entities from $E$ is denoted by $(x, E)$. We do not require that the set $E$ consists of all entities of the interworking; instead we require that the entities of the interworking are given by $E \cup E(x)$. The reason for this choice is that in this setting every tuple from which the dynamical behaviour describes an interworking, can be interpreted as an Interworking in the extended semantics. Such a tuple $(x, E)$ will be called a labelled process. On labelled processes we define the operators $+, \cdot, q_w, \|_w$, and $\|_w$. The set of all labelled processes is called $LP$. The axioms for labelled processes are, for $x, y$ processes, $E, E_1$, and $E_2 \subseteq EID$, and $\Theta \in \{+, \cdot, q_w, \|_w\}$, given in Table 9.

<table>
<thead>
<tr>
<th>$\Theta$</th>
<th>$\Theta(x, E, y, E_2)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$+$</td>
<td>$(x \cup E \cup E(x), E_1 \cup E_2)$ if $E(x) \subseteq E$</td>
</tr>
<tr>
<td>$\cdot$</td>
<td>$(x \cdot y, E_1 \cup E_2)$ if $E(x) \subseteq E_1 \land E(y) \subseteq E_2$</td>
</tr>
<tr>
<td>$q_w$</td>
<td>$(x |_w y, E_1 \cup E_2)$ if $E(x) \subseteq E_1 \land E(y) \subseteq E_2$</td>
</tr>
</tbody>
</table>

Table 9: Extension to labelled processes

Example 2.3.6 The labelled processes associated to the Interworkings $A$, $B$, and $C$ from Figure 1 are the following.

$A = (c(p, q, m1) q_w c(q, r, m2), \{p, q, r\})$

$B = (c(p, q, m1) q_w c(q, r, m2), \{p, q, r, s\})$

$C = (c(q, r, m2) q_w c(r, s, m3), \{q, r, s\})$

Then we have the following computations for $A \|_w C$ and $B \|_w C$.

$A \|_w C = (c(p, q, m1) q_w c(q, r, m2), \{p, q, r\}) \|_w (c(q, r, m2) q_w c(r, s, m3), \{q, r, s\})$

$= (c(p, q, m1) q_w c(q, r, m2) \|_w c(q, r, m2) q_w c(r, s, m3), \{p, q, r, s\})$

$= (c(p, q, m1) q_w c(q, r, m2) q_w c(r, s, m3), \{p, q, r, s\})$
and
\[
B \parallel_{\omega} C = (c(p, q, m1) \omega c(q, r, m2), \{p, q, r, s\}) \parallel_{\omega} (c(q, r, m2) \omega c(r, s, m3), \{g, r, s\})
\]
\[
= (c(p, q, m1) \omega c(q, r, m2) \parallel_{\omega} (c(q, r, m2)) /\omega c(r, s, m3), \{p, q, r, s\})
\]
\[
= (c(p, q, m1) \omega c(q, r, m2) \parallel_{\omega} \delta, \{p, q, r, s\})
\]

Clearly, the expressions do not denote the same interworking in this extended setting. In other words, the labelled processes are not entity bisimilar (see Definition 2.3.7).

Observe that the role of the empty process as a neutral element for the sequential composition, the interworking sequencing and the interworking merge is, with respect to closed labelled processes, taken over by the labelled process \( (c, 0) \), i.e.
\[
(c, 0) \parallel_{\omega} x = x = x \parallel_{\omega} (c, 0).
\]

The role of deadlock as the left zero element for the sequential composition operator is taken over by the labelled process \( (\delta, EID) \), i.e.
\[
(\delta, EID) \parallel_{\omega} x = (\delta, EID).
\]

These properties of the labelled processes can all easily be derived from the axioms from Table 9 and the axioms for the operators in the process algebra.

Next, we define a structured operational semantics for labelled processes. The deduction rules of the term deduction system \( T(IWE_e(A, EID, E)) \) are given by the deduction rules of \( T(IW_e(A, EID, E)) \) and the deduction rules from Table 10.

<table>
<thead>
<tr>
<th>Rule</th>
<th>Derivation</th>
</tr>
</thead>
<tbody>
<tr>
<td>( x \parallel \omega (x, E) \downarrow )</td>
<td>( (x, E) \parallel_{\omega} \delta, {x, E} )</td>
</tr>
<tr>
<td>( x \parallel \omega (x, E) \downarrow )</td>
<td>( (x, E) \parallel_{\omega} \delta, {x, E} )</td>
</tr>
<tr>
<td>( (x, E) \parallel_{\omega} (y, F) \downarrow )</td>
<td>( (x, E) \parallel_{\omega} (y, F) \downarrow )</td>
</tr>
<tr>
<td>( (x, E) \parallel_{\omega} (y, F) \downarrow )</td>
<td>( (x, E) \parallel_{\omega} (y, F) \downarrow )</td>
</tr>
<tr>
<td>( (x, E) \parallel_{\omega} (y, F) \downarrow )</td>
<td>( (x, E) \parallel_{\omega} (y, F) \downarrow )</td>
</tr>
</tbody>
</table>

Table 10: Structured Operational Semantics of labelled processes

Definition 2.3.7 The closed LP terms \( (s, E) \) and \( (t, F) \) are entity bisimilar, \( (s, E) \equiv_{E} (t, F) \), if and only if \( s \equiv t \) and \( E \cup E(x) = F \cup E(y) \).

