Eindhoven University of Technology
Department of Mathematics and Computing Science

Composition and Decomposition
in a CPN Model

by

P.M.P Rambags

92/10

Computing Science Note 92/10
Eindhoven, May 1992
This is a series of notes of the Computing Science Section of the Department of Mathematics and Computing Science Eindhoven University of Technology. Since many of these notes are preliminary versions or may be published elsewhere, they have a limited distribution only and are not for review. Copies of these notes are available from the author.

Copies can be ordered from:
Mrs. F. van Neerven
Eindhoven University of Technology
Department of Mathematics and Computing Science
P.O. Box 513
5600 MB EINDHOVEN
The Netherlands
ISSN 0926-4515

All rights reserved
editors: prof.dr.M.Rem
prof.dr.K.M.van Hee.
Composition and Decomposition in a CPN model

P.M.P. Rambags

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

Abstract

We present a composition and decomposition theory for a high-level Petri Net model, viz. the DES model [13, 14, 15], that resembles Coloured Petri Nets (CPN) [18, 19]. We give a syntax for the connection and disconnection of DES’ses, we introduce the notion of a module and we give a semantics for modules. Furthermore, we give sufficient and necessary conditions for the replacement of modules in the DES model.
1 Introduction

A system designer is not able to specify a large, complex system all at once. He usually elaborates his design step by step. There are two well-known design strategies:

Composition, a bottom-up approach. The designer has already a set of well-defined subsystems (or modules) at his disposal, which form the components of the system to be constructed. Some of them are selected and joined. The result is extended with new components until the system is complete.

Decomposition, which goes top-down. The designer starts with a global description of the whole system. He repeatedly selects a part of the description and elaborates it in more detail, until the intended level of detail has been reached.

In the field of Petri Nets, research has mainly be devoted to P/T systems [3, 20, 22], Free Choice systems [6, 7, 8, 9] and a restricted form of Free Choice systems [11]. The aim of all authors is to keep certain nice properties invariant while changing the net. Examples of such properties are proper termination, deadlock freeness, covering by S-invariants, (structurally) liveness and (structurally) boundedness.

In case of a decomposition, if the original system has such a property then a refinement rule guarantees the resulting, more detailed system to have the same property. The original system can also be analyzed with the aid of such refinement rules. Assume it is split into several subsystems by a refinement rule. The subsystems are easily analyzed and they appear to have a property \( P \), preserved by the rule. Then the original system has property \( P \), too, because a refinement rule has an inverse that also preserves the property. Such inverse rules allow for bottom-up approaches (composition).

A set of rules is called complete if all and only nets with a property \( P \) can be constructed from an elementary one by applying the rules. Completeness results for certain classes of Free Choice systems can be found in the publications of Esparza and Silva [6, 7, 8, 9].

In the DES model [13, 14, 15, see also Appendix B], a CPN-like model, tokens have values that might be very complex, e.g. database states. Whether a part of the system at hand can be replaced does no longer depend on nice properties only, but rather on complex values, difficult computations and an unconstrained net structure. Free choice systems, e.g., have a constrained net structure.

The idea is to replace a module with another one, while the environment cannot possibly detect any difference. Decomposition is a case where the latter is more detailed than the former, composition is the other way around. There are also intermediate cases where a module is simply replaced with another one, not necessary less or more detailed, just different.

For Coloured Petri Nets, Chehaibar [5] has explored Reentrant Nets, a kind of modules to transport tokens from certain so-called interface places to other interface places, where all interface places must have the same domain (the same type). In order to be reentrant, a subnet must satisfy various structural and behavioural conditions. We do not have such restrictions, we consider the general case where a module can be any subnet.

First, we describe how to split and join DES’ses. This is the syntactical aspect, we
shall discuss it in Section 3. The syntactical operations should be *compositional* [2], i.e. their application to different DES'ses must yield a DES.

Next, there is a semantical aspect. Suppose we have a system $Y$ in which we can distinguish a module $A$. If $A$ is replaced with a similarly behaving module $B$, then obviously the behaviour of $Y$ does not change. But what is the behaviour of a module? That depends on the environment. In general, a module can do less when it is put in isolation. In Section 4, we define a semantics for modules and we give sufficient and necessary conditions for the replacement of modules in the DES model. In Section 5, we give an application of the developed theory and we conclude in Section 6. See Appendix A for some notations.

Section 2 is a basis for this article. It presents a method to give a semantics for discrete systems in general and techniques to compare such systems.

## 2 Unlabeled Transition System

This section is a short summary of [14, Section 2].

Transition systems can be used to give a semantics of discrete systems. We use unlabeled transition systems to give a non-interleaving semantics for DES'ses.

**Definition 2.1 Unlabeled transition system**

An unlabeled transition system is a triple $(S, L, T)$, where:

- $S$ is a countable set
- $L \subseteq S$
- $T \subseteq S \times S$.

$S$ is called the *state space*, $L$ the set of *initial states* and $T$ the *transition relation*.

An unlabeled transition system consists of a countable set of states. Some states are initial. The system starts in an initial state and then moves from one state to another. Actually, an unlabeled transition system is a directed graph.

This definition of an unlabeled transition system can be found in [10]. In literature, also other classes of transition systems are described, e.g., see [16, 21, 23]. They differ from ours in mainly two aspects, viz.:

- There is only one initial state;
- The transition relation $T$ has been replaced by a set of actions $A$ and a relation $R \subseteq S \times A \times S$, where $(s, a, s')$ in $R$ if action $a$ can make the system move from state $s$ to state $s'$.

Hence, there may be different transitions between two states, while in Definition 2.1 only the existence of a transition can be indicated.
We remark that all these classes of transition systems can be transformed into each other.

One of our primary goals is to compare transition systems with expressions as 'system A is more powerful than system B,' 'system A simulates system B' or 'system A is in fact the same as system B.' Several approaches to formalize such comparisons have been described in literature, e.g. observation equivalence \[21\] and bisimulation equivalence \[1, 16\], but they consider other classes of transition systems and other application areas. We shall introduce our own similarity relationships. An interesting topic for further research would be to relate all classes of transition systems and their comparison techniques.

We apply a reduction on any unlabeled transition system such that all states are reachable and the transition relation is reflexive. This reduction gives just the information we need to determine the behaviour of an unlabeled transition system. All similarity relationships will be based thereupon.

**Definition 2.2 Reduced unlabeled transition system**

Let $X = (S, L, T)$ be an unlabeled transition system. Its reduction $X' = (\hat{S}, \hat{L}, \hat{T})$ satisfies:

- $\hat{S} = \{ s \in S \mid \exists n \in \mathbb{N}_0 : \exists s_0, \ldots, s_n \in S : s_0 \in L \land s_n = s \land \forall i \in \{1, \ldots, n\} : (s_{i-1}, s_i) \in T \}$
- $\hat{L} = L$
- $\hat{T} = \{ (s, s') \in \hat{S} \times \hat{S} \mid s = s' \lor (s, s') \in T \}$

A reduced unlabelled transition system cannot be reduced further, its reduction equals itself.

We map reachable states of a system $A$ onto reachable states of a system $B$ by a total function $f \in \hat{S}_A \rightarrow \hat{S}_B$. Hence, all reachable states of system $A$ have to have a correspondent in system $B$. In fact, they are partitioned into classes and each class of $A$ corresponds to a reachable state of $B$. The sizes of the classes indicate, e.g., the efficiency or the level of detail of system $A$ as compared to system $B$.

Actually, $f$ is a morphism from $A$ to $B$ with an additional constraint on the initial states.

**Definition 2.3 Realization**

Let $A$ and $B$ be unlabeled transition systems. $A$ realizes $B$ with respect to function $f$ iff

- $f \in \hat{S}_A \rightarrow \hat{S}_B$
- $f(\hat{L}_A) \subseteq \hat{L}_B$
Intuitively, we say 'A realizes B' if we have a mapping to project states of A onto states of B such that the mapped behaviour of A is also behaviour of B. Besides this, we require in simulation behaviour of B, under the image of f, to be behaviour of A.

**Definition 2.4 Simulation**

A simulates B with respect to function f iff

- A realizes B with respect to f
- f is surjective
- \( f(L_A) = L_B \)
- \( \forall (t, t') \in \hat{T}_B : \forall s_0 \in f^{-1}(t) : \exists n \in \mathbb{N}_0 : \exists s_1, \ldots, s_n \in f^{-1}(t) : \exists s_{n+1} \in f^{-1}(t') : \forall i \in \{0, \ldots, n\} : (s_i, s_{i+1}) \in \hat{T}_A \)

This definition resembles the definition of branching equivalence [12]. Branching equivalence is defined on labelled transition systems where the arcs are related via their labels. The silent step \( \tau \) is a special one. Instead of a function, a relation is used.

**Lemma 2.5**

Let A simulate B with function f and suppose: f is injective.

Then B simulates A with \( f^{-1} \).

This property gives rise to an equivalence relation on the set of unlabeled transition systems.

**Definition 2.6 Equivalence**

Two unlabeled transition systems A and B are equivalent iff a bijective function \( f : \hat{S}_A \rightarrow \hat{S}_B \) exists with the following properties:

- \( f(L_A) = L_B \)
- \( \forall s, s' \in \hat{S}_A : (s, s') \in \hat{T}_A \iff \langle f(s), f(s') \rangle \in \hat{T}_B \).

Notation: \( A \cong B \).
We say ‘$A$ is equivalent with $B$ w.r.t. $f$’ if $f$ satisfies all requirements of this definition. It is easy to see that equivalence equals simulation with an injective function.

Lemma 2.7

$A \cong B$ iff an injective function $f$ exists such that $A$ simulates $B$ with $f$.

\[ \square \]

The simulation relation has actually been defined too strong. However, weakening would make it memory dependent, which is a disadvantage in many practical applications. Consider, for example, Figure 2.1.

\begin{figure}[h]
\centering
\includegraphics[width=0.8\textwidth]{railway.png}
\caption{Railway.}
\end{figure}

$B$ is a specification. Dots $a$ to $f$ represent cities and an arrow between two cities corresponds to a directed railway. For some reason the constructors decided to build two railway stations in cities $c$ and $d$, giving implementation $A$.

The implementation is correct, since each trip of $B$ is possible in $A$ and $A$ allows nothing more. Thus one might expect $A$ to simulate $B$ with $g$ where $g$ is the identity function except for $g(c1) = g(c2) = c$ and $g(d1) = g(d2) = d$. But $A$ does not simulate $B$: In $B$ there is a transition from $d$ to $f$ and the simulation relation requires a transition in $A$ from $d1 \in g^{-1}(d)$, possibly via $d2$, to $f$.

An exact simulation relation would also consider all predecessors of $d1$: From $a$, $b$, $c1$ and $c2$ it is possible to go to $d2$ and next to $f$. That is what we call memory dependence.

In order to verify the exact simulation relation one has to keep track of all previous states and transitions, whereas in Definition 2.4 only states in $f^{-1}(t) \cup f^{-1}(t')$ have to be considered, no prior ones. This saves a lot of effort, especially in complex cases.

The exact forms of the realisation and simulation relation look like, resp.:

$A$ realizes $B$ with $f$: $f(\text{paths in } A) \subseteq \text{paths in } B$

$A$ simulates $B$ with $f$: $f(\text{paths in } A) = \text{paths in } B$

where a path is a sequence of states where each pair of successive states belongs to the transition relation.

Our realization relation (Definition 2.3) is exact and our simulation relation is stronger than the exact form.
3 Disconnection and Connection of DES'ses

In this section, we describe a technique for splitting and joining DES'ses.\footnote{See Appendix B for a definition of the DES model.} We start with the former.

3.1 Disconnection

We intend to split some DES $D$ into two parts $A$ and $B$. In literature, two approaches have been described to do so [17]: Draw a line through either some processors or some channels. We shall first discuss the splitting of processors.

![Figure 3.1: A processor being split.](image)

Figure 3.1 shows a processor being split into two parts. The processor has actually been duplicated in some way. Several questions have to be answered here, for example:

- Is it possible to give a definition of the processors in DES $A$ and $B$?
- If so, how should they be defined?

Please remember: Processors are mathematical functions over their input channels. We do not know how to split these functions. Besides, we have an argument in disfavour of the processor splitting: In Figure 3.1, the processor of DES $D$ can fire only if \textit{both} input channels have a token. On the contrary, in $A$ and $B$ a processor can already fire if \textit{only one} of these channels contains a token. The splitting of processors apparently results in a totally different semantics. Therefore, we abandon the idea of splitting processors. We proceed with the other case: Splitting of channels.

Figure 3.2 shows some DES $D$. Before we describe how to split certain channels, we remark that the set of channels to be split should not be chosen arbitrarily. For example, channels connected to just one processor need not be split. This consideration results in an exclusion of channel $c_4$ (among others). Moreover, if we decide to draw a line through channel $c_1$, we implicitly assign processors $p_1$ and $p_2$ to different DES'ses. Since we do not want to change the definition of processors, we have to split channel $c_3$, too. As a consequence, channel $c_2$ or $c_5$ must also be split.
Processors should not be altered, so it is useful to determine for each processor, to which part (A or B) it is going to belong. The splitting of a DES D is done by partitioning all processors into two classes \( P_A \) and \( P_B \). By doing so, the channels are partitioned implicitly, too, yielding four classes:

- Channels connected only to processors of \( P_A \);
- Channels connected only to processors of \( P_B \);
- Common channels;
- Channels connected to no processor whatsoever.

In many practical cases, the last class is empty, i.e. all channels in \( D \) are connected to a processor. In Figure 3.3, we have made such a division. Processors from \( P_A \) and \( P_B \) have been marked with an 'A' and 'B', respectively. Common channels have been denoted with 'CC'.

The resulting DES'ses \( A \) and \( B \) are obtained by duplication of the common channels.

Now we formalize the disconnection technique. We distinguish symbols related to different DES'ses by means of subscripts.

**Definition 3.1** Restriction operator \( \uparrow \)

Let \( D = \langle R_D, C_D, I_D, O_D, L_D \rangle \) be a DES with processors \( P_D \), channels \( K_D \), token set \( Q_D \), transition system \( \langle S_D, L_D, T_D \rangle \) and let \( P_X \subseteq P_D \) be a set of processor indices. Then we define a set of channel indices:

\[
K_X := \{ k \in K_D \mid \exists p \in P_X : k \in I_D(p) \cup O_D(p) \}
\]

We define a DES \( X \):

\[
X := \langle R_D \downarrow P_X, C_D \downarrow K_X, I_D \downarrow P_X, O_D \downarrow P_X, \{ \uparrow l \downarrow Q_X \mid l \in L_D \} \rangle.
\]

\(^2\)Common channels might be compared with *interface places* in [5], though we pose no restrictions on connections and (initial) states and we don't distinguish initial and final common channels.
3 Disconnection and Connection of DES'ses

Figure 3.3: Disconnection.
X implicitly defines a token set $Q_X$, a state space $S_X$ and a transition relation $T_X$. We call $X$ the restriction of $D$ to $P_X$, denoted as $D \uparrow P_X$.

In Figure 3.3, $A = D \uparrow P_A$ and $B = D \uparrow P_B$. The set of common channels $CC$ equals $K_A \cap K_B$.