Theorem 2.3.8 (Congruence) Entity bisimulation equivalence is a congruence for the function symbols in the signature of \( IWE_e(A, EID, E) \) which are defined on LP terms.

Proof The theorem from [BV93] as used in the previous theorems on congruence is not applicable in this case. This theorem is only formulated for strong bisimulation equivalence. Nevertheless we will see that Theorem 2.3.8 is not too hard to prove. Suppose that \( (x_1, E_1) \equiv_{E} (x_2, E_2) \) and \( (y_1, F_1) \equiv_{F} (y_2, F_2) \). Now we have to prove, for \( \oplus \in \{+, \parallel_{\omega}, \parallel_{\omega} \} \), that \( (x_1, E_1) \oplus (y_1, F_1) \equiv_{E} (x_2, E_2) \oplus (y_2, F_2) \). From the assumptions we have the following: \( x_1 \equiv x_2, y_1 \equiv y_2, E_1 \cup E(x_1) = E_2 \cup E(x_2) \) and \( F_1 \cup E(y_1) = F_2 \cup E(y_2) \).

We will first verify the termination predicate. Suppose that \( (x_1, E_1) \parallel_{\omega} (y_1, F_1) \downarrow \). According to the deduction rules this is only the case if \( x_1 \parallel \omega y_1 \). From \( x_1 \equiv x_2 \) and \( y_1 \equiv y_2 \) we obtain \( x_2 \parallel \omega y_2 \). Using the same deduction rule as before we have \( (x_2, E_2) \parallel_{\omega} (y_2, F_2) \downarrow \). In the other direction the proof is analogous. Suppose that \( (x_1, E_1) \parallel_{\omega} (y_1, F_1) \downarrow \). Then we have \( (x_1, E_1) \downarrow \).
and \((y_1, F_1) \downarrow\). But then also \(x_1 \downarrow\) and \(y_1 \downarrow\). By the assumptions we then have \(x_2 \downarrow\) and \(y_2 \downarrow\). Then we can deduce \((x_2, E_2) \downarrow\) and \((y_2, E_2) \downarrow\) and from that \((x_2, E_2) \downarrow_{lw} (y_2, F_2) \downarrow\). In the other direction the proof is analogous.

Next, we will show that every step from the left-hand side can be mimicked by the right-hand side, and vice versa. Also we will show that the resulting labelled processes are entity bisimilar. Suppose that \((x_1, E_1) \downarrow_{lw} (y_1, F_1) \downarrow (z, G)\). Inspection of the deduction rules gives us that we must have that \(x_1 \oplus y_1 \sim z\) and \(G = E_1 \cup F_1 \cup E(x_1) \cup E(y_1)\). From this we get with the assumptions \(x_2 \oplus y_2 \sim z\) and \(G = E_2 \cup F_2 \cup E(x_2) \cup E(y_2)\). Therefore, with the same deduction rule we deduce \((x_2, E_2) \oplus (y_2, F_2) \sim (z, G)\). The proof in the other direction is analogous.

Theorem 2.3.8 (Soundness) The process algebra \(IWE_e(A, EID, E)\) is a sound axiomatization of bisimulation equivalence on closed \(IWE_e(A, EID, E)\) terms. The process algebra \(IWE_e(A, EID, E)\) is a sound axiomatization of entity bisimulation on closed \(IWE_e(A, EID, E)\) terms.

Proof For the first proposition see Theorem 2.3.2 and observe that we did not add any axioms relating closed \(IWE_e(A, EID, E)\) terms. We will prove the second proposition. Since entity bisimulation is a congruence for the closed terms of \(LP\) we only have to show that the axioms are sound. For the first axiom relate the left-hand side to the right-hand side. For the other axioms we additionally relate each term to itself.

Theorem 2.3.9 (Conservativity) The process algebra \(IWE_e(A, EID, E)\) is a conservative extension of the process algebra \(IWE_e(A, EID, E)\).

Proof The theorem follows from the following observations:

1) Bisimulation is definable in terms of predicate and relation symbols only,
2) \(IWE_e(A, EID, E)\) is a complete axiomatization of bisimulation on closed \(IWE_e(A, EID, E)\) terms (see Theorem 2.3.5),
3) \(IWE_e(A, EID, E)\) is a sound axiomatization of bisimulation on closed \(IWE_e(A, EID, E)\) terms (see Theorem 2.3.9), and
4) \(T(IWE_e(A, EID, E))\) is pure, well-founded and in path format, and
5) \(T(IWE_e(A, EID, E))\) is in path format.

Definition 2.3.11 Basic \(LP\) terms are defined inductively as follows:

1) if \(s\) is a closed \(IWE_e(A, EID, E)\) term and \(E \subseteq EID\) such that \(E(s) \subseteq E\), then \(\langle s, E \rangle\) is a basic \(LP\) term
2) no other closed \(LP\) terms are basic \(LP\) terms

Theorem 2.3.12 (Elimination) For every closed \(LP\) term \(s\) there exists a basic \(LP\) term \(t\) such that \(IWE_e(A, EID, E) \vdash s = t\).
Proof This theorem is proven with induction on the structure of a closed LP term. First consider the case \( s \equiv (s', E') \) \( (s' \) a closed \( IWc(A, EID, E) \) term and \( E' \subseteq EID \). If \( E(s') \subseteq E' \) then \( s \) is a basic LP term. If \( E(s') \not\subseteq E' \) then we have \( (s', E') = (s', E' \cup E(s')) \) which is a basic LP term. Next, consider the case \( s \equiv s_1 \oplus s_2 \) \( (s_1, s_2 \) closed LP terms) for \( \oplus \in \{+, \cdot, \cup, \mid\} \). Then we have by induction that \( s_1 \) and \( s_2 \) are basic LP terms, i.e. \( s_1 \equiv (t_1, E_1) \) and \( s_2 \equiv (t_2, E_2) \) for some \( t_1, t_2 \) closed \( IWc(A, EID, E) \) terms and \( E_1, E_2 \subseteq EID \). Then we can apply the second axiom to obtain \( s = (t_1 \oplus t_2, E_1 \cup E_2) \) which is a basic LP term. Finally, consider the case \( s \equiv s_1 \mid\mid s_2 \). Again by induction we have that \( s_1 \) and \( s_2 \) are basic LP terms. Therefore, we have that \( s_1 \equiv (t_1, E_1) \) with \( E(t_1) \subseteq E_1 \) and that \( s_2 \equiv (t_2, E_2) \) with \( E(t_2) \subseteq E_2 \). Then we can apply the third axiom and obtain \( s \equiv (t_1 \mid\mid s_2, t_2 \mid\mid s_2, E_1 \cup E_2) \) which is a closed LP term.

**Theorem 2.3.13 (Completeness)** The process algebra \( IWc(A, EID, E) \) is a complete axiomatization of entity bisimulation on closed LP terms.

**Proof** By the elimination theorem we only have to prove this theorem for basic LP terms. Let \( (s, E_1) \) and \( (t, E_2) \) be basic LP terms such that \( (s, E_1) \sim (t, E_2) \). By the definition of entity bisimulation we have \( s \sim t \) and \( E_1 \cup E(s) = E_1 = E_2 = E_2 \cup E(t) \). In Theorem 2.3.5 we proved that \( IWc(A, EID, E) \) is a complete axiomatization of bisimulation equivalence on closed \( IWc(A, EID, E) \) terms. So we have \( (s, E_1) = (t, E_2) \).

### 3 Properties

In this section we will show that the interworking sequencing is commutative under the assumption that the active entities of the operands are disjoint and that the interworking sequencing is associative. We will also prove that the interworking merge is both commutative and associative. The interworking merge as defined by [MvWW93] did not have the associativity property. This is a direct consequence of our decision to maintain the entities of an Interworking statically. We can illustrate this with the following example.

**Example 3.1** In the semantics of [MvWW93] the Interworkings as shown in Figure 3 are given by \( A = c(p, q, m) \) and \( B = c(p, q, n) \), whereas in this paper they are represented by \( A = (c(p, q, m), \{p, q\}) \) and \( B = (c(p, q, n), \{p, q\}) \).

![Figure 3: Example Interworkings](image)

We compute the expressions \( (A \mid\mid s A) \mid\mid s B \) and \( A \mid\mid s (A \mid\mid s B) \) in both settings. In the semantics of [MvWW93] we have the following computations.

\[
(A \mid\mid s A) \mid\mid s B = (c(p, q, m) \mid\mid s c(p, q, m)) \mid\mid s c(p, q, n)
\]

\[
= c(p, q, m) \mid\mid s c(p, q, n)
\]

\[
= \delta
\]
\[ A \llw (A \llw B) = c(p, q, m) \llw (c(p, q, m) \llw c(p, q, n)) \]
\[ = c(p, q, m) \llw \delta \]
\[ = c(p, q, m) \cdot \delta \]

In the semantics of Interworkings as presented in this paper we have the following computations. Denote the set \( \{p, q\} \) by \( K \).

\[(A \llw A) \llw B = (c(p, q, m), K) \llw (c(p, q, m), K) \llw (c(p, q, n), K) \]
\[= (c(p, q, m), K) \llw \delta, K \]
\[= (c(p, q, m), K) \llw (c(p, q, n), K) \]
\[= (c(p, q, m), K) \llw (c(p, q, n), K) \]
\[= (c(p, q, m), K) \llw \delta, K \]

\[ A \llw (A \llw B) = (c(p, q, m), K) \llw (c(p, q, m), K) \llw (c(p, q, n), K) \]
\[= (c(p, q, m), K) \llw (c(p, q, m), K) \llw (c(p, q, n), K) \]
\[= (c(p, q, m), K) \llw \delta, K \]
\[= (c(p, q, m), K) \llw (c(p, q, n), K) \]
\[= (c(p, q, m), K) \llw \delta, K \]

These two computations illustrate the difference fairly well. In the computation of \( A \llw B \) in the semantics of [MvWW93] we lost information on the entities which are present, whereas in the second computation we did not. Observe that in the definition of interworking merge as presented in this paper \( (A \llw A) \llw B \) and \( A \llw (A \llw B) \) are entity bisimilar.