A DES $D$ equals its restriction to its own processor set if and only if $D$ has no detached channels, i.e., iff the fourth channel class (see page 8) is empty.

**Lemma 3.2**
Let $D$ be a DES.

$$D = D \uparrow P_D \text{ iff } \forall k \in K_D : \exists p \in P_D : k \in I_D(p) \cup O_D(p).$$

**Proof**
Follows immediately from the construction of $K_X$ in Definition 3.1.

Most Petri Net models forbid isolated nodes. See also the remark on page 35.

### 3.2 Connection

This time we have two DES'ses $A$ and $B$, which are the constituents of a larger, to be constructed DES $D$. We intend to attach them. A picture of this idea would be like Figure 3.3 with the big arrows in the middle reversed.

Not each pair of DES'ses fit together.

**Definition 3.3** *fit*
Let $A$ and $B$ be DES'ses. They fit together, notation $A \text{ fit } B$, iff

- $P_A \cap P_B = \emptyset$
- $C_A \uparrow CC = C_B \uparrow CC$
- $I_A \uparrow Q_{CC} = I_B \uparrow Q_{CC}$

where $CC = K_A \cap K_B$ and $Q_{CC} = Q_A \cap Q_B$.

In this definition, $CC$ denotes the set of common channels of $A$ and $B$ and $Q_{CC}$ is its token set. Please note: By means of a renaming of the processors, two DES'ses $A$ and $B$ can always be transformed in such a way that the first requirement $(P_A \cap P_B = \emptyset)$ is met. The second requirement asserts that a common channel must have the same type in each DES. According to the last requirement, every initial state in each DES must
have a counterpart in the other one. If we would omit this requirement, then several initial states might disappear when \( A \) and \( B \) would be connected (see next definition). This would mean that a given DES could no longer be initiated in certain initial states if it would be connected to another one.

**Definition 3.4** *Connection operator \( \oplus \)\n
Let \( A \) and \( B \) be DES’s that fit together. We define a DES \( D \):

\[
D := \langle R_A \cup R_B, C_A \cup C_B, I_A \cup I_B, O_A \cup O_B, \\
\{s \in S_D \mid s \mid Q_A \in L_A \land s \mid Q_B \in L_B \} \rangle.
\]

\( D \) implicitly defines a token set \( Q_D \), a state space \( S_D \) and a transition relation \( T_D \).

We call \( D \) the connection of \( A \) and \( B \), denoted as \( A \oplus B \). \( \Box \)

Because of the last requirement in Definition 3.3, no initial state of \( A \) or \( B \) got lost in \( D \): \( L_D \upharpoonright Q_A = L_A \) and \( L_D \upharpoonright Q_B = L_B \).

Without proof we mention that \( \oplus \) is a commutative operator and for any DES \( D \), \( D \uplus D \) and \( D \oplus D = D \).

Furthermore, if \( X, Y \) and \( Z \) are DES’s that pairwise fit together, then \( (X \oplus Y) \uplus Z \), \( X \uplus (Y \oplus Z) \) and \( (X \oplus Y) \oplus Z = X \oplus (Y \oplus Z) \).

Suppose we split a given DES \( X \) into two parts \( A \) and \( B \) and we join \( A \) and \( B \). The resulting DES \( Y \) does not always equal \( X \). As we already mentioned before, \( X \) should have no detached channels. If \( X \) meets this requirement, then \( X \) and \( Y \) can only differ in their initial states. To be more precisely, \( L_X \subseteq L_Y \) is always true but a counterexample for \( L_X = L_Y \) can be found. Next lemma expresses when \( L_X = L_Y \), i.e., \( X \) and \( Y \) are equal.

**Lemma 3.5**

Let \( X \) be a DES and \( P \subseteq P_X \). Define \( A := X \uparrow P \) and \( B := X \uparrow (P_X \setminus P) \). Then:

- \( A \uplus B \);
- \( X = A \oplus B \) iff \( X \) has no detached channels and
  \[
  \forall s \in S_X : \forall l, l' \in L_X : \\
  s \upharpoonright Q_A = l \upharpoonright Q_A \land s \upharpoonright Q_B = l' \upharpoonright Q_B \Rightarrow s \in L_X.
  \]

**Proof**

Let \( X \) be a DES, \( P \subseteq P_X \) and define \( A := X \uparrow P \) and \( B := X \uparrow (P_X \setminus P) \). Then \( P_A \cap P_B = P \cap (P_X \setminus P) = \emptyset \), \( C_A \uplus CC = C_X \uplus CC = C_B \uplus CC \) and \( I_A \uplus Q_{CC} = \{l \upharpoonright Q_A \mid l \in L_X\} \upharpoonright Q_{CC} = \{l \upharpoonright Q_{CC} \mid l \in L_X\} = L_B \upharpoonright Q_{CC} \), hence \( A \uplus B \). Define \( Y := A \oplus B \). By construction, \( Y = \langle R_X, C_X, I_X, O_X \rangle \) if and only if \( X \) has no detached channels. Then \( S_Y = S_X, T_Y = T_X \) and \( L_Y = \{s \in S_X \mid \exists l, l' \in L_X : s \upharpoonright Q_A = l \upharpoonright Q_A \land s \upharpoonright Q_B = l' \upharpoonright Q_B\} \).

Hence, \( L_X \subseteq L_Y \). Moreover, \( L_Y \subseteq L_X \) if and only if \( \forall s \in S_X : \forall l, l' \in L_X : s \upharpoonright Q_A = l \upharpoonright Q_A \land s \upharpoonright Q_B = l' \upharpoonright Q_B \Rightarrow s \in L_X \).
\( \Box \)
To illustrate this lemma, we present an example of a DES $X$ where the number of initial states really increases when $X$ is split and immediately afterwards joined.

**Example 3.6 More initial states**

Let $X$ be a DES with $P_X = \{a, b\}$, $K_X = \{c_1, c_2, c_3\}$,

\[
\begin{align*}
C_X & = \{(c_1, N_0), (c_2, N_0), (c_3, N_0)\}, \\
I_X & = \{(a, \{c_1\}), (b, \{c_2\})\}, \\
O_X & = \{(a, \{c_2\}), (b, \{c_3\})\}
\end{align*}
\]

and let it have three initial states:

\[
L_X = \{(c_1, 1), (c_2, 2), (c_3, 3)\}, \\
\{(c_1, 3), (c_2, 2), (c_3, 1)\}, \\
\{(c_2, 4), (c_3, 5)\}\}
\]

Define $A := X \uparrow \{a\}$, $B := X \uparrow \{b\}$ and $Y := A \oplus B$, see Figure 3.4. Then

\[
\begin{align*}
L_A & = \{(c_1, 1), (c_2, 2)\}, \{(c_1, 3), (c_2, 2)\}, \{(c_2, 4)\}\}, \\
L_B & = \{(c_2, 2), (c_3, 3)\}, \{(c_2, 2), (c_3, 1)\}, \{(c_2, 4), (c_3, 5)\}\} \text{ and } \\
L_Y & = \{(c_1, 1), (c_2, 2), (c_3, 3)\}, \\
\{(c_1, 1), (c_2, 2), (c_3, 1)\}, \\
\{(c_1, 3), (c_2, 2), (c_3, 3)\}, \\
\{(c_1, 3), (c_2, 2), (c_3, 1)\}, \\
\{(c_2, 4), (c_3, 5)\}\}.
\end{align*}
\]

So DES $Y$ has two additional initial states as compared to $X$.

A transition in the connection of two DES'ses corresponds to a transition in either one of them, or both.

**Lemma 3.7**

Let $A$ and $B$ be DES'ses such that $A \uparrow \downarrow B$. Then:

\[
\forall s, s' \in S_{A \oplus B} : (s, s') \in T_{A \oplus B} \iff \\
\exists a, a' \in S_A : \exists b, b' \in S_B : s = a \cup b \land s' = a' \cup b' \land \\
( (a, a') \in T_A \land (b, b') \in T_B ) \lor \\
((a, a') \in T_A \land b = b') \lor \\
(a = a' \land (b, b') \in T_B ) .
\]
Proof

\[ \Rightarrow \text{: According to the definition of } T_{A \oplus B}, (s, s') \in T_{A \oplus B} \iff \exists e \in F_{A \oplus B}(s) : s' = s \setminus \text{rng}(e) \cup \bigcup_{p \in \text{dom}(e)} R_{A \oplus B}(p)(e(p)). \text{ We split } e \text{ into } ea := e| P_A \text{ and } eb := e| P_B \text{ and } s \text{ into} \]

\[
\begin{align*}
  a &:= s| (Q_A \setminus Q_{CC}) \cup (\text{rng}(ea) \cap Q_{CC}) \cup x \quad \text{and} \\
  b &:= s| (Q_B \setminus Q_{CC}) \cup (\text{rng}(eb) \cap Q_{CC}) \cup ((s \setminus \text{rng}(e)) \cap Q_{CC} \setminus x)
\end{align*}
\]

where \( x \) may be any bag satisfying \( x \subseteq (s \setminus \text{rng}(e)) \cap Q_{CC} \).

Hence, \( e = ea \cup eb \) and \( s = a \cup b \). We define

\[
\begin{align*}
  a' &:= a \setminus \text{rng}(ea) \cup \bigcup_{p \in \text{dom}(ea)} R_A(p)(ea(p)) \quad \text{and} \\
  b' &:= b \setminus \text{rng}(eb) \cup \bigcup_{p \in \text{dom}(eb)} R_B(p)(eb(p)) .
\end{align*}
\]

Then \( s' = a' \cup b' \). Since \( \text{dom}(e) \neq \emptyset \), there are three possibilities for \( \text{dom}(ea) \) and \( \text{dom}(eb) \):

1. \( \text{Dom}(ea) \neq \emptyset \land \text{dom}(eb) \neq \emptyset \), hence \( \langle a, a' \rangle \in T_A \land \langle b, b' \rangle \in T_B \)
2. \( \text{Dom}(ea) \neq \emptyset \land \text{dom}(eb) = \emptyset \), meaning \( \langle a, a' \rangle \in T_A \land b = b' \)
3. \( \text{Dom}(ea) = \emptyset \land \text{dom}(eb) \neq \emptyset \), implying \( a = a' \land \langle b, b' \rangle \in T_B \).
4 Replacement of Modules

\[ ( (a, a') \in T_A \land (b, b') \in T_B ) \lor (a, a') \in T_A \land b = b') \lor (a = a' \land (b, b') \in T_B ) \]

\[ \Rightarrow \]

\[ ( \exists ea \in F_A(a) : a' = a \setminus \text{mrng}(ea) \cup \bigcup_{p \in dom(ea)} R_A(p)(ea(p))) \land \exists eb \in F_B(b) : b' = b \setminus \text{mrng}(eb) \cup \bigcup_{p \in dom(eb)} R_B(p)(eb(p))) \lor \langle a \cup b, a' \cup b' \rangle \in T_{A \oplus B} \lor \langle a \cup b, a' \cup b' \rangle \in T_{A \otimes B} \]

\[ \Rightarrow \]

\[ \iff \]

\[ \langle s, s' \rangle \in T_{A \oplus B} \]

\[ \square \]

Remark 3.8

By choosing \( x := 0 \) in the \( \Rightarrow \)-part of the above proof, we obtain an \( a \in S_A \) for which \( a \upharpoonright Q_{CC} \) is finite.

\[ \square \]

The next section deals with modules.

4 Replacement of Modules

This section concentrates on the semantical aspects of the disconnection and connection technique from the previous section. Section 3 gives only a syntax, it describes merely a way of splitting and joining DES'ises. What we would like to have is something like Figure 4.1. Here we have a system \( Y \) with module \( A \). The remainder of \( Y \), called \( X \),

\[
\begin{array}{c}
X \\
System Y
\end{array}
\]

\[
\xrightarrow{\text{transform into}}
\]

\[
\begin{array}{c}
X \\
System Y'
\end{array}
\]

\[
A
\]

\[
B
\]

Figure 4.1: Replacement of modules.

can be seen as an environment of module \( A \). Suppose we replace \( A \) by a similarly behaving, but syntactically different module \( B \), then obviously the resulting system \( Y' \) behaves identical to \( Y \). Now several problems arise, such as:
4 Replacement of Modules

- What is the semantics and behaviour of a (closed) system?
- Does the behaviour of a module depend on the environment's behaviour?
  Note: We regard a module as an open system.
- How to compare behaviours?

The first and last problems have already been solved, see Section 2. We claim that any model for discrete systems can be semantically represented at the level of unlabeled transition systems (esp. the DES model) and we have defined three relationships for the comparison of transition systems, viz. realization, simulation and equivalence.

Let us return to the problem in the middle: What is the behaviour of a module and does that depend on the environment's behaviour? Yes, it does. Consider for instance a module consisting of one processor with an input and output channel, as in Figure 4.2. If the input channel is empty at start, the module can do nothing when put in isolation. If, on the other hand, the module is to interact with an environment that produces tokens for the input channel, then the module can start working after a while, even if the input channel is empty at start. In general: The more the environment does, the more the module can do.

![Figure 4.2: A module.](image)

**Definition 4.1 Module**

A module is a pair \( (M, CC) \), where \( M \) is a DES and \( CC \subseteq K_M \).

\( CC \) is the set of common channels, i.e. the set of channels by which the module can interact with an environment.

\( \square \)

**Definition 4.2 Environment**

Let \( (M, CC) \) be a module. A DES \( X \) is an environment of \( (M, CC) \) iff \( X \models M \) and \( K_X \cap K_M \subseteq CC \).

\( \square \)

Hence, the empty DES \( E = (\emptyset, \emptyset, \emptyset, \emptyset, \emptyset) \) is an environment of every module without initial states. The initiate-able DES \( E = (\emptyset, \emptyset, \emptyset, \{\emptyset\}) \) is an environment of each module with initial states.
Now we define a notion of maximality among environments. A maximal environment can influence the module in every possible way another environment might influence the module.

**Definition 4.3 Maximal environment**
Let \( X \) be an environment of module \( (M, CC) \). \( X \) is maximal iff

\[
\begin{align*}
\forall s \in \hat{S}_X : \forall q \in Q_{CC} : \exists s' \in \hat{S}_X : \\
(s, s') & \in T_X \land s' \upharpoonright Q_{CC} = s \upharpoonright Q_{CC} \cup \{q\} \\
\forall s \in \hat{S}_X : \forall q \in s \upharpoonright Q_{CC} : \exists s' \in \hat{S}_X : \\
(s, s') & \in T_X \land s' \upharpoonright Q_{CC} = s \upharpoonright Q_{CC} \setminus \{q\} \quad \text{and} \\
\forall t, t' \in \hat{S}_X : (t, t') & \in T_X \land \\
t \upharpoonright (Q_X \setminus Q_{CC}) & = s \upharpoonright (Q_X \setminus Q_{CC}) \\
t' \upharpoonright (Q_X \setminus Q_{CC}) & = s' \upharpoonright (Q_X \setminus Q_{CC}) \\
\Rightarrow q & \in t \upharpoonright Q_{CC} \\
\forall s, s' \in \hat{S}_X : (s, s') & \in T_X \Rightarrow s \upharpoonright (Q_X \setminus Q_{CC}) \neq s' \upharpoonright (Q_X \setminus Q_{CC})
\end{align*}
\]

The first two requirements indicate that a maximal environment \( X \) can always produce an arbitrary token for, resp. consume an arbitrary token from, the common channels. Moreover, in case of consumption, the state-change in the private part of \( X \) indicates which token has been consumed. This will be used in the proof of Theorem 4.27.