**Proposition 3.2 (Commutativity of \( \llw \) and \( \llw \))**

For closed \( IW_e(A, EID, E) \) terms \( x \) and \( y \) and sets of entities \( E \), \( E_1 \) and \( E_2 \) we have

\[ E(x) \cap E(y) = \emptyset \Rightarrow x \llw y = y \llw x \quad (1) \]
\[ E(x) \cap E(y) = \emptyset \Rightarrow (x, E_1) \llw (y, E_2) = (y, E_2) \llw (x, E_1) \quad (2) \]
\[ x \llw y = y \llw x \quad (3) \]
\[ (x, E_1) \llw (y, E_2) = (y, E_2) \llw (x, E_1) \quad (4) \]
\[ (x, E_1) \llw (y, E_2) = (y, E_2) \llw (x, E_1) \quad (5) \]

**Proof** For proofs of the propositions (1) and (3) we refer to [Ren93]. Proposition (2) follows immediately from proposition (1). Proposition (4) follows immediately from proposition (3). Proposition (5) can be proven as follows. Denote \( E_1 \cup E(x) \) and \( E_2 \cup E(y) \) by \( E'_1 \) and \( E'_2 \) respectively.

\[ (x, E_1) \llw (y, E_2) = (x, E'_1) \llw (y, E'_2) \]
\[ = (x \llw E'_1 \cap E_2, E'_1 \cup E'_2) \]
\[ = (x, E'_1) \llw (y, E'_2) \]
\[ = (y, E'_2) \llw (x, E'_1) \]
\[ = (y \llw E'_1 \cap E_2, E'_1 \cup E'_2) \]
\[ = (y, E'_2) \llw (x, E'_1) \]
\[ = (y, E_2) \llw (x, E_1) \]

13
Proposition 3.3 (Associativity of \( q_w \) and \( \|_{iw} \)) For closed \( IW(A, EID, E) \) terms \( x, y, \) and \( z \) and \( E_1, E_2, E_3 \subseteq EID \) we have

\[
(x \text{ \( q_w \)} y) \text{ \( q_w \)} z = x \text{ \( q_w \)} (y \text{ \( q_w \)} z) 
\]

\[
(x, E_1) \text{ \( q_w \)} (y, E_2) \text{ \( q_w \)} (z, E_3) = (x, E_1) \text{ \( q_w \)} ((y, E_2) \text{ \( q_w \)} (z, E_3)) 
\]

\[
(x ||_{iw} E_1) ||_{iw} ||_{iw} E_2 ||_{iw} E_3 = x ||_{iw} (E_1 \cup E_2 \cup E_3) \]

\[
(x, E_1) ||_{iw} (y, E_2) ||_{iw} (z, E_3) = (x, E_1) ||_{iw} ((y, E_2) ||_{iw} (z, E_3)) 
\]

Proof For a proof of proposition (1) we refer to [Ren93]. Proposition (2) follows easily from proposition (1):

\[
(x, E_1) \text{ \( q_w \)} (y, E_2) \text{ \( q_w \)} (z, E_3) = (x, E_1) \text{ \( q_w \)} (y \text{ \( q_w \)} z, E_1 \cup E_2 \cup E_3) 
\]

Proposition (3) is proven with simultaneous induction on the total number of symbols in \( x, y \) and \( z \) of the following propositions. We have used the following abbreviations:

\[
S = E_1 \cup E_2 \cup E_3, \\
A = (E_1 \cup E_2) - S, \\
B = (E_2 \cup E_3) - S, \text{ and } C = (E_1 \cup E_3) - S. 
\]

We will not prove these propositions. Proposition (4) follows from proposition (3) as follows. We use the following abbreviations:

\[
F_1 = E_1 \cup E(x), \\
F_2 = E_2 \cup E(y), \text{ and } F_3 = E_3 \cup E(z). 
\]

4 Algebraic Definition of Interworking Refinement

Interworking refinement is the replacement of one entity by a number of entities such that the behaviour of the refining Interworking is identical, in a sense to be made precise shortly, to the original Interworking.

Let \( f : EID \rightarrow EID \) be a partial mapping from entities to entities. An Interworking \( z \) can be an \( f \)-refinement of another Interworking \( y \), denoted by \( z \subseteq_f y \). This is the case if, after renaming of a set of entities of \( z \) into one entity of \( y \) (according to \( f \)) and after removal of all internal actions on the refined entity within the refined interworking and the removal of all internal actions on the
refining entities within the refining interworking, the behaviour of both Interworkings is equal. The mapping $f$ is partial in order to distinguish between an entity $p$ which is not refined at all ($p \notin \text{dom}(f)$) and an entity $p$ which is refined by (amongst others) an entity $p$ ($f(p) = p$). For an example of interworking refinement see Figure 4. The entities $q_1$ and $q_2$ refine the entity $q$.

![Figure 4: Interworking Refinement](image)

The intuition is that the external behaviour of a single entity within the Interworking $x$ can be refined into, or implemented by, a collective behaviour of a number of entities within the Interworking $y$. So, the emphasis is on inter-entity communication and not so much on intra-entity behaviour. Besides the singular $f$-entity refinement discussed above, it is also allowed to consider a number of refinements at the same time: multiple refinement. An example of multiple refinement is given in Figure 5. The entity $p$ is refined by the entities $p_1$ and $p_2$, and the entity $q$ is refined by the entities $q_1$ and $q_2$.

![Figure 5: Multiple Interworking Refinement](image)

The mapping from refining entities to refined entities is provided by the partial function $f$. Before we define this refinement formally we define the renaming function $\rho_f$. This operator renames all occurrences of $e \in \text{EID}$ into $f(e)$. For the axiomatization of this operator it is easier to have a total function instead of a partial one. Therefor, we extend the partial function $f$ to the total function $f^*$ by asserting that $f^*(x) = x$ for all $x$ for which $f$ is not defined. Let $F$ be a set and let $f : F \rightarrow F$ be a partial function. The total function associated with $f$, notation $f^*$, is for all $x \in F$, defined by

$$f^*(x) = \begin{cases} f(x) & \text{if } x \in \text{dom}(f) \\ x & \text{if } x \notin \text{dom}(f) \end{cases}$$

Let $f : \text{EID} \rightarrow \text{EID}$ be a partial function, then the renaming operator $\rho_f$ related to $f$ is defined by the axioms in Table 11. This renaming operator resembles the renaming operator $\rho_f$ from [BB88].
\[ \rho_f (\epsilon) = \epsilon \]
\[ \rho_f (\delta) = \delta \]
\[ \rho_f (c(p, q, m)) = c(f^*(p), f^*(q), m) \]
\[ \rho_f (a \cdot x) = \rho_f (a) \cdot \rho_f (x) \]
\[ \rho_f (x + y) = \rho_f (x) + \rho_f (y) \]

Table 11: Entity Renaming function on processes

In Table 12 the entity renaming operator on processes is extended to labelled processes. Note that also the entity component of a labelled process is renamed with respect to the mapping \( f \).

\[ \rho_f ((x, E)) = (\rho_f (x), \{ f^*(e) \mid e \in E \}) \]

Table 12: Entity Renaming function on labelled processes

In order to remove all internal actions on entities we are not interested in, we substitute the empty process \( \epsilon \) for them. The entities for which we remove the internal actions are given by \( \text{rng}(f) \). We define the set of all internal actions on the entities of a set \( E \) as follows:

\[ \text{Int}(E) = \{ a \in A \mid E(a) \subseteq E \land |E(a)| = 1 \} \]

Let \( I \) be a set of atomic actions, in Table 13 we define the operator \( \epsilon_I \) that renames atomic actions from \( I \) into \( \epsilon \). This operator is taken from [Vra91].

\[ \epsilon_I (\epsilon) = \epsilon \]
\[ \epsilon_I (\delta) = \delta \]
\[ \epsilon_I (a \cdot x) = \epsilon_I (x) \quad \text{if } a \in I \]
\[ \epsilon_I (a \cdot x) = a \cdot \epsilon_I (x) \quad \text{if } a \not\in I \]
\[ \epsilon_I (x + y) = \epsilon_I (x) + \epsilon_I (y) \]

Table 13: Renaming atomic actions into \( \epsilon \)

In Table 14 the operator for removing internal communications is extended to labelled processes. By renaming actions into \( \epsilon \) it can be the case that entity information is removed completely from the process expression. Therefore, we first make sure that all entity information is contained in the entity component of the labelled process.

Let \( f : EID \rightarrow EID \) be a refinement mapping, the \( f \)-refinement relation on labelled processes is then defined by the equation in Table 15.

Next, we extend this notion of refinement with a fixed mapping to a notion of refinement which abstracts from this mapping. This notion of refinement is called entity refinement. Interworking \( x \) is an entity refinement of Interworking \( y \), notation \( x \sqsubseteq_f y \) if and only if there exists a refinement mapping \( f \) such that \( x \sqsubseteq_f y \).