The last requirement indicates that the occurrence of a transition in \( X \) can be detected by considering only the private channels of \( X \). We use that in the proofs of Lemma 4.25 and Theorems 4.26 and 4.27. Without that, a counterexample for Lemma 4.25 can be constructed.

In order to show that maximal environments do exist, we construct one.

**Example 4.4 A maximal environment**
Let \( (M, CC) \) be a module with set of initial states \( L_M \) and choose a name \( \alpha \) such that \( \alpha \notin K_M \). Let \( \sim, \text{prod} \) and \( \text{cons} \) be injective functions with \( \text{dom}(\sim) = \text{dom}(\text{prod}) = \text{dom}(\text{cons}) = CC \) and furthermore:

- \( \text{rng}(\sim) \cap (K_M \cup \{\alpha\}) = \emptyset \)
- \( \text{rng}(\text{prod}), \text{rng}(\text{cons}) \) and \( P_M \) mutually disjunct.

Note that such functions can be constructed. In the sequel, we write \( \bar{c} \) instead of \( \sim(c) \), \( c \in CC \). Let \( V := \{ (\bar{c}, v) \mid c \in CC \land v \in C_M(c) \} \). We define a DES \( X \):
Replacement of Modules

\[ P_X := \text{rng}(\text{prod}) \cup \text{rng}(\text{cons}), \quad K_X := CC \cup \{ \tilde{e} \mid e \in CC \} \cup \{ \alpha \} \]
\[ C_X := C_M \downarrow CC \cup \bigcup_{e \in \text{CC}} \{ (\tilde{e}, C_M(e)) \} \cup \{ (\alpha, \emptyset) \cup V \} \]
\[ I_X := \bigcup_{e \in \text{CC}} \{ (\text{prod}(e), \{ \tilde{e} \}) \}, \{ \text{cons}(e), \{ e \} \} \}
\[ O_X := \bigcup_{e \in \text{CC}} \{ (\text{prod}(e), \{ \tilde{e}, e, \alpha \}) \}, \{ \text{cons}(e), \{ \alpha \} \} \}

and for \( e \in CC \) and \( v \in C_M(e) \),
\[ R_X(\text{prod}(e))(\{(\tilde{e}, v)\}) := \{(\tilde{e}, v), (e, v), (\alpha, \bullet)\} \quad \text{and} \]
\[ R_X(\text{cons}(e))(\{(e, v)\}) := \{(\alpha, (e, v))\} . \]

See Figure 4.3. The set of initial states of \( X \):
\[ L_X := \{ V \cup \Omega_{CC} \mid l \in L_M \} . \]

\[
\text{Environment } X
\]

\[
\text{Module } \langle M, CC \rangle
\]

\[
\text{Figure 4.3: A maximal environment.}
\]

Lemma 4.5
DES \( X \) from the above example is a maximal environment.

Proof
It can easily be checked that \( X \) is an environment of module \( \langle M, CC \rangle \). Moreover, note that for each reachable state \( s \in \hat{S}_X \), \( s \upharpoonright Q_\alpha \) is finite, where \( Q_\alpha = \{ (\alpha, v) \mid v \in C_X(\alpha) \} \), because \( s \upharpoonright Q_\alpha \) is empty at start. Furthermore, \( s \upharpoonright \{ \tilde{e} \mid e \in CC \} = V \), since prod-processors return each consumed token. We have to prove the three requirements of Definition 4.3.

i. Let \( s \in \hat{S}_X \) and \( q \in Q_{CC} \), i.e. \( q = (e, v) \) for some \( e \in CC \) and \( v \in C_M(e) \). Then \( (\tilde{e}, v) \in V \), so \( (\tilde{e}, v) \in s \) and by the construction of processor \( \text{prod}(e) \), \( (s, s') \in T_X \) where \( s' = (s \setminus \{(\tilde{e}, v)\}) \cup \{(\tilde{e}, v), (c, v), (\alpha, \bullet)\} \) and \( s' \upharpoonright Q_{CC} = s \upharpoonright Q_{CC} \cup \{ q \} . \)
ii. Let \( s \in \hat{S}_X \) and \( q \in s \uparrow Q_{CC} \), i.e. \( q = \langle c, v \rangle \) for some \( c \in CC \) and \( v \in C_M(c) \). By the construction of processor \( \text{cons}(c) \), \( (s, s') \in T_X \) where \( s' = (s \setminus \{ q \}) \cup \{ \langle \alpha, q \rangle \} \) and \( s' \uparrow Q_{CC} = s \uparrow Q_{CC} \setminus \{ q \} \).

Next, let \( t, t' \in \hat{S}_X \) such that \( (t, t') \in T_X \), \( t \uparrow (Q_X \setminus Q_{CC}) = s \uparrow (Q_X \setminus Q_{CC}) \) and \( t' \uparrow (Q_X \setminus Q_{CC}) = s' \uparrow (Q_X \setminus Q_{CC}) \). Then \( t'((\alpha, q)) = t((\alpha, q)) + 1 \), hence a token with value \( q \) has been produced for channel \( \alpha \) in the transition from \( t \) to \( t' \). The only processor that can do so is \( \text{cons}(c) \), however, \( \text{cons}(c) \) has to consume a token \( q \) from channel \( c \) in order to produce \( \langle c, v \rangle \). Hence, \( q \in t \).

iii. Let \( s, s' \in \hat{S}_X \) such that \( (s, s') \in T_X \). Let \( n \) be the number of processors involved in this transition, then \( n \geq 1 \) and \#\( s' \uparrow Q_\alpha \) = \#\( s \uparrow Q_\alpha \) + \( n \). Since \( Q_\alpha \subseteq Q_X \setminus Q_{CC} \), we have \( s \uparrow (Q_X \setminus Q_{CC}) \neq s' \uparrow (Q_X \setminus Q_{CC}) \).

\[ \Box \]

Hence, maximal environments do exist. It is even possible to construct a single maximal environment for a finite collection of modules, that have the same set of common channels. Suppose modules \( \langle M_i, CC \rangle \) be given \( (i \in \{ 1, \ldots, n \} \) for some \( n \in \mathbb{N}_1 \), then a similar approach as in Example 4.4 is appropriate. This time, \( \alpha \) is required not to be in \( K_i \) for every \( i \in \{ 1, \ldots, n \} \) and functions \( \sim \), \( \text{prod} \) and \( \text{cons} \) must satisfy \( \text{rng}(\sim) \cap (K_i \cup \{ \alpha \}) = \emptyset \). Moreover, \( \text{rng}(\text{prod}), \text{rng}(\text{cons}) \) and \( P_i \) must be mutually disjoint.

The behaviour of a maximal environment is not influenced by the module.

**Lemma 4.6**

Let \( X \) be a maximal environment of module \( \langle M, CC \rangle \).

Then

\[ \hat{S}_{X \oplus M} \uparrow Q_X = \hat{S}_X. \]

**Proof**

\[ \subseteq : \] Let \( s \in \hat{S}_{X \oplus M} \), then \( \exists l \in L_{X \oplus M} : \exists n \in \mathbb{N}_0 : l \xrightarrow{\pi}_{X \oplus M} s \). We serialize the path \( l \xrightarrow{n}_{X \oplus M} s \) into a path \( l = s_0 \xrightarrow{1} K_1 \xrightarrow{1} \ldots \xrightarrow{1} K_{n'} \xrightarrow{1} s_{n'} = s \) for some \( n \geq n, K_i \in \{ X, M \} \) such that each transition involves just one processor. We use induction.

If \( n' = 0 \) then \( s = l \) and \( l \uparrow Q_X \in L_X \subseteq \hat{S}_X \) by definition. Assume \( n' > 0 \), then \( l \xrightarrow{n' - 1}_{X \oplus M} s' \xrightarrow{1} K_{n'} \xrightarrow{1} s \) for some \( s' \in \hat{S}_{X \oplus M} \). Induction hypothesis: \( s' \uparrow Q_X \in \hat{S}_X \).

If \( K_{n'} = X \) then \( s'Q_X \in \hat{S}_X \), else \( K_{n'} = M \) and the transition from \( s' \) to \( s \) has been brought about by a processor from \( M \). Processors can produce and consume only finitely many tokens, hence \( s \uparrow Q_X = (s' \uparrow Q_X \setminus b') \cup b \) for some finite \( b', b \in B(Q_{CC}), b' \subseteq s' \uparrow Q_{CC} \). By Definition 4.3, \( s' \uparrow Q_X \xrightarrow{\#b'}_{X} s' \uparrow Q_X \setminus b' \xrightarrow{\#b} X (s' \uparrow Q_X \setminus b') \cup b = s \uparrow Q_X \), so \( s \uparrow Q_X \in \hat{S}_X \).


\[ \exists : \text{ Let } x \in \hat{S}_X, \text{ then } \exists k \in L_X : \exists n \in \mathbb{N}_0 : k \xrightarrow{n}{X} x. \text{ Since } X \text{ fit } M, \text{ we have } \\
\exists l \in L_{X \oplus M} : l \Downarrow Q_X = k, \text{ hence } l \xrightarrow{n}{X \oplus M} l(\downarrow (Q_M \setminus Q_{CC}) \cup x) \in \hat{S}_{X \oplus M} \text{ and } \\
(l(\downarrow (Q_M \setminus Q_{CC}) \cup x)) \Downarrow Q_X = x. \]

\[ \Box \]

Now we define a closure for modules. The closure of a module \( (M, CC) \) is a DES with the same topology and functionality as \( M \), but with a larger set of initial states. This set comprises everything an arbitrary environment could do.

**Definition 4.7 Closure**

Let \( (M, CC) \) be a module with a maximal environment \( X \). The closure of \( (M, CC) \), denoted as \( \overline{M} \), is the DES

\[ \langle R_M, C_M, I_M, O_M, \hat{S}_{X \oplus M} \Downarrow Q_M \rangle. \]

\[ \Box \]

Remark: The definition of \( \overline{M} \) does not depend on the choice of \( X \), i.e. each maximal environment gives the same set of initial states \( L_{\overline{M}} \). Lemma 4.10 (see below) gives a construction of \( L_{\overline{M}} \) without the use of a maximal environment.

**Definition 4.8 Semantics of a module**

The semantics of a module \( (M, CC) \) is the semantics of its closure \( \overline{M} \).

\[ \Box \]

We shall now introduce several lemmas that we need in the sequel. They all apply to a module \( (M, CC) \) with closure \( \overline{M} \).

**Lemma 4.9**

\[ \hat{S}_{\overline{M}} = L_{\overline{M}}. \]

**Proof**

\[ \hat{S}_{\overline{M}} \supseteq L_{\overline{M}} \text{ by definition. We have to prove: } \hat{S}_{\overline{M}} \subseteq L_{\overline{M}}. \text{ Let } m \in \hat{S}_{\overline{M}}, \text{ then } \exists l \in L_{\overline{M}} : \\
\exists n \in \mathbb{N}_0 : l \xrightarrow{n}{\overline{M}} m \text{ and } l \in L_{\overline{M}} \text{ means } \exists s \in \hat{S}_{X \oplus M} : l = s \Downarrow Q_M. \text{ Hence, } s \xrightarrow{n}{X \oplus M} \\
(s \Downarrow l) \cup m \in \hat{S}_{X \oplus M} \text{ and } ((s \Downarrow l) \cup m) \Downarrow Q_M = m. \]

\[ \Box \]

The next lemma indicates that \( \hat{S}_{\overline{M}} \) can be constructed as follows: Take an initial state of \( M \), add finitely many tokens to the common channels, let \( M \) run for a while and delete a finite amount of tokens from the common channels.
Lemma 4.10

\[ \mathcal{S}'_M = \{ s \mid s \in \mathcal{S}_M, b' \subseteq s \mid Q_{CC}, b' \text{ is finite } \land \exists l \in L_M : \exists b \in B(Q_{CC}) : \exists n \in \mathbb{N}_0 : b \text{ is finite } \land l \cup b \xrightarrow{n_M} s \} . \]

Proof

Let \( V \) be the set to the right of the '='-sign.

\( \subseteq \) \n
Let \( v \in \mathcal{S}'_M \). Then \( v \in L_M \) (previous lemma), so \( \exists l \in L_{X \oplus M} : \exists n \in \mathbb{N}_0 : \exists s \in \mathcal{S}'_{X \oplus M} : l \xrightarrow{n_{X \oplus M}} s \land s \mid Q_M = v \). Note that \( l \mid Q_M \in L_M \). We serialize the path \( l \xrightarrow{n_{X \oplus M}} s \) into an at least as long path \( l = s_0 \xrightarrow{1_{K_1}} \ldots \xrightarrow{1_{K_{n'}}} s' = s \) \((n' \geq n; K_i \in \{X, M\} \text{ for } i \in \{1, \ldots, n'\})\) and we define \( J := \{i \in \{1, \ldots, n'\} \mid K_i = X\} \).

For each \( j \in J \), the transition from \( s_{j-1} \) to \( s_j \) has been caused by \( \text{DES}_X \), so \( s_j \mid Q_M = (s_{j-1} \mid Q_M \backslash b_j') \cup b_j \) for some finite \( b_j', b_j \subseteq s_{j-1} \mid Q_{CC} \). We define \( b := \bigcup_{j \in J} b_j' \) and \( b := \bigcup_{j \in J} b_j \), then \( b' \) and \( b \) are finite, \( l \mid Q_M \cup b \xrightarrow{n'-|J|} m \) for some \( m \in \mathcal{S}_M', b' \subseteq m \mid Q_{CC} \) and \( m \setminus b' = s \mid Q_M = v \).

\( \supseteq \) \n
Let \( v \in V \), then \( \exists s \in \mathcal{S}_M : \exists b' \subseteq s \mid Q_{CC} : v = s \setminus b' \land b' \text{ is finite } \land \exists l \in L_M : \exists b \in B(Q_{CC}) : \exists n \in \mathbb{N}_0 : b \text{ is finite } \land l \cup b \xrightarrow{n} s \). We must prove \( v \in \mathcal{S}'_M \).

By the previous lemma, it suffices to show \( \exists k \in L_{X \oplus M} : \exists m \in \mathbb{N}_0 : \exists y \in \mathcal{S}_{X \oplus M} : k \xrightarrow{m_{X \oplus M}} y \land v = y \mid Q_M \).

Let \( l \in L_M \) and \( X \parallel M \), so \( \exists k \in L_{X \oplus M} : k \mid Q_M = l \). Then \( k \mid Q_X \in L_X \) and by Definition 4.3, \( \exists x \in \mathcal{S}_X : k \mid Q_X \xrightarrow{\#_x} x \land x \mid Q_{CC} = k \mid Q_{CC} \cup b \).