**Example 4.1** As an illustration of this algebraic definition of refinement the refinement relation between the Interworkings in Figure 5 is computed. The left-hand side Interworking will be called
\[ \varepsilon_f((x, E)) = (\varepsilon_f(x), E \cup E(x)) \]

---

**Table 14:** Removing internal communications from labelled processes

\[ x \sqsubseteq_f y \iff \varepsilon_{Int(rng(f))}(y) = \varepsilon_{Int(rng(f))}(\rho_f(x)) \]

---

**Table 15:** f-Refinement

A and the right-hand side Interworking B. Semantically these Interworkings are represented by

\[ A = (c(p_1, p_2, m_3) \cdot q_w \cdot c(p_2, q_1, m_1) \cdot q_w \cdot c(q_1, q_2, m_4) \cdot q_w \cdot c(q_2, q_1, m_5) \]

\[ B = (c(p, q, m_1) \cdot q_w \cdot c(q, p, m_2), \emptyset) \]

Elimination of the \( q_w \) yields the following equations

\[ A = (c(p_1, p_2, m_3) \cdot c(p_2, q_1, m_1) \cdot c(q_1, q_2, m_4) \cdot c(q_2, q_1, m_5) \]

\[ \cdot c(q_1, p_2, m_6), \{q_2, p_1, q_1, q_2\}) \]

\[ B = (c(p, q, m_1) \cdot c(q, p, m_2), \{p, q\}) \]

The refinement mapping \( f \) is given by \( f(p_1) = f(p_2) = p \) and \( f(q_1) = f(q_2) = q \). First, we rename the entities of Interworking \( A \) according to \( f \).

\[ \rho_f(A) = (c(p, p, m_3) \cdot c(p, q, m_1) \cdot c(q, q, m_4) \cdot c(q, q, m_5) \cdot c(q, p, m_2) \cdot c(p, p, m_6), \{p, q\}) \]

The set of actions which should be removed is given by

\[ Int(rng(f)) = \{c(p, p, m) \mid m \in MID\} \cup \{c(q, q, m) \mid m \in MID\} \]

Removing these actions from the Interworkings \( \rho_f(A) \) and \( B \) results in the following equations

\[ \varepsilon_{Int(rng(f))}(\rho_f(A)) = (c(p, q, m_1) \cdot c(q, p, m_2), \{p, q\}) \]

\[ \varepsilon_{Int(rng(f))}(B) = (c(p, q, m_1) \cdot c(q, p, m_2), \{p, q\}) \]

We can conclude that Interworking \( A \) is an \( f \)-refinement of Interworking \( B \).

For the entity refinement relation we have the following properties.

**Proposition 4.2 (Reflexivity)** For all closed labelled processes \( x \) we have \( x \sqsubseteq x \).

**Proof** We have to show that there exists a partial mapping \( f : EID \to EID \) such that \( x \sqsubseteq_f y \).

Take the mapping \( f \) with empty domain. Then \( x \) is an \( f \)-refinement of \( x \).

**Proposition 4.3 (Transitivity)** For all closed labelled processes \( x, y, \) and \( z \) we have

\( x \sqsubseteq y \) and \( y \sqsubseteq z \) implies \( x \sqsubseteq z \)

Table 16: Entity refinement
Proof Let $F$ be some set and let $f : F \rightarrow F$ be a partial function. For all $G \subseteq F$ the extension of $f$ with respect to $G$, notation $f^G$, is, for all $x \in F$, defined as follows

$$f^G(x) = \begin{cases} 
  f(x) & \text{if } x \in \text{dom}(f) \\
  x & \text{if } x \notin \text{dom}(f) \land x \in G \\
  \text{undefined} & \text{if } x \notin \text{dom}(f) \land x \notin G 
\end{cases}$$

Suppose that there exist $f, g : EID \rightarrow EID$ such that $x \subseteq_f y$ and $y \subseteq_g z$. It is our claim that $x \subseteq_{f \circ g} z$. The proof of this claim is omitted.

We do not have that the relation $\subseteq$ is anti-symmetrical. This is due to the treatment of internal actions. Consider, for example, the Interworkings $x = c(p, p, m)$ and $y = c(p, p, n)$. Then we have $x \subseteq y$ and $y \subseteq x$, but we do not have $x = y$. For Interworkings without internal communications we do have antisymmetry of entity refinement. So for Interworkings without internal communication the entity refinement relation is a partial ordering. For the more general class of Interworkings entity refinement is a pre-order.

5 Conclusions

We have given a semantics for Interworkings in which we solved the problems encountered in a former semantics and which allows a definition of refinement. An empty interworking is simply represented by the empty process. The anomaly with respect to empty entities in the context of an interworking merge has been solved by attributing the process expressions with the set of entities involved. The definition of refinement presented here is a little easier than a former definition. We have derived several properties and obtained sound and complete theories.

It is expected that this new semantics has only limited impact on the already existing tools for Interworkings. Furthermore, due to the formal definition also an implementation of refinement seems plausible.