Then also \( k \xrightarrow{\#_b} k \mid (Q_M \backslash Q_{CC}) \cup x = k \mid (Q_M \backslash Q_{CC}) \cup k \mid Q_{CC} \cup b \cup x \mid (Q_X \setminus Q_{CC}) = x \mid (Q_X \setminus Q_{CC}) \cup b \cup x \xrightarrow{\#_b} x \mid (Q_X \setminus Q_{CC}) \cup s \). \( s \in \mathcal{S}_{X \oplus M} \).

By Lemma 4.6, \( (x \mid (Q_X \setminus Q_{CC}) \cup s) \mid Q_X \in \mathcal{S}_X \) and by Definition 4.3, \( (x \mid (Q_X \setminus Q_{CC}) \cup s) \mid Q_X \xrightarrow{\#_b} (x \mid (Q_X \setminus Q_{CC}) \cup s) \mid Q_X \).

Hence, \( x \mid (Q_X \setminus Q_{CC}) \cup s \xrightarrow{\#_b} x \mid (Q_X \setminus Q_{CC}) \cup (s \setminus b') = x \mid (Q_X \setminus Q_{CC}) \cup v \) and \( x \mid (Q_X \setminus Q_{CC}) \cup v \mid Q_M = v \). Finally, take for \( m \) the value \( \#_b + n + \#_b' \).

\( \Box \)

Corollary: If we take an \( s \in \mathcal{S}'_M \) and delete finitely many tokens from the common channels, then the resulting state \( s' \) is also in \( \mathcal{S}'_M \).

Corollary 4.11

\( \forall s \in \mathcal{S}'_M : \forall b \subseteq s \mid Q_{CC} : b \text{ is finite } \Rightarrow s \setminus b \in \mathcal{S}'_M . \)

\( \Box \)
We have an identical lemma in case of addition of finitely many tokens to the common channels.

Lemma 4.12

\[ \forall s \in \hat{S}_{M} : \forall b \in B(Q_{CC}) : b \text{ is finite} \Rightarrow s \cup b \in \hat{S}_{M}. \]

Proof

Let \( s \in \hat{S}_{M} \) and \( b \in B(Q_{CC}) \) with \( b \) finite. By Lemma 4.10, \( \exists l \in L_{M} : \exists b' \in B(Q_{CC}) : \exists n \in \mathbb{N}_{0} : \exists s' \in S_{M} : b'' \subseteq s' \uparrow Q_{CC} : b', b'' \text{ finite} \) and \( l \uplus b' \xrightarrow{\frac{n}{M}} s' \land s = s' \backslash b' \). Then also \( b \cup b' \text{ finite} \) and \( l \uplus b \cup b' \xrightarrow{\frac{n}{M}} s' \cup b = s \cup b \cup b'' \). Hence, \( s \cup b \in \hat{S}_{M} \) by Lemma 4.10. \( \square \)

For any environment \( X \) of \( \langle M, CC \rangle \), each reachable state of \( X \oplus M \) restricted to \( Q_{M} \) is a reachable state in the closure of \( \langle M, CC \rangle \).

Lemma 4.13

Let \( X \) be an environment of module \( \langle M, CC \rangle \). Then:

\[ \hat{S}_{X \oplus M} \upharpoonright Q_{M} \subseteq \hat{S}_{M}. \]

Proof

Let \( s \in \hat{S}_{X \oplus M} \), then \( \exists l \in L_{X \oplus M} : \exists n \in \mathbb{N}_{0} : l \xrightarrow{\frac{n}{X \oplus M}} s \). We use induction. If \( n = 0 \) then \( l = s \) and \( l \uparrow Q_{M} \in L_{M} \subseteq \hat{S}_{M} \).

Suppose \( n > 0 \), then \( l \xrightarrow{\frac{n-1}{X \oplus M}} s' \xrightarrow{\frac{1}{X \oplus M}} s \) for some \( s' \in \hat{S}_{X \oplus M} \). By Lemma 3.7, \( \exists x, x' \in S_{X} : \exists m, m' \in S_{M} : s' = x' \uplus m' \land s = x \uplus m \land \exists i, j \in \{0, 1\} : i + j \geq 1 \land x' \xrightarrow{\frac{j}{X}} x \land m' \xrightarrow{\frac{i}{M}} m \). Then \( \exists b' \subseteq x' : \exists b \in S_{X} : b', b \text{ finite} \land x = (x' \backslash b') \uplus b. \)

\[ \downarrow \text{Induction hypothesis} \downarrow \]

\[ s' \uparrow Q_{M} \in \hat{S}_{M} \]
\[ \Leftrightarrow \quad \downarrow s' = x' \uplus m' \upharpoonright \]
\[ x' \uparrow Q_{CC} \cup m' \in \hat{S}_{M} \]
\[ \Rightarrow \quad \downarrow m' \xrightarrow{\frac{i}{M}} m \upharpoonright \]
\[ x' \uparrow Q_{CC} \cup m \in \hat{S}_{M} \]
\[ \Rightarrow \quad \downarrow b' \uparrow Q_{CC} \subseteq x' \uparrow Q_{CC} \}, b' \text{ is finite}, \text{Lemma 4.11} \upharpoonright \]
\[ (x' \uparrow Q_{CC} \backslash b' \uparrow Q_{CC}) \cup m \in \hat{S}_{M} \]
\[ \Rightarrow \quad \downarrow b \text{ is finite}, \text{Lemma 4.12} \upharpoonright \]
\[ (x' \uparrow Q_{CC} \backslash b' \uparrow Q_{CC}) \cup b \uparrow Q_{CC} \cup m \in \hat{S}_{M} \]
\[ \Leftrightarrow \quad \downarrow x = (x' \backslash b') \uplus b \upharpoonright \]
\[ x \uparrow Q_{CC} \cup m \in \hat{S}_{M} \]
For a maximal environment, we have equality:

**Lemma 4.14**
Let $X$ be a maximal environment of module $(M, CC)$. Then:

$$\hat{S}_{X\otimes M} \upharpoonright Q_M = \hat{S}_M.$$ 

**Proof**
Definition 4.7 with Lemma 4.9.

Now we return to the central problem of this section (see Figure 4.1): What conditions must we impose upon modules $(A, CC)$ and $(B, CC)$ such that $Y$ and $Y'$ behave identically?

To be more precise, we are interested in conditions (predicates) $P_1, P_2$ and $P_3$ depending only on $(A, CC)$ and $(B, CC)$, such that for every environment $X$ of $(A, CC)$ and $(B, CC)$:

$$P_1(A, B, CC) \Rightarrow Y \text{ realizes } Y'$$  \hspace{1cm} (4.1)
$$P_2(A, B, CC) \Rightarrow Y \text{ simulates } Y'$$  \hspace{1cm} (4.2)
$$P_3(A, B, CC) \Rightarrow Y \text{ is equivalent with } Y'$$  \hspace{1cm} (4.3)

where $Y = X \oplus A$ and $Y' = X \oplus B$.

Two observations should be added here:

1. The problem becomes trivial when $P_1, P_2$ and $P_3$ may be chosen too strong (e.g. $P_1 = P_2 = P_3 = \text{false}$). We are interested only in the *weakest* conditions, i.e. an environment $X$ of $(A, CC)$ and $(B, CC)$ should exist such that:

$$Y \text{ realizes } Y' \Rightarrow P_1(A, B, CC)$$  \hspace{1cm} (4.4)
$$Y \text{ simulates } Y' \Rightarrow P_2(A, B, CC)$$  \hspace{1cm} (4.5)
$$Y \text{ is equivalent with } Y' \Rightarrow P_3(A, B, CC)$$  \hspace{1cm} (4.6)

where again, $Y = X \oplus A$ and $Y' = X \oplus B$.

**Lemma 4.15**
These requirements really give the *weakest* $P_i$ ($i \in \{1, 2, 3\}$).

**Proof**
Let $(A, CC)$ and $(B, CC)$ be given and assume Eqs. 4.4 and 4.(i+3) to hold, i.e.
\[
\forall X : P_i(A, B, CC) \Rightarrow X \oplus A \text{ is related to } X \oplus B,
\]
\[
\exists M : M \oplus A \text{ is related to } M \oplus B \Rightarrow P_i(A, B, CC)
\]

Moreover, assume there exists a predicate \(Q(A, B, CC)\) such that
\[
\forall X : Q(A, B, CC) \Rightarrow
\]
\[
X \oplus A \text{ is related to } X \oplus B \land (P_i(A, B, CC) \Rightarrow Q(A, B, CC))
\]

which expresses that \(Q\) is weaker or as weak as \(P_i\). To be proven \(Q \Rightarrow P_i\).

Proof: Assume \(Q(A, B, CC)\). Eq. 4.(i + 3) gives \(\exists M\). Substitution of \(M\) in the last equation gives \(M \oplus A\) is related to \(M \oplus B\) and Eq. 4.(i + 3) then yields \(P_i(A, B, CC)\).

\(\Box\)

We shall choose for \(X\) a maximal environment of \(A\) and \(B\).

2. The comparison relations between transition systems are based upon a mapping between the reachable states of both systems. A statement as ‘\(Y\) simulates \(Y'\)’ depends on a function \(f \in \hat{S}_Y \rightarrow \hat{S}_{Y'}\) that maps reachable states of \(Y\) to reachable states of \(Y'\). In order to calculate with \(f, f\) should also be applicable to non-reachable states. But not each \(f \in S_Y \rightarrow S_{Y'}\) satisfies. Tokens from the unaltered part of \(Y\) (i.e. \(X\)) should not be changed by \(f\) and tokens from \(A\) should be mapped onto \(B\). Figure 4.4 gives a graphical representation, where \(Y, Y', Q_X\) and \(Q_B\) should be substituted for \((D, D', Q_1, Q_2)\). Furthermore, at start no extra tokens may be added to the unaltered part, i.e. the slanted arrow in Figure 4.4 should not be used with initial states.

\[
Q_D : \begin{array}{c}
Q_1 \\
Q_1 \setminus Q_C C \\
Q_{C C} \\
Q_D \setminus Q_1
\end{array}
\]

\(Q_{C C} = Q_1 \cap Q_2\)

\[
f(s) = s \upharpoonright Q_1 \cup f(s \upharpoonright (Q_D \setminus Q_1)) \upharpoonright Q_2
\]

\[
Q_{D'} : \begin{array}{c}
Q_1 \setminus Q_C C \\
Q_{C C} \\
Q_2 \setminus Q_C C
\end{array}
\]

\(Q_2\)

Figure 4.4: A correct function.

We introduce a predicate ‘correct’:

**Definition 4.16 Predicate ‘correct’**

Let \(f\) be a function, \(D\) a DES with set of initial states \(L_D\) and let \(Q_1\) and \(Q_2\) be tokensets.
4 Replacement of Modules

\[ \text{correct}(f, D, Q_1, Q_2) := \]
\[ \text{dom}(f) = S_D \land \]
\[ \forall s \in S_D : f(s) = s \upharpoonright Q_1 \uplus f(s \upharpoonright (Q_D \setminus Q_1)) \upharpoonright Q_2 \land \]
\[ \forall l \in L_D : f(l) \upharpoonright Q_1 = l \upharpoonright Q_1 . \]

\[ \square \]

Lemma 4.17
If \text{correct}(f, D, Q_1, Q_2) then
\[ \forall b \in B(Q_1) : \forall s \in S_D : f(b \uplus s) = b \uplus f(s) . \]

Proof
\[ f(b \uplus s) = (b \uplus s) \upharpoonright Q_1 \uplus f((b \uplus s) \upharpoonright (Q_D \setminus Q_1)) \upharpoonright Q_2 \]
\[ = b \uplus s \upharpoonright Q_1 \uplus f(s \upharpoonright (Q_D \setminus Q_1)) \upharpoonright Q_2 \]
\[ = b \uplus f(s) . \]
\[ \square \]

We remark that the correctness of a function does not mean that the function is monotonous w.r.t. bag union. To be more specific, \text{correct}(f, D, Q_1, Q_2) does not imply \[ \forall s, t \in S_D : f(s \uplus t) = f(s) \uplus f(t) , \]
as we do not require \[ f(\emptyset) = \emptyset . \]

If a correct function is injective, then the slanted arrow in Figure 4.4 is not used:

Lemma 4.18
If \text{correct}(f, D, Q_1, Q_2) and \( f \) is injective, then
\[ \forall s \in S_D : f(s) = s \upharpoonright Q_1 \uplus f(s \upharpoonright (Q_D \setminus Q_1)) \upharpoonright (Q_2 \setminus Q_1) . \]

Proof
Assume \text{correct}(f, D, Q_1, Q_2) \land f \text{ injective.}
Let \( s \in S_D \), then \( f(s) = s \upharpoonright Q_1 \uplus f(s \upharpoonright (Q_D \setminus Q_1)) \upharpoonright Q_2 \). We define \( s' := s \upharpoonright (Q_D \setminus Q_1) \) and \( t := f(s') \upharpoonright (Q_2 \setminus Q_1) \). Then \( s' \upharpoonright Q_1 = \emptyset \) and \( f(s') = f(s') \upharpoonright Q_1 \uplus t \). We show \( f(s') \upharpoonright Q_1 = \emptyset \).
\[ f(s') = f(s') \upharpoonright Q_1 \uplus t = f(s') \upharpoonright Q_1 \uplus f(f^{-1}(t)) = \downarrow \text{Lemma 4.17} \uparrow f(f(s') \upharpoonright Q_1 \uplus f^{-1}(t)) , \]
which gives with the injectivity of \( f \), \( s' = f(s') \upharpoonright Q_1 \uplus f^{-1}(t) \). Consequently, \( f(s') \upharpoonright Q_1 \uplus f^{-1}(t) \upharpoonright Q_1 = s' \upharpoonright Q_1 = \emptyset . \)
\[ \square \]

Corollary 4.19
If \text{correct}(f, D, Q_1, Q_2) \land f \text{ is injective, then}
\[ \forall s \in S_D : f(s) \upharpoonright Q_1 = s \upharpoonright Q_1 . \]
Proof
\[ f(s)\{Q_1 = (s\{Q_1 \cup f(s\{(Q_D\{Q_1)\})\})\}Q_2\{Q_1\} = s\{Q_1 \cup \emptyset = s\{Q_1 . \]

For a given \( g \in S_A \rightarrow S_B \), we can construct an \( f \in S_Y \rightarrow S_Y \), and vice versa:

Construction 4.20
We define \( \mathcal{F} \in (S_A \rightarrow S_B) \rightarrow (S_Y \rightarrow S_Y') \).

\[ \mathcal{F} := \lambda g \in S_A \rightarrow S_B : \]
\[ (\lambda s \in S_Y : s\{Q_Y\{Q_A \} \cup g(s\{Q_A \}).\]  

Construction 4.21
We define \( \mathcal{G} \in (S_Y \rightarrow S_Y') \rightarrow (S_A \rightarrow S_B) \).

\[ \mathcal{G} := \lambda f \in S_Y \rightarrow S_Y' : \]
\[ (\lambda a \in S_A : f(a)\{Q_B \} ). \]

(Please note: \( S_A \subseteq S_Y \).

Lemma 4.22
Let \( g \in S_A \rightarrow S_B \) such that \( \text{correct}(g, A, Q_{CC}, Q_B) \). Then

- \( \text{correct}(\mathcal{F}(g), Y, Q_X, Q_B) \)
- \( \mathcal{G}(\mathcal{F}(g)) = g \).

Lemma 4.23
Let \( f \in S_Y \rightarrow S_Y' \) such that \( \text{correct}(f, Y, Q_X, Q_B) \). Then

- \( \text{correct}(\mathcal{G}(f), A, Q_{CC}, Q_B) \)
- \( \mathcal{F}(\mathcal{G}(f)) = f \).

Proof Lemma 4.22
Let \( g \in S_A \rightarrow S_B \) and assume \( \text{correct}(g, A, Q_{CC}, Q_B) \).
Define \( f := \mathcal{F}(g) = \lambda s \in S_Y : s\{Q_Y\{Q_A \} \cup g(s\{Q_A \}. So \( \text{dom}(f) = S_Y \). Let \( s \in S_Y \), then

\[ f(s) = s\{Q_Y\{Q_A \} \cup g(s\{Q_A \)
\[ = s\{Q_X\{Q_{CC} \} \cup s\{Q_{CC} \} \cup g(s\{Q_A\{Q_{CC} \})\}Q_B \]
\[ = s\{Q_X \cup (\emptyset \cup g(s\{Q_A\{Q_{CC} \})\}Q_B \]
\[ = s\{Q_X \cup (s\{Q_Y\{Q_X \}\})\{Q_Y\{Q_A \} \cup g(s\{Q_Y\{Q_X \}\}Q_A ))\}Q_B \]
\[ = s\{Q_X \cup f(s\{Q_Y\{Q_X \}Q_B . \]
Let $l \in L_Y$, then according to Definition 3.1: $\ll Q_A \in L_A$, hence $g(\ll Q_A) \ll Q_{CC} = \ll Q_{CC}$.

\[
f(l) \downarrow Q_X = \ll (Q_Y \setminus Q_A) \downarrow Q_X \cup g(\ll Q_A) \downarrow Q_X \\
= \ll (Q_X \setminus Q_A) \cup g(\ll Q_A) \downarrow Q_{CC} \\
= \ll (Q_X \setminus Q_{CC}) \cup \ll Q_{CC} \\
= \ll Q_X .
\]

Consequently, $\text{correct}(f, Y, Q_X, Q_B)$.

Let $a \in S_A$, then $(G(f))(a) = f(a) \downarrow Q_B = a \ll (Q_Y \setminus Q_A) \downarrow Q_B \cup g(a \ll Q_A) \ll Q_B = \emptyset \cup g(a) = g(a)$.

$\square$

**Proof Lemma 4.23**

Let $f \in S_Y \to S_Y$, and suppose $\text{correct}(f, Y, Q_X, Q_B)$.

Define $g := G(f) = \lambda a \in S_A : f(a) \downarrow Q_B$. So $\text{dom}(g) = S_A$. Let $a \in S_A$, then

\[
g(a) = f(a) \downarrow Q_B = a \ll Q_X \downarrow Q_B \cup f(a \ll (Q_Y \setminus Q_X)) \downarrow Q_B \\
= a \ll Q_{CC} \cup f(a \ll (Q_A \setminus Q_{CC})) \downarrow Q_B \\
= a \ll Q_{CC} \cup g(a \ll (Q_A \setminus Q_{CC})) \downarrow Q_B .
\]

Let $k \in L_A$, then $\exists x \in L_X : k \ll Q_{CC} = x \ll Q_{CC}$, because $A \NTconsistent X$. We define $l := (x \ll Q_{CC}) \cup k$, then according to Definition 3.4: $l \in L_Y$.

\[
g(k) \downarrow Q_{CC} = f(k) \downarrow Q_{CC} = k \ll Q_X \downarrow Q_{CC} \cup f(k \ll (Q_Y \setminus Q_X)) \downarrow Q_{CC} \\
= \ll Q_X \downarrow Q_{CC} \cup f(l \ll (Q_Y \setminus Q_X)) \downarrow Q_{CC} = f(l) \downarrow Q_{CC} \\
= f(l) \downarrow Q_X \downarrow Q_{CC} = \ll Q_X \downarrow Q_{CC} = \ll Q_{CC} = k \ll Q_{CC} .
\]

Hence, $\text{correct}(g, A, Q_{CC}, Q_B)$.

Let $s \in S_Y$, then

\[
(F(g))(s) = s \downarrow (Q_Y \setminus Q_A) \cup g(s \downarrow Q_A) = s \downarrow (Q_Y \setminus Q_A) \cup f(s \downarrow Q_A) \downarrow Q_B \\
= s \downarrow (Q_X \setminus Q_{CC}) \cup s \downarrow Q_A \downarrow Q_X \downarrow Q_B \cup f(s \downarrow Q_A \downarrow (Q_Y \setminus Q_X)) \downarrow Q_B \\
= s \downarrow (Q_X \setminus Q_{CC}) \cup s \downarrow Q_{CC} \cup f(s \downarrow (Q_Y \setminus Q_X)) \downarrow Q_B \\
= s \downarrow Q_X \downarrow Q_B \downarrow (Q_Y \setminus Q_X)) \downarrow Q_B \\
= f(s) .
\]

$\square$

**Lemma 4.24**

Let $g \in S_A \to S_B$ such that $\text{correct}(g, A, Q_{CC}, Q_B)$.

\[
\forall a, a' \in S_A : a \overset{g}{\Rightarrow} a' \Rightarrow a \overset{F(g)}{\Rightarrow} a' .
\]
Proof

Note that $$\forall a \in S_A : (F(g))(a) = a^f(Q_Y \setminus Q_A) \cup g(a^f Q_A) = g(a)$$ and apply the definition of $$\xrightarrow{s}$$ and $$F(g)$$.

\[ \square \]

We have a similar lemma in the case that $$X$$ is a maximal environment of module $$(A, CC)$$ and $$s \xrightarrow{f} s'$$, where $$s$$ and $$s'$$ are reachable states of $$X \oplus A$$.

Lemma 4.25

Let $$(A, CC)$$ and $$(E, CC)$$ be modules, $$X$$ a maximal environment of $$(A, CC)$$ and $$(E, CC)$$ and define $$Y := X \oplus A$$ and $$Y' := X \oplus B$$. Let $$f \in S_Y \rightarrow S_Y$$ and assume $$correct(f, Y, Q_X, Q_B)$$. Then

$$\forall s, s' \in S_Y : s \xrightarrow{f} s' \Rightarrow s^f Q_A \xrightarrow{g(f)} s'^f Q_A.$$

Proof

Define $$g := G(f)$$, let $$s, s' \in S_Y$$ and suppose $$s \xrightarrow{f} s'$$, then $$\exists n \in \mathbb{N}_0 : s_0, \ldots, s_n \in S_Y : s_0 = s \land s_n = s' \land \forall i \in \{1, \ldots, n\} : f(s_{i-1}) = f(s_i) \land (s_{i-1}, s_i) \in T_Y$$. We use induction.

If $$n = 0$$ then $$s = s'$$, hence $$s^f Q_A = s'^f Q_A$$ and $$g(s^f Q_A) = g(s'^f Q_A)$$. Next, suppose $$n > 0$$, then $$s^f Q_A \xrightarrow{g} s_{n-1}^f Q_A$$ by induction. We prove $$g(s_{n-1}^f Q_A) = g(s'^f Q_A)$$ and $$\langle s_{n-1}^f Q_A, s'^f Q_A \rangle \in T_A$$, which implies with the induction hypothesis: $$s^f Q_A \xrightarrow{f} s'^f Q_A$$.

We have $$g(s_{n-1}^f Q_A) = f(s_{n-1}^f Q_A) \cup f(s_{n-1}^f Q_A) = (s_{n-1}^f Q_Y \setminus Q_A) \cup g(s_{n-1}^f Q_A) \cup f(s_{n-1}^f Q_A)$$.

By Lemma 4.17 \( \upharpoonright f(s_{n-1}^f Q_B) = f(s'_{n-1}^f Q_B) \equiv f(s'_{n-1}^f Q_B) \equiv g(s'^f Q_A) \).

By Lemma 3.7, $$\langle s_{n-1}^f, s'_{n-1}^f \rangle \in T_Y \Rightarrow \exists x, x' \in S_X : \exists a, a' \in S_A : s_{n-1} = x \cup a \land s' = x' \cup a' \land \exists i, j \in \{0, 1\} : i + j \geq 1 \land z \xrightarrow{a_{i+1}} x' \land a_{i+1} \xrightarrow{j} a'$$. Without loss of generality, we assume $$a \cup Q_{CC}$$ to be finite (see Remark 3.8). By Lemma 4.6, $$s_{n-1}^f Q_X = x \cup a \cup Q_{CC} \in S_X$$ and by Definition 4.3, $$x \cup a \cup Q_{CC} \#(a_{i+1} Q_{CC}) x$$, so $$x, x' \in S_X$$.

Assume $$i = 1$$. Then $$\langle x, x' \rangle \in T_X$$ and $$x^f (Q_X \setminus Q_{CC}) \neq x'^f (Q_X \setminus Q_{CC})$$ by Definition 4.3. Because $$correct(f, Y, Q_X, Q_B)$$, $$f(s_{n-1})^f (Q_X \setminus Q_{CC}) = (s_{n-1}^f Q_X \cup f(s_{n-1}^f (Q_Y \setminus Q_X)) \cup f(s_{n-1}^f Q_B)) \cap (Q_X \setminus Q_{CC}) = s_{n-1}^f (Q_X \setminus Q_{CC}) = x^f (Q_X \setminus Q_{CC})$$ and $$f(s'^f) (Q_X \setminus Q_{CC}) = x'^f (Q_X \setminus Q_{CC})$$. However, $$f(s_{n-1}) = f(s')$$, so $$f(s_{n-1})^f (Q_X \setminus Q_{CC}) = f(s'^f) (Q_X \setminus Q_{CC})$$, i.e. $$x^f (Q_X \setminus Q_{CC}) = x'^f (Q_X \setminus Q_{CC})$$. Contradiction.

So $$i = 0$$, hence $$z = x \land \langle a, a' \rangle \in T_A$$ and also $$\langle z^f Q_{CC} \cup a, z^f Q_{CC} \cup a' \rangle = \langle s_{n-1}^f Q_A, s'^f Q_A \rangle \in T_A$$.

\[ \square \]

Now we have come to the first main theorem of this section.

Theorem 4.26 Realization

Let $$\overline{A}$$ and $$\overline{B}$$ be the closures of modules $$(A, CC)$$ and $$(B, CC)$$, respectively. Predicate $$P_1$$ from Eqs. 4.1 and 4.4 satisfies:
Proof
We start with Eq. 4.1.
Let \( \langle A, CC \rangle \) and \( \langle B, CC \rangle \) be modules, \( X \) an environment of \( \langle A, CC \rangle \) and \( \langle B, CC \rangle \)
and define \( Y := X \oplus A \) and \( Y' := X \oplus B \). Suppose \( P_1 \) is valid for \( g \) and define \( f := F(g) \)
(see Construction 4.20), then \( \text{correct}(f, Y, Q_X, Q_B) \) (Lemma 4.22).
We have to prove that \( Y \) realizes \( Y' \) with \( f \), that is (see page 4):

a. \( f(L_Y) \subseteq L_Y' \)
b. \( f(\hat{S}_Y) \subseteq \hat{S}_Y' \)
c. \( \forall (s, s') \in \hat{T}_Y : (f(s), f(s')) \in \hat{T}_Y' \)

a. Let \( i \in L_Y \). To be proven: \( f(l) \downarrow Q_X \in L_X \) and \( f(l) \downarrow Q_B \in L_B \).
\( f(l) \downarrow Q_X = \downarrow Q_X \in L_X \) because \( \text{correct}(f, Y, Q_X, Q_B) \) and \( f(l) \downarrow Q_B = \downarrow Q_B \in L_B \)
since \( l \downarrow Q_A \subseteq L_A \) and \( g(L_A) \subseteq L_B \).

b. Let \( t \in f(\hat{S}_Y) \), then \( \exists m \in \mathbb{N}_0 : \exists s_0, \ldots, s_m \in \hat{S}_Y : s_0 \in L_Y \land f(s_m) = t \land \forall i \in \{1, \ldots, m\} : (s_{i-1}, s_i) \in T_Y \).
We use induction. If \( m = 0 \) then \( t = f(s_0) \in f(L_Y) \subseteq L_Y' \subseteq \hat{S}_Y' \).
Next, assume \( m > 0 \). Induction hypothesis: \( f(s_{m-1}) \in \hat{S}_Y' \).
\( (s_{m-1}, s_m) \in T_Y \) implies by Lemma 3.7 \( \exists x, x' \in S_X : \exists a, a' \in S_A : s_{m-1} = x \uplus a \land s_m = x' \uplus a' \land \exists i, j \in \{0, 1\} : i + j \geq 1 \land x \begin{array}{c} \downarrow \hline \uparrow \end{array} x' \land a \begin{array}{c} \downarrow \hline \uparrow \end{array} a' \). From \( s_{m-1} \in \hat{S}_Y \) and Lemma 4.13 we conclude \( s_{m-1} \downarrow Q_A \subseteq \hat{S}_A \), that is \( x \downarrow Q_{CC} \uplus a \begin{array}{c} \downarrow \hline \uparrow \end{array} x \downarrow Q_{CC} \uplus a' \).