References


<table>
<thead>
<tr>
<th>Computing Science Reports</th>
<th>Department of Mathematics and Computing Science</th>
</tr>
</thead>
<tbody>
<tr>
<td>In this series appeared:</td>
<td>Eindhoven University of Technology</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>93/01 R. van Geldrop</td>
<td>Deriving the Aho-Corasick algorithms: a case study into the synergy of programming methods, p. 36.</td>
</tr>
<tr>
<td>93/02 T. Verhoeff</td>
<td>A continuous version of the Prisoner's Dilemma, p. 17</td>
</tr>
<tr>
<td>93/03 T. Verhoeff</td>
<td>Quicksort for linked lists, p. 8.</td>
</tr>
<tr>
<td>93/04 E.H.L. Aarts</td>
<td>Deterministic and randomized local search, p. 78.</td>
</tr>
<tr>
<td>J.H.M. Korst</td>
<td></td>
</tr>
<tr>
<td>P.J. Zwietering</td>
<td></td>
</tr>
<tr>
<td>93/05 J.C.M. Baeten</td>
<td>A congruence theorem for structured operational semantics with predicates, p. 18.</td>
</tr>
<tr>
<td>C. Verhoef</td>
<td></td>
</tr>
<tr>
<td>93/06 J.P. Velkamp</td>
<td>On the unavoidability of metastable behaviour, p. 29</td>
</tr>
<tr>
<td>93/07 P.D. Moerland</td>
<td>Exercises in Multiprogramming, p. 97.</td>
</tr>
<tr>
<td>93/08 J. Verhoostel</td>
<td>A Formal Deterministic Scheduling Model for Hard Real-Time Executions in DEDOS, p. 32.</td>
</tr>
<tr>
<td>93/10 K.M. van Hee</td>
<td>Systems Engineering: a Formal Approach Part II: Frameworks, p. 44.</td>
</tr>
<tr>
<td>J.A. Bergstra</td>
<td></td>
</tr>
<tr>
<td>J.A. Bergstra</td>
<td></td>
</tr>
<tr>
<td>R.N. Bol</td>
<td></td>
</tr>
<tr>
<td>93/16 H. Schepers</td>
<td>A Trace-Based Compositional Proof Theory for Fault Tolerant Distributed Systems, p. 27</td>
</tr>
<tr>
<td>J. Hooman</td>
<td></td>
</tr>
<tr>
<td>93/17 D. Alstein</td>
<td>Hard Real-Time Reliable Multicast in the DEDOS system, p. 19.</td>
</tr>
<tr>
<td>P. van der Stok</td>
<td></td>
</tr>
<tr>
<td>93/18 C. Verhoef</td>
<td>A congruence theorem for structured operational semantics with predicates and negative premises, p. 22.</td>
</tr>
<tr>
<td>93/19 G.J. Houben</td>
<td>The Design of an Online Help Facility for ExSpect, p. 21.</td>
</tr>
<tr>
<td>93/21 M. Codish</td>
<td>Freeness Analysis for Logic Programs - And Correctness, p. 24</td>
</tr>
<tr>
<td>D. Dans</td>
<td></td>
</tr>
<tr>
<td>G. Filé</td>
<td></td>
</tr>
<tr>
<td>M. Bruynoogh</td>
<td></td>
</tr>
<tr>
<td>93/22 E. Poll</td>
<td>A Typechecker for Bijective Pure Type Systems, p. 28.</td>
</tr>
<tr>
<td>93/23 E. de Kogel</td>
<td>Relational Algebra and Equational Proofs, p. 23.</td>
</tr>
<tr>
<td>93/24 E. Poll and Paula Severi</td>
<td>Pure Type Systems with Definitions, p. 38.</td>
</tr>
<tr>
<td>93/26 W.M.P. van der Aalst</td>
<td>Multi-dimensional Petri nets, p. 25.</td>
</tr>
<tr>
<td>93/27 T. Kloks and D. Kratsch</td>
<td>Finding all minimal separators of a graph, p. 11.</td>
</tr>
<tr>
<td>93/28 F. Kamareddine and R. Nederpelt</td>
<td>A Semantics for a fine λ-calculus with de Bruijn indices, p. 49.</td>
</tr>
<tr>
<td>93/29 R. Post and P. De Bra</td>
<td>GOLD, a Graph Oriented Language for Databases, p. 42.</td>
</tr>
<tr>
<td>93/30 J. Deogun</td>
<td>On Vertex Ranking for Permutation and Other Graphs, p. 11.</td>
</tr>
<tr>
<td>T. Kloks</td>
<td></td>
</tr>
<tr>
<td>D. Kratsch</td>
<td></td>
</tr>
<tr>
<td>H. Müller</td>
<td></td>
</tr>
<tr>
<td>Page</td>
<td>Author(s)</td>
</tr>
<tr>
<td>------</td>
<td>----------</td>
</tr>
<tr>
<td>93/31</td>
<td>W. Köver</td>
</tr>
<tr>
<td>93/33</td>
<td>L. Loyens and J. Moonen</td>
</tr>
<tr>
<td>93/34</td>
<td>J.C.M. Baeten and J.A. Bergstra</td>
</tr>
<tr>
<td>93/36</td>
<td>J.C.M. Baeten and J.A. Bergstra</td>
</tr>
<tr>
<td>93/38</td>
<td>C. Verhoef</td>
</tr>
<tr>
<td>93/41</td>
<td>A. Bijlsma</td>
</tr>
<tr>
<td>93/42</td>
<td>P.M.P. Rambags</td>
</tr>
<tr>
<td>93/43</td>
<td>B.W. Watson</td>
</tr>
<tr>
<td>93/44</td>
<td>B.W. Watson</td>
</tr>
<tr>
<td>93/48</td>
<td>R. Gerth</td>
</tr>
<tr>
<td>94/01</td>
<td>P. America, M. van der Kammen, R.P. Nederpelt, O.S. van Roosmalen, H.C.M. de Swart</td>
</tr>
<tr>
<td>94/02</td>
<td>F. Kamareddine, R.P. Nederpelt</td>
</tr>
<tr>
<td>94/04</td>
<td>J.C.M. Baeten, J.A. Bergstra</td>
</tr>
<tr>
<td>94/06</td>
<td>T. Basten, T. Kunz, J. Black, M. Coffin, D. Taylor</td>
</tr>
<tr>
<td>94/08</td>
<td>O.S. van Roosmalen</td>
</tr>
<tr>
<td>94/09</td>
<td>J.C.M. Baeten, J.A. Bergstra</td>
</tr>
<tr>
<td>Year</td>
<td>Authors</td>
</tr>
<tr>
<td>------</td>
<td>---------</td>
</tr>
<tr>
<td>1994</td>
<td>R. Seljée</td>
</tr>
<tr>
<td>1994</td>
<td>W. Peremans</td>
</tr>
<tr>
<td>1994</td>
<td>R. Bloo, F. Kamareddine, R. Nederpelt</td>
</tr>
<tr>
<td>1994</td>
<td>B.W. Watson</td>
</tr>
<tr>
<td>1994</td>
<td>S. Mauw and M.A. Reniers</td>
</tr>
</tbody>
</table>
94/39  A. Blokhuis  
T. Klok  