Since \( a \begin{array}{c} \downarrow \hline \uparrow \end{array} a' \) also \( x \downarrow Q_{CC} \uplus a \begin{array}{c} \downarrow \hline \uparrow \end{array} x \downarrow Q_{CC} \uplus a' \). \( \text{realizes } \hat{B} \) with \( g \) gives us \( \exists k \in \{0, 1\}, k \leq j : g(x \downarrow Q_{CC} \uplus a) \begin{array}{c} \downarrow \hline \uparrow \end{array} g(x \downarrow Q_{CC} \uplus a'), \text{ so } g(x \downarrow Q_{CC} \uplus a) \begin{array}{c} \downarrow \hline \uparrow \end{array} g(x \downarrow Q_{CC} \uplus a') \), too.
\( f(s_{m-1}) = f(x \uplus a) = x \downarrow (Q_X \downarrow Q_{CC}) \uplus g(x \downarrow Q_{CC} \uplus a) \begin{array}{c} \downarrow \hline \uparrow \end{array} g(x \downarrow Q_{CC} \uplus a') \begin{array}{c} \downarrow \hline \uparrow \end{array} (x \downarrow Q_{CC} \uplus a') \begin{array}{c} \downarrow \hline \uparrow \end{array} x' \uplus g(a') \).
\( x \downarrow Q_{CC} \uplus a \begin{array}{c} \downarrow \hline \uparrow \end{array} x \downarrow Q_{CC} \uplus a' \).
\( x' \uplus g(a') \).
\( \text{Lemma 4.17: } f(x' \uplus a') = f(s_{m}) = t. \)
Hence, \( f(s_{m-1}) \begin{array}{c} \downarrow \hline \uparrow \end{array} t. \) This yields, with the induction hypothesis, \( t \in \hat{S}_Y' \).

c. Let \( (s, s') \in \hat{T}_Y \), then \( s, s' \in \hat{S}_Y \) and \( f(s), f(s') \in \hat{S}_Y' \). We have to prove: \( f(s) = f(s') \) or \( (f(s), f(s')) \in \hat{T}_Y' \). If \( s = s' \) then \( f(s) = f(s') \). Assume \( s \neq s' \), then \( (s, s') \in \hat{T}_Y \). By Lemma 3.7, \( \exists x, x' \in S_X : \exists a, a' \in S_A : s = x \uplus a \land s' = x' \uplus a' \land \exists i, j \in \{0, 1\} : i + j \geq 1 \land x \begin{array}{c} \downarrow \hline \uparrow \end{array} x' \land a \begin{array}{c} \downarrow \hline \uparrow \end{array} a' \). We assume without loss of generality: \( x \downarrow Q_{CC} \) is finite (see Remark 3.8). According to Lemma 4.13, \( s \downarrow Q_A = x \downarrow Q_{CC} \uplus a \in \hat{S}_A \) and by Lemma 4.11, \( a \in \hat{S}_A \), which implies \( a' \in \hat{S}_A \). From \( a \begin{array}{c} \downarrow \hline \uparrow \end{array} a' \) and \( \text{realizes } \hat{B} \) with \( g \) we deduce \( \exists k \in \{0, 1\}, k \leq j : g(a) \begin{array}{c} \downarrow \hline \uparrow \end{array} g(a') \), which gives with Lemma 3.7: \( x \uplus g(a) \begin{array}{c} \downarrow \hline \uparrow \end{array} x' \uplus g(a'), \text{ i.e. } f(s) \begin{array}{c} \downarrow \hline \uparrow \end{array} f(s') \).
Because \( 0 \leq \max(i, k) \leq 1 \) we conclude \( f(s) = f(s') \) or \( (f(s), f(s')) \in \hat{T}_Y' \).
We continue with Eq. 4.4. Let \((A, CC)\) and \((B, CC)\) be modules. We choose for \(X\) a maximal environment of \((A, CC)\) and \((B, CC)\) and we define \(Y := X \oplus A\) and \(Y' := X \oplus B\).

Suppose \(Y\) realizes \(Y'\) with \(f \land \text{correct}(f, Y, Q_X, Q_B)\). Define \(g := G(f)\) (see Construction 4.21), then \(\text{correct}(g, A, Q_{CC}, Q_B)\) (Lemma 4.23). We shall prove that \(A\) realizes \(B\) with \(g\) and \(\overline{A}\) realizes \(\overline{B}\) with \(g\), i.e.:

\begin{align*}
&\text{a. } g(L_A) \subseteq L_B & \text{d. } g(L_{\overline{A}}) \subseteq L_{\overline{B}} \\
&\text{b. } g(S_A) \subseteq S_B & \text{e. } g(S_{\overline{A}}) \subseteq S_{\overline{B}} \\
&\text{c. } \forall (a, a') \in \hat{T}_A : (g(a), g(a')) \in \hat{T}_B & \text{f. } \forall (a, a') \in \hat{T}_{\overline{A}} : (g(a), g(a')) \in \hat{T}_{\overline{B}}
\end{align*}

We shall first prove

\[ \forall a, a' \in \hat{S}_A : (a, a') \in T_A \Rightarrow g(a) = g(a') \lor (g(a), g(a')) \in T_B \]  
\[ \text{Eq. } 4.7 \]

This result is very useful to prove the other items.

**Eq. 4.7:** Let \(a, a' \in \hat{S}_A\) and assume \((a, a') \in T_A\). By Lemma 4.14, \(\exists s \in \hat{S}_Y : a = s \upharpoonright Q_A\).

Define \(x := s \upharpoonright (Q_Y \setminus Q_A)\) and \(s' := x \uplus a',\) then \(s = x \uplus a\), \(f(s) = x \uplus g(a)\), \(f(s') = x \uplus g(a')\) and \((s, s') \in Y\). Since \(Y\) realizes \(Y'\) with \(f\), we have \(f(s) = f(s')\) or \((f(s), f(s')) \in T_{Y'}\), hence \(g(a) = g(a')\) or \((x \uplus g(a), x \uplus g(a')) \in T_{Y'}\).

We continue with the case \((x \uplus g(a), x \uplus g(a')) \in T_{Y'}\). By Lemma 3.7, \(\exists b \in g(a)\upharpoonright Q_{CC} : b \in g(a')\upharpoonright Q_{CC} : \exists i, j \in \{0, 1\}, i + j = 1: x \uplus b \rightarrow b \\
x \uplus b' \land g(a) \uplus b' \\
g(a') \uplus b'\).

Suppose \(i = 1\). Then \((x \uplus b, x \uplus b') \in T_X\) and hence \((x \uplus g(a) \upharpoonright Q_{CC}, x \uplus b' \uplus g(a) \upharpoonright Q_{CC} \cup b) \in T_X\). \(Y\) realizes \(Y'\) gives \(f(s) \in \hat{S}_Y\), i.e. \(x \uplus g(a) \upharpoonright Q_{CC} \in \hat{S}_Y\) and Lemma 4.6 then gives \(x \uplus g(a) \upharpoonright Q_{CC} \in \hat{S}_X\). With Definition 4.3 we get \((x \uplus g(a) \upharpoonright Q_{CC} \cup Q_Y \setminus Q_A) \neq (x \uplus b' \uplus g(a) \upharpoonright Q_{CC} \cup b) \upharpoonright (Q_Y \setminus Q_A \cup Q_X \setminus Q_CC);\) i.e. \(x \neq x\).

Contradiction.

Hence \(i = 0\), \(b = b'\) and \((g(a) \uplus b, g(a') \uplus b) \in T_B\). Then also \((g(a), g(a')) \in T_B\).

Now we return to a - f.

\begin{align*}
&\text{a. } \text{Let } b \in g(\hat{S}_A), \text{ then } \exists n \in N_0 : \exists a_0, ..., a_n \in \hat{S}_A : a_0 \in L_A \land g(a_n) = b \land \\
&\forall i \in \{1, ..., n\} : (a_{i-1}, a_i) \in T_A.
&\text{If } n = 0 \text{ then } b = g(a_0) \in g(L_A) \subseteq 4 \text{ see a. } \notin L_B \subseteq \hat{S}_B.
&\text{Next, assume } n > 0, \text{ then by induction } g(a_{n-1}) \in \hat{S}_B. \text{ Now } (a_{n-1}, a_n) \in T_A, \\
&\hat{S}_A \subseteq \hat{S}_A\text{ and Eq. 4.7 yields } g(a_{n-1}) = g(a_n) \lor (g(a_{n-1}), g(a_n)) \in T_B.
&\text{Hence } g(a_n) = b \in \hat{S}_B.
\end{align*}
c. Let \((a, a') \in \tilde{T}_A\). From Eq. 4.7 we deduce \(g(a) = g(a') \lor (g(a), g(a')) \in T_B\) and b. gives \((g(a), g(a')) \in \tilde{S}_B\). Hence \((g(a), g(a')) \in \tilde{T}_B\).

d. Let \(a \in L_{\overline{A}}\), then by Definition 4.7, \(\exists s \in \hat{S}_Y: a = s \uparrow Q_A\). Then \(f(s) \in \hat{S}_Y\), because \(Y\) realizes \(Y'\) with \(f\) and hence \(f(s) \uparrow Q_B \in L_B\). Now \(f(s) \uparrow Q_B = \dagger\) Lemma 4.23 \(\dagger s \uparrow (Q_Y \setminus Q_A) \uparrow Q_B \cup f(s \uparrow Q_A) = \emptyset \cup g(a) = g(a)\).

e. Combine Lemma 4.9 and d.

f. Let \((a, a') \in \tilde{T}_A\). Eq. 4.7 gives \(g(a) = g(a') \lor (g(a), g(a')) \in T_B\) and e. gives \((g(a), g(a')) \in \tilde{S}_B\). Hence \((g(a), g(a')) \in \tilde{T}_B\).

\[
\square
\]

**Theorem 4.27 Simulation**

Let \(\overline{A}\) and \(\overline{B}\) be the closures of modules \(\langle A, CC \rangle\) and \(\langle B, CC \rangle\), respectively. Predicate \(P_2\) from Eqs. 4.2 and 4.5 satisfies:

\[
P_2 = \exists g \in S_A \rightarrow S_B, \text{ correct}(g, A, Q_{CC}, Q_B) : \\
A \text{ simulates } B \text{ with } g \land \overline{A} \text{ simulates } \overline{B} \text{ with } \overline{g} \land \\
\forall a \in \tilde{S}_A : \forall b \subseteq g(a) \uparrow Q_{CC} : \text{ \(b\) is finite } \Rightarrow \\
\exists a' \in \tilde{S}_A : b \subseteq a' \uparrow Q_{CC} \land a \Rightarrow a'.
\]

The last part of \(P_2\) states: If a finite amount of tokens is mapped by \(g\) onto the common channels, then those tokens are already there or they can be transported there within the same class.

**Proof**

We start with Eq. 4.2.

Let \(\langle A, CC \rangle\) and \(\langle B, CC \rangle\) be modules, \(X\) an environment of \(\langle A, CC \rangle\) and \(\langle B, CC \rangle\) and define \(Y := X \oplus A\) and \(Y' := X \oplus B\). Suppose \(P_2\) is valid for \(g\) and define \(f := F(g)\) (see Construction 4.20), then \(\text{correct}(f, Y, Q_X, Q_B)\) (Lemma 4.22).

We have to prove that \(Y\) simulates \(Y'\) with \(f\). From the previous theorem we know that \(Y\) realizes \(Y'\) with \(f\). What remains to prove is:

\[\begin{align*}
a. & \quad L_Y \subseteq f(L_Y) \\
b. & \quad \hat{S}_Y \subseteq f(\hat{S}_Y) \\
c. & \quad \forall (t, t') \in \tilde{T}_Y : \forall s_0 \in f^{-1}(t) \cap \hat{S}_Y : \exists n \in \mathbb{N}_0 : \exists s_1, \ldots, s_n \in f^{-1}(t) : \\
& \exists s_{n+1} \in f^{-1}(t') : \forall i \in \{0, \ldots, n\} : (s_i, s_{i+1}) \in \tilde{T}_Y
\end{align*}\]

a. Let \(l' \in L_Y\), then \(l' \uparrow Q_X \in L_X\) and \(l' \uparrow Q_B \in L_B\). Since \(g(L_A) = L_B\), a \(k \in L_A\) exists such that \(g(k) = l' \uparrow Q_B\). We define \(l := l' \uparrow (Q_X \setminus Q_{CC}) \cup k\) and we show that
f(l) = l' and l ∈ L_Y, that is l\|Q_X ∈ L_X and l\|Q_A ∈ L_A.
f(l) = l\|Q_Y \setminus Q_A \cup g(l\|Q_A) = l'\|Q_X \setminus Q_{CC} \cup g(k) = l''\|Q_X \setminus Q_{CC} \cup l''\|Q_B = l'';
\text{I} l\|Q_X = l''\|Q_X \setminus Q_{CC} \cup k \in Q_{CC} = \notin \text{correct}(g, Q_{CC}, Q_B) \notin l''\|Q_X \setminus Q_{CC} \cup g(k) = l''\|Q_X \setminus Q_{CC} \cup l''\|Q_B = l''\|Q_X \in L_X;
\text{I} l\|Q_A = k \in L_A.

b. Let \( t \in \hat{S}_Y \), then \( \exists n \in \mathbb{N}_0 : \exists l' \in L_Y : l' \rightarrow^* \hat{s} \rightarrow t \). To be proven: \( \exists s \in \hat{S}_Y : f(s) = t \). We use induction.

For \( n = 0 \), see a. Assume \( n > 0 \), then \( \exists l' \in \hat{S}_Y : l' \rightarrow^* \hat{s} \rightarrow t \). Induction hypothesis: \( \exists s' \in \hat{S}_Y : f(s') = t', i.e. s'\|Q_X \setminus Q_{CC} = t'\|Q_X \setminus Q_{CC} \) and \( g(s'\|Q_A) = t'\|Q_B \).

By Lemma 3.7, \( \exists x', b \in S_B : t' = x' \cup b' \land t = x \cup b \land \exists i, j \in \{0, 1\} : i + j \geq 1 \land x' \rightarrow^{i,j} x \land b' \rightarrow^{i,j} b \). Without loss of generality we assume \( x'\|Q_B \) to be finite (see Remark 3.8).

By Lemma 4.13, \( s'\|Q_A \in \hat{S}_A \) and since \( g(\hat{S}_A) = \hat{S}_B \), \( g(s'\|Q_A) \in \hat{S}_B \).

With \( A \) simulates \( B \) with \( g \) we derive \( \exists a \in S_A : \exists n' \in \mathbb{N}_0 : s'\|Q_A \rightarrow^{n'} a \land g(a) = x'\|Q_{CC} \cup b \). So \( a \in \hat{S}_A \).

The last part of \( P_2 \) gives \( \exists a' \in \hat{S}_A : x'\|Q_{CC} \subseteq a'\|Q_{CC} \land a \rightarrow^{\alpha} a' \).

Then \( g(a) = g(a') \) and \( \exists m \in \mathbb{N}_0 : a \rightarrow^{\alpha} a' \). Hence, \( s'\|Q_A \rightarrow a' \land s'\|Q_{CC} \cup a' = x'\|Q_X \setminus Q_{CC} \cup (x'\|Q_{CC} \cup a'\|Q_{CC}) = x'\|Q_X \setminus Q_{CC} \).

Note that \( g(\hat{S}_A) = \hat{S}_B \) and the last part of \( P_2 \) gives \( \exists a \in \hat{S}_A : x\|Q_{CC} \subseteq a\|Q_{CC} \land s_0\|Q_A \rightarrow^* a \). We define \( a_0 := a|x\|Q_{CC} \), then \( g(a_0) = g(a)\|Q_{CC} = g(s_0\|Q_A)\|Q_{CC} = b \).

We distinguish two cases:

1. \( b = b' \).

   Assume \( i = 0 \), then \( x = x' \), hence \( t = x \cup b = x' \cup b' = t' \). Contradiction.

   So \( i = 1 \) and \( \langle x, x' \rangle \in T_X \). We define \( s := x \cup a_0 \) and \( s' := x' \cup a_0 \), then:

   - By Lemma 4.24, \( s_0 \rightarrow s_0 \text{\|Q}_{XQCC} \cup a = x \|QXQCC} \cup a = x \cup (a\|Q_{CC}) = s \).
   - \( s \rightarrow s' \).
   - \( f(s') = x' \cup g(a_0) = x' \cup b = x' \cup b' = t' \).

2. \( b \neq b' \). Then \( j = 1 \) and \( \langle b, b' \rangle \in T_B \).