94/40  D. Alstein  

94/41  T. Klok  
D. Kratsch  

94/42  J. Engelfriet  
J.J. Vereijken  

94/43  R.C. Backhouse  
M. Bijsterblok  

94/44  E. Brinkema  
J. Davies  
R. Getth  
S. Graf  
W. Janssen  
B. Jonsson  
S. Katz  
G. Lowe  
M. Poel  
A. Pnueli  
C. Rump  
J. Zwiers  

94/45  G.J. Houben  

94/46  R. Bloo  
F. Kamareddine  
R. Nederpelt  

94/47  R. Bloo  
F. Kamareddine  
R. Nederpelt  

94/48  Mathematics of Program Construction Group  

94/49  J.C.M. Baeten  
I.A. Bergstra  

94/50  H. Geuvers  

94/51  T. Klok  
D. Kratsch  
H. Muller  

94/52  W. Penczek  
R. Kuiper  

94/53  R. Getth  
R. Kuiper  
D. Peled  
W. Penczek  

95/01  J.J. Lukkien  

95/02  M. Bezem  
R. Bol  
J.F. Groote  

95/03  J.C.M. Baeten  
C. Verhoef  

95/04  J. Hidders  

95/05  P. Severi  

95/06  T.W.M. Vossen  
M.G.A. Verhoeven  
H.M.M. ten Eikelder  
E.H.L. Aarts  

95/07  G.A.M. de Bruyn  
O.S. van Roosmalen  

95/08  R. Bloo  

95/09  J.C.M. Baeten  
J.A. Bergstra  

95/10  R.C. Backhouse  
R. Verhoeven  
O. Weber  

On the equivalence covering number of splitgraphs, p. 4.  
Distributed Consensus and Hard Real-Time Systems, p. 34.  
Computing a perfect edge without vertex elimination ordering of a chordal bipartite graph, p. 6.  
Concatenation of Graphs, p. 7.  
Category Theory as Coherently Constructive Lattice Theory: An Illustration, p. 35.  
Verifying Sequentially Consistent Memory, p. 160  
Tutorial voor de ExSpect-bibliotheek voor "Administratieve Logistiek", p. 43.  
The $\lambda$-cube with classes of terms modulo conversion, p. 16.  
On $\Pi$-conversion in Type Theory, p. 12.  
Fixed-Point Calculus, p. 11.  
Process Algebra with Propositional Signals, p. 25.  
A short and flexible proof of Strong Normalization for the Calculus of Constructions, p. 27.  
Listing simplicial vertices and recognizing diamond-free graphs, p. 4.  
Traces and Logic, p. 81  
A Partial Order Approach to Branching Time Logic Model Checking, p. 20.  
The Construction of a small Communication Library, p. 16.  
Formalizing Process Algebraic Verifications in the Calculus of Constructions, p. 49.  
Concrete process algebra, p. 134.  
An Isotopic Invariant for Planar Drawings of Connected Planar Graphs, p. 9.  
A Type Inference Algorithm for Pure Type Systems, p. 20.  
A Quantitative Analysis of Iterated Local Search, p. 23.  
Drawing Execution Graphs by Parsing, p. 10.  
Preservation of Strong Normalization for Explicit Substitution, p. 12.  
Discrete Time Process Algebra, p. 20  
Mathpad: A System for On-Line Preparation of Mathematical Documents, p. 15