   By Lemma 4.11, \( a_0 \in \hat{S}_A \) and with \( A \) simulates \( B \) with \( g \) we get \( \exists a' \in g^{-1}(b) \):
\exists a'' \in g^{-1}(b') : \exists k \in \{0,1\} : a_0 \xrightarrow{g} a' \xrightarrow{k A} a''$. The case $k = 0$ leads to a contradiction: If $k = 0$ then $g(a') = g(a'')$, i.e. $b = b'$. Hence $a_0 \xrightarrow{g} a' \xrightarrow{A} a''$. Then also $x \upharpoonright Q_{CC} \uplus a_0 = a \xrightarrow{g} x \upharpoonright Q_{CC} \uplus a'$ and $s_0 \upharpoonright Q_A \xrightarrow{g} x \upharpoonright Q_{CC} \uplus a'$ and with Lemma 4.24, $s_0 \xrightarrow{f} s_0 \upharpoonright (Q_X \setminus Q_{CC}) \uplus x \upharpoonright Q_{CC} \uplus a' = x \uplus a'$.

We define $s := x \uplus a'$ and $s' := x' \uplus a''$, then:

- $x \xrightarrow{f} s$
- $x \xrightarrow{f} x'$ and $a' \xrightarrow{A} a''$ results with Lemma 3.7 in $s \xrightarrow{Y} s'$
- $f(s') = x' \uplus g(a'') = x' \uplus b' = t'$.

We continue with Eq. 4.5.

Let $(A,CC)$ and $(B,CC)$ be modules. We choose for $X$ a maximal environment of $(A,CC)$ and $(B,CC)$ and we define $Y := X \oplus A$ and $Y' := X \oplus B$.

Suppose $Y$ simulates $Y'$ with $f \land \text{correct}(f,Y,Q_X,Q_B)$. Define $g := G(f)$ (see Construction 4.21), then $\text{correct}(g,A,Q_{CC},Q_B)$ (Lemma 4.23). From the previous theorem we know that $P_1$ is valid. What remains to prove is:

a. $L_B \subseteq g(L_A)$

b. $\hat{S}_B \subseteq g(\hat{S}_A)$

c. $\forall (b,b') \in \hat{T}_B : \forall a_0 \in g^{-1}(b) \cap \hat{S}_A : \exists n \in N_0 : \exists a_1, \ldots, a_n \in g^{-1}(b) : \exists a_{n+1} \in g^{-1}(b') : \forall i \in \{0, \ldots, n\} : (a_i, a_{i+1}) \in \hat{T}_A$

d. $L_B \subseteq g(L_A)$

e. $\hat{S}_B \subseteq g(\hat{S}_A)$

f. $\forall (b,b') \in \hat{T}_B : \forall a_0 \in g^{-1}(b) \cap \hat{S}_A : \exists n \in N_0 : \exists a_1, \ldots, a_n \in g^{-1}(b) : \exists a_{n+1} \in g^{-1}(b') : \forall i \in \{0, \ldots, n\} : (a_i, a_{i+1}) \in \hat{T}_A$

g. $\forall a \in \hat{S}_A : \forall b \subseteq g(a) \upharpoonright Q_{CC} : b$ is finite $\Rightarrow \exists a' \in \hat{S}_A : b \subseteq a' \upharpoonright Q_{CC} \land a \xrightarrow{g} a'$

a. Let $b \in L_B$. We construct a $k \in L_A$ with $g(k) = b$.

Since $X \upharpoonright B$, $\exists x \in L_X : x \upharpoonright Q_{CC} = b \upharpoonright Q_{CC}$. We define $y := (x \setminus x \upharpoonright Q_{CC}) \uplus b$, then $y \in L_Y$ by the definition of $\uplus$. Because $Y$ simulates $Y'$ with $f$, there is an $l \in L_Y$ such that $f(l) = y$. We define $k := l \upharpoonright Q_A$, then $k \in L_A$ and $g(k) = f(k) \upharpoonright Q_B = (l \upharpoonright (Q_X \setminus Q_{CC}) \cup f(k)) \upharpoonright Q_B = (l \upharpoonright (Q_Y \setminus Q_A) \cup f(l \upharpoonright Q_A)) \upharpoonright Q_B = f(l) \upharpoonright Q_B = y \upharpoonright Q_B = b$.

b. Let $b \in \hat{S}_B$, then $\exists n \in N_0 : \exists b_0 \in L_B : b_0 \xrightarrow{A} b$. To be proven $\exists a \in \hat{S}_A : g(a) = b$.

We use induction.

If $n = 0$ then $b_0 = b$: See a.

Assume $n > 0$, then $\exists b' \in \hat{S}_B : b_0 \xrightarrow{A} b' \xrightarrow{B} b$. Induction hypothesis: $\exists a' \in \hat{S}_A : g(a') = b'$.

Since $\hat{S}_A \subseteq \hat{S}_A$, we have with Lemma 4.14: $\exists s' \in \hat{S}_A : s' \upharpoonright Q_A = a'$. We define
4 Replacement of Modules

Let \( g \) be \( (x') \equiv g((a')) = x' \cup b' \). Because \( Y \) simulates \( Y' \) with \( f \), we have \( f(s') \in \hat{S}_Y \), i.e. \( x' \cup b' \in \hat{S}_Y \). Moreover, \( x' \cup b' \stackrel{f}{\rightarrow} x \cup b \) and \( Y \) simulates \( Y' \) with \( f \) gives \( \exists b'' \in \hat{S}_Y : s' \stackrel{f}{\leftrightarrow} b'' \wedge f(s) = x' \cup b \).

By Lemma 4.25, \( s'' \mid Q_A = a' \Rightarrow s'' \mid Q_A \in \hat{S}_A \) and by Lemma 3.7, \( \exists x'' \in X \), \( x \in X \) : \( \exists a'', a \in S_A : s'' = x'' \cup a'' \wedge b \in \cap a'' \in S_A \). Since correct \( f(Y, Q_X, Q_B, f(s'') \mid (Q_X \mid QC)) = s'' \mid (Q_X \mid QC) \), i.e. \( x'' \mid (Q_X \mid QC) = x'' \mid (Q_X \mid QC) \).

Similarly, \( f(s) \mid (Q_X \mid QC) = s \mid (Q_X \mid QC) \), i.e. \( x' \mid (Q_X \mid QC) = \hat{x} \mid (Q_X \mid QC) \).

Hence \( x'' \mid (Q_X \mid QC) = \hat{x} \mid (Q_X \mid QC) \).

Suppose \( i = 1 \), then \( x'', x \in T_X \), so \( x'' \cup a'' \mid Q_C = x \cup a'' \mid Q_C \in T_X \), too.

By Lemma 4.6, \( s'' \mid Q_X = x'' \cup a'' \mid Q_C \in \hat{S}_X \) and with Definition 4.3 we get \( x'' \cup a'' \mid Q_C \mid (Q_X \mid QC) = (x \cup a'' \mid Q_C) \mid (Q_X \mid QC) \), i.e. \( x'' \mid (Q_X \mid QC) \neq \hat{x} \mid (Q_X \mid QC) \).

Contradiction, so \( i = 0 \), hence \( x'' = x \) and \( a'' \stackrel{\alpha}{\leftrightarrow} a \).

Then also \( a'' \cup x'' \mid Q_C \stackrel{\alpha}{\leftrightarrow} a \cup x \mid Q_C \), i.e. \( s'' \mid Q_A \stackrel{\alpha}{\leftrightarrow} s \mid Q_A \) and \( g(s \mid Q_A) = (s \mid Q_X \mid Q_C) \cup (g(s \mid Q_C)) \mid Q_B = \neq \hat{x} \mid Q_B = 0 \hat{b} \).

c. Let \( \langle b, b' \rangle \in \hat{T}_B \) and \( a_0 \in g^{-1}(b) \cap \hat{S}_A \). If \( b = b' \) then we are through. Assume \( b \neq b' \), then \( \langle b, b' \rangle \in \hat{T}_B \) and with the proof of b. we get \( \exists x \in X \) : \( \exists a, a' \in S_A : a_0 \Rightarrow x \mid Q_C = a \wedge x \mid Q_C = a' \) \( \wedge x \mid Q_C = a \) \( \wedge x \mid Q_C = a' \) \( \wedge g(x \mid Q_C) = b' \) which implies \( \exists a \in \hat{C}_B : \forall a \in \hat{C}_B : \forall a \in \hat{C}_B : a_i \in \hat{C}_B \) : \( (a_i, a_{i+1}) \in T_A \). Since \( a_0 \in S_A \) we have \( \langle a_i, a_{i+1} \rangle \in \hat{T}_A \) for all \( i \in \{0, \ldots, n\} \).

d. \( L_B = \neq \hat{S}_B = \neq \hat{S}_Y \mid B = \neq Y \) simulates \( Y' \) with \( f \), hence \( f(\hat{S}_Y) = \hat{S}_Y \), \( f(\hat{S}_Y) \mid B = f(s) \mid Q_B \mid s \in \hat{S}_Y \) = \( \neq \hat{S}_A \).

\( \{g(s) \mid Q_A \mid s \in \hat{S}_Y \} = g(\hat{S}_Y) \mid Q_A = \neq \hat{L}_A \).

e. See d.

f. Similar to c. (Replace \( \hat{S}_A, \hat{T}_A, \hat{S}_B \) by \( \hat{A}, \hat{T}_A, \hat{S}_B \) and \( \hat{T}_B \), respectively.)

g. Let \( a \in \hat{S}_A \), \( b \subseteq g(a) \mid Q_C \) and assume: \( b \) is finite. We use induction to the number of elements in \( b \backslash a \mid Q_C \).

If \( b \subseteq a \mid Q_C \), then we are through since \( \Rightarrow \) is reflexive.

Next, assume \( \exists q \in b : q \notin a \mid Q_C \) and define \( b' := b \backslash \{q\} \). Induction hypothesis: \( \exists a' \in \hat{S}_A : b' \subseteq a' \mid Q_C \wedge a \Rightarrow a' \).

By Corollary 4.11, \( a' \mid b' \in \hat{S}_A \). Please note: \( q \in g(a' \backslash b') \mid Q_C \), since \( g(a' \backslash b') = \{q\} \cup (g(a) \backslash b) \). Lemma 4.14: \( \exists s_0 \in \hat{S}_Y : s_0 \mid Q_A = a' \backslash b' \). Then also \( q \in f(s_0) \mid Q_C \), because \( f(s_0) = \chi(X_0 \mid Q_C) \cup g(a) \backslash b \). Lemma 4.6 yields \( f(s_0) \mid Q_X \in \hat{S}_X \) and with Definition 4.3 we get \( \exists x \in X \) : \( (f(s_0) \mid Q_X, x) \in T_X \wedge x \mid Q_C = f(s_0) \mid Q_C \backslash \{q\} = g(a' \backslash b') \mid Q_C \backslash \{q\} \) and \( f(s_0) \mid Q_X \mid Q_C = s_0 \mid Q_X \mid Q_C \neq x \mid Q_X \mid Q_C \).

We define \( t := x \mid Q_X \mid Q_C \cup (g(a' \backslash b') \cup \{q\}) \), then \( f(s_0), x \cup f(s_0) \mid (Q_B \mid Q_C) = (f(s_0), t) \in T_Y \).
4 Replacement of Modules

Y simulates \( Y' \) with \( f \) implies \( \exists n \in \mathbb{N}_0 : \exists s_1, \ldots, s_n \in f^{-1}(f(s_0)) : \exists s_{n+1} \in f^{-1}(t) : \forall i \in \{1, \ldots, n\} : (s_{i-1}, s_i) \in T_Y \), i.e. \( s_0 \xrightarrow{f} s_n \xrightarrow{f^{-1}} s_{n+1} \). (The case \( s_n = s_{n+1} \) fails, because that would imply \( f(s_n)\upharpoonright(Q_X \setminus QC) = f(s_{n+1})\upharpoonright(Q_X \setminus QC) \), which can be rewritten to \( f(s_0)\upharpoonright(Q_X \setminus QC) = f(s_{n+1})\upharpoonright(Q_X \setminus QC) \) what contradicts the above.)

By Lemma 4.25, \( s_0 \upharpoonright QA = a' \Leftrightarrow s_n \upharpoonright QA \). Then also \( a' \xrightarrow{g} s_n \upharpoonright QA \) and \( a \xrightarrow{g} s_n \upharpoonright QA \). We show that \( q \in s_n \upharpoonright QC \), which completes the proof.

Lemma 3.7 gives \( \exists x_n, x_{n+1} \in S_X : \exists a_n, a_{n+1} \in S_A : s_n = x_n \uplus a_n \wedge s_{n+1} = x_{n+1} \uplus a_{n+1} \wedge \exists i, j \in \{0, 1\}, i + j \geq 1 : x_n \xrightarrow{\chi} x_{n+1} \wedge a_n \xrightarrow{\chi} a_{n+1} \).

Suppose \( i = 0 \), then \( s_n \upharpoonright(Q_X \setminus QC) = s_{n+1} \upharpoonright(Q_X \setminus QC) \), i.e. \( s_0 \upharpoonright(Q_X \setminus QC) = \chi(Q_X \setminus QC) \). Contradiction, hence \( i = 1 \) and \( (x_n, x_{n+1}) \in T_X \) and \( (x_n \uplus a_n \upharpoonright QC, x_{n+1} \uplus a_n \upharpoonright QC) \in T_X \), too. Lemma 4.6: \( s_n \upharpoonright QX \in S_X \), i.e. \( x_n \uplus a_n \upharpoonright QC \in S_X \).

Note that \( (x_n \uplus a_n \upharpoonright QC)\upharpoonright(Q_X \setminus QC) = (f(s_0)\upharpoonright QX)\upharpoonright(Q_X \setminus QC) \) and \( (x_{n+1} \uplus a_n \upharpoonright QC)\upharpoonright(Q_X \setminus QC) = \chi(Q_X \setminus QC) \), hence by Definition 4.3, \( q \in (x_n \uplus a_n \upharpoonright QC)\upharpoonright QC = s_n \upharpoonright QC \).

\( \square \)

**Theorem 4.28** Equivalence

Let \( A \) and \( B \) be the closures of modules \( \langle A, CC \rangle \) and \( \langle B, CC \rangle \), respectively. Predicate \( P_3 \) from Eqs. 4.3 and 4.6 satisfies:

\[
P_3 = \exists g \in S_A \rightarrow S_B, \text{correct}(g, A, QC, QB):
A \text{ is equivalent with } B \ \text{w.r.t.} \ g \ \text{and } \overline{A} \text{ is equivalent with } \overline{B} \ \text{w.r.t.} \ g.
\]

**Proof**

First, note that \( \text{correct}(g, A, QC, QB) \) implies \( P_3 = (P_2 \text{ with an injective } g) \). Because of the injectivity of \( g \), the last part of \( P_2 \) cancels out (see Corollary 4.19, \( \xrightarrow{g} \) is reflexive).

We start with Eq. 4.3.

Let \( \langle A, CC \rangle \) and \( \langle B, CC \rangle \) be modules, \( X \) an environment of \( \langle A, CC \rangle \) and \( \langle B, CC \rangle \) and define \( Y := X \oplus A \) and \( Y' := X \oplus B \). Suppose \( P_3 \) applies for \( g \). Then \( P_2 \) is valid.

We have to prove that \( Y \) is equivalent with \( Y' \), or (see Lemma 2.7):

An injective \( f \) exists such that \( Y \) simulates \( Y' \) with \( f \).

Again, \( f \) will be \( F(g) \) (see Construction 4.20), so \( \text{correct}(f, Y, Qx, QB) \) (Lemma 4.22).

The previous theorem tells us that \( Y \) simulates \( Y' \) with \( f \). To prove the injectivity of \( f \), let \( s, s' \in SY \) and assume \( f(s) = f(s') \). We show \( s\upharpoonright QA = s'\upharpoonright QA \) and \( s\upharpoonright(QY \setminus QA) = s'\upharpoonright(QY \setminus QA) \).

\( f(s) = f(s') \) implies \( f(s)\upharpoonright QB = f(s')\upharpoonright QB \), i.e. \( (s\upharpoonright QY \setminus QA)\uplus g(s\upharpoonright QA)\upharpoonright QB = g(s'\upharpoonright QA) = g(s'\upharpoonrightQA) \). With the injectivity of \( g \) we get \( s\upharpoonright QA = s'\upharpoonright QA \);

\( s\upharpoonright(QY \setminus QA) = (s\upharpoonright(QY \setminus QA)\uplus g(s\upharpoonright QA))\upharpoonright QY \setminus QA = f(s)\upharpoonright(QY \setminus QA) = f(s')\upharpoonright(QY \setminus QA) = s'\upharpoonright(QY \setminus QA) \).
We continue with Eq. 4.6.

Let \( (A, CC) \) and \( (B, CC) \) be modules. We choose for \( X := X \oplus A \) and \( Y' := X \oplus B \).

Assume an injective \( f \) exists such that \( Y \) simulates \( Y' \) with \( f \) (i.e. \( Y \) is equivalent with \( Y' \)) and \( \text{correct}(f, Y, Q_X, Q_B) \). Define \( g := G(f) \) (see Construction 4.21), then \( \text{correct}(g, A, Q_{CC}, Q_B) \) (Lemma 4.23). From the previous theorem we know that \( P_2 \) is valid. What remains to prove for \( P_3 \) is:

\[ g \text{ is injective}. \]

Let \( a, a' \in S_A \) and suppose \( g(a) = g(a') \), i.e. \( f(a) \uparrow Q_B = f(a') \uparrow Q_B \). Now \( f(a) = \notin \text{correct}(f, Y, Q_X, Q_B) \notin a \uparrow (Q_Y \setminus Q_X) \uplus f(a \uparrow Q_A) \uparrow Q_B = \emptyset \uplus f(a) \uparrow Q_B = f(a) \uparrow Q_B = f(a') \uparrow Q_B = f(a') \). With the injectivity of \( f \) we get \( a = a' \).

\[ \square \]

Above theorems have shown

I. \( P_1(A, B, CC) = A \) realizes \( B \land \overline{A} \) realizes \( \overline{B} \)

II. \( P_2(A, B, CC) \Rightarrow A \) simulates \( B \land \overline{A} \) simulates \( \overline{B} \)

III. \( P_3(A, B, CC) = A \) is equivalent with \( B \land \overline{A} \) is equivalent with \( \overline{B} \)

where \( P_2 \) is really stronger than the assertion ‘\( A \) simulates \( B \land \overline{A} \) simulates \( \overline{B} \).’ We give an example where \( A \) simulates \( B, \overline{A} \) simulates \( \overline{B} \) and not \( P_2(A, B, CC) \).

**Example 4.29**

Let \( A = \{(p, \{(c, \bullet), \{(d, \bullet)\})\}, \{(c, \bullet), \{(d, \bullet)\}, \{(p, \{c\}), \{(p, \{d\})\}, \{\emptyset\})\} \) be a DES with two channels \( c \) and \( d \) and a processor \( p \) that can move tokens from \( c \) to \( d \).

Let \( B = \emptyset, \{(c, \bullet)\}, \emptyset, \emptyset, \{\emptyset\} \) be a DES with a single channel \( c \). Both DES’ses can be initiated with no tokens at all.

Let \( \{c\} \) be the set of common channels, then we have modules \( (A, \{c\}) \) and \( (B, \{c\}) \) as in the figure below.

![Diagram of DES modules](image_url)

\[ \text{Module } (A, \{c\}) \quad \text{Module } (B, \{c\}) \]

**Remark.** For reasons not clear to us, all Petri Net models described in [4] forbid isolated nodes, e.g. see [4, page 30]. Thus \( B \) would not be properly defined, as it consists of a single isolated channel.
Now $A$ simulates $B$ and $\overline{A}$ simulates $\overline{B}$ with the following function $g \in S_A \rightarrow S_B$ that maps all tokens of channel $d$ to channel $c$:

$$g(\lambda \langle c, \cdot, m \rangle, \langle d, \cdot, n \rangle) \in S_A : \{(c, \cdot), m + n\}.$$

Note that $S_A = \mathcal{W}(\{(c, \cdot), (d, \cdot)\})$, $S_B = \mathcal{W}(\{(c, \cdot)\})$, $L_A = L_B = \hat{S}_A = \hat{S}_B = \emptyset$, $g(L_A) = L_B$, $\text{correct}(g, A, \{(c, \cdot)\}, Q_B)$.

$L_{\overline{A}} = \hat{S}_{\overline{A}} = \{(c, \cdot), m, (d, \cdot), n\} \mid m, n \in \mathbb{N}_0$, $L_{\overline{B}} = \hat{S}_{\overline{B}} = \{(c, \cdot), n\} \mid n \in \mathbb{N}_0$, $g(\hat{S}_{\overline{A}}) = \hat{S}_{\overline{B}}$.

$T_A = \{(c, \cdot, m + 1), (d, \cdot, n), (c, \cdot, m), (d, \cdot, n + 1)\} \mid m, n \in \mathbb{N}_0 \cup \{\infty\}$, $T_B = \emptyset$, $\bar{T}_A = \bar{T}_B = \{(\emptyset, \emptyset)\}$, $\hat{T}_A = \{t, t' \in \hat{S}_A \times \hat{S}_A \mid t = t' \vee \langle t, t' \rangle \in T_A \}$ and $\hat{T}_B = \{\langle t, t' \rangle \mid t \in \hat{S}_B\}$.

The assertions ‘$A$ simulates $B$ with $g$’ and ‘$\overline{A}$ simulates $\overline{B}$’ are easily verified.

The negation of the last part of $P_2$ corresponds to

$$\exists a \in \hat{S}_{\overline{A}} : \forall b \subseteq g(a) \{\{c, \cdot\} : b \text{ is finite } \wedge$$

$$\forall a' \in \hat{S}_{\overline{A}} : a \xrightarrow{\overline{g}} a' \Rightarrow \neg(b \subseteq a' \{\{c, \cdot\}\}).$$

Choose $a = \{(d, \cdot)\}$ and $b = g(a) = \{(c, \cdot)\}$, then the only $a' \in \hat{S}_{\overline{A}}$ that satisfies $a \xrightarrow{\overline{g}} a'$ is $a$ itself, but certainly $b$ is not a sub-bag of $\emptyset = a \{\{c, \cdot\}\}$.

We think that the reason of this asymmetry has to do with the definition of the simulation relation, which has actually been defined too strong. See also page 6.

When we intend to build a huge DES, we usually start with a global definition and apply a kind of top-down strategy in order to reach the intended level of detail. Actually, we repeatedly select a module of the DES at hand and replace it with a more detailed one. Next lemma can be used for this procedure. It gives a sufficient condition to safely replace a module with another one.

**Lemma 4.30**

Let $\langle A, CC \rangle$ and $\langle B, CC \rangle$ be modules and let $X$ be an environment of $\langle A, CC \rangle$ and $\langle B, CC \rangle$. Suppose $X$ is connected to either $A$ or $B$.

If $P_2(A, B, CC)$ then $X$ can impossibly determine which of $A$ and $B$ it has been connected to.

**Proof**

Suppose $X$ can distinguish between $A$ and $B$. Then either a reachable state $s \in \hat{S}_{X \oplus A}$ exists such that $s \downarrow (Q_X \setminus Q_{CC}) \notin \hat{S}_{X \oplus B} \downarrow (Q_X \setminus Q_{CC})$ or there is an $s \in \hat{S}_{X \oplus B}$ with $s \downarrow (Q_X \setminus Q_{CC}) \notin \hat{S}_{X \oplus A} \downarrow (Q_X \setminus Q_{CC})$.

We shall prove that $\hat{S}_{X \oplus A} \downarrow (Q_X \setminus Q_{CC}) = \hat{S}_{X \oplus B} \downarrow (Q_X \setminus Q_{CC})$, hence such an $s$ cannot exist.

Let $g$ satisfy $P_2$ and define $f := \mathcal{F}(g)$. Then $X \oplus A$ simulates $X \oplus B$ with $f$, so $f(\hat{S}_{X \oplus A}) = \hat{S}_{X \oplus B}$. Then
\[
\hat{S}_{X \oplus B} \updownarrow (Q_X \setminus QC) = \{ t(\hat{t}(Q_X \setminus QC) \mid t \in f(\hat{S}_{X \oplus A})) = \{ f(s) \uparrow (Q_X \setminus QC) \mid s \in \hat{S}_{X \oplus A} \} = \\
\{ (s(\hat{t}(Q_X \setminus QC) \uparrow g(s(\hat{t}(Q_A)))) \uparrow (Q_X \setminus QC) \mid s \in \hat{S}_{X \oplus A} \} = \\
\{ s(\hat{t}(Q_X \setminus QC)) \uparrow \emptyset \mid s \in \hat{S}_{X \oplus A} \} = \hat{S}_{X \oplus A} \uparrow (Q_X \setminus QC).
\]

\[
\hat{S}_{X \oplus B} \updownarrow (Q_X \setminus QC) = \{ t(\hat{t}(Q_X \setminus QC) \mid t \in f(\hat{S}_{X \oplus A})) = \{ f(s) \uparrow (Q_X \setminus QC) \mid s \in \hat{S}_{X \oplus A} \} = \\
\{ s(\hat{t}(Q_X \setminus QC)) \uparrow \emptyset \mid s \in \hat{S}_{X \oplus A} \} = \hat{S}_{X \oplus A} \uparrow (Q_X \setminus QC).
\]

The above result can easily be generalized.

Next corollary gives, for a module \( M \) and an environment \( X \), a set \( U \) of modules with \( M \in U \) such that \( X \) cannot observe a difference between modules of \( U \).

**Corollary 4.31**

Let \( X \) be an environment of module \( (D, CC) \) and let \( U \) be the smallest set satisfying:

- \( (D, CC) \in U \);
- For all DES's \( D' \) with \( K_{D'} \supseteq CC \):
  - If \( X \) is an environment of \( (D', CC) \) and
    \( (\exists D'' \in U : P_2(D', D'', CC) \text{ or } P_2(D'', D', CC)) \)
  - then \( (D', CC) \in U \).

Then \( X \) cannot discriminate among any module of \( U \).

**Proof**

Like the previous proof, we have for any two \( (D, CC), (D', CC) \in U \),
\[
\hat{S}_{X \oplus D} \updownarrow (Q_X \setminus QC) = \hat{S}_{X \oplus D'} \updownarrow (Q_X \setminus QC).
\]

This corollary gives rise to a notion of observation equivalence (cf. [21]).

**Definition 4.32 Observation equivalence**

Two modules \( (D_1, CC) \) and \( (D_2, CC) \) are observation equivalent w.r.t. an environment \( X \) if \( X \) cannot distinguish between them, i.e. if \( \hat{S}_{X \oplus D_1} \updownarrow (Q_X \setminus QC) = \hat{S}_{X \oplus D_2} \updownarrow (Q_X \setminus QC) \). If this property holds for all environments \( X \), then they are observation equivalent.

Corollary 4.31 gives a set of modules \( U \), observation equivalent with \( (D, CC) \) w.r.t. environment \( X \). Note that \( U \) may not be the largest set of modules containing \( (D, CC) \) among which \( X \) cannot observe any difference. E.g., if \( P_X = \emptyset \) and \( K_X = CC \), then each module is identical to \( X \). For a maximal environment \( X, U \) might be the set of all modules observation equivalent with \( (D, CC) \). This is an item for further investigation.

5 Application

In this section, we shall apply the theory to an example.
Consider modules $M_1$ and $M_2$ that can communicate with an environment via channels $c$ and $d$. Their task is to transport tokens from $c$ to $d$ and both of them use a private channel $e$.

Module $M_1$:

$M_1 = \langle D_1, \{c, d\} \rangle$ with

\[ C_{D_1} = \{(c, N_0), (d, N_0), (e, N_0)\}, \quad L_{D_1} = \{\emptyset\}, \]
\[ I_{D_1} = \{(p, \{e\}), (q, \{e\})\}, \quad O_{D_1} = \{(p, \{e\}), (q, \{d\})\} \]
and for $n \in N_0$,
\[ R_{D_1}(p)(\{(c, n)\}) = \{(e, n)\} \quad \text{and} \quad R_{D_1}(q)(\{(e, n)\}) = \{(d, n)\}. \]

Module $M_2$:

$M_2 = \langle D_2, \{c, d\} \rangle$ with

\[ C_{D_2} = C_{D_1} \cup \{(c, d) \cup \{(e, N_0 \not\rightarrow N_1)\}\} \quad \text{and} \quad L_{D_2} = \{\{(e, \emptyset)\}\} \]
\[ I_{D_2} = \{(p, \{c, e\}), (q, \{e\})\}, \quad O_{D_2} = \{(p, \{e\}), (q, \{d, e\})\} \]
and for $n \in N_0$ and $f \in N_0 \not\rightarrow N_1$,
\[ R_{D_2}(p)(\{(c, n), (e, f)\}) = \]
\[ \text{if } n \in \text{dom}(f) \]
\[ \text{then } \{(e, f \backslash \{n, f(n)\}) \cup \{n, f(n) + 1\}\} \]
\[ \text{else } \{(e, f \cup \{n, 1\})\} \]
\[ \text{fi} \quad \text{and} \]

This time, $e$ is a database in which values of tokens can be stored.
\[ R_{D_2}(q)(\{(e, f)\}) = \]
\[
\begin{cases} 
  \text{if } \text{dom}(f) = \emptyset & \{(e, f)\} \\
  \text{else if } f(n) = 1 & \{(d, n), (e, f \setminus \{(n, 1)\})\} \\
  \text{else} & \{(d, n), (e, f \setminus \{(n, f(n))\} \cup \{(n, f(n) - 1)\})\}
\end{cases}
\]

\text{fi where } n = \min(\text{dom}(f))

Notice that processor \( q \) selects the smallest value of the contents of store \( e \) and produces a token with that value for channel \( d \).

We claim: No environment can distinguish between \( M_1 \) and \( M_2 \). We shall try to prove this fact.

\textbf{Attempt 1:} Try Lemma 4.30.

First, we must give a correct mapping \( g \in S_{D_1} \rightarrow S_{D_2} \) or a \( g \in S_{D_2} \rightarrow S_{D_1} \). This \( g \) should leave tokens in channels \( c \) and \( d \) unchanged because of the correctness requirement (Definition 4.16). Tokens in channel \( e \) of \( D_1 \) can be mapped in a one-to-one fashion to a database state of store \( e \) of \( D_2 \). Let \( Q_{\{c,d\}} \) be the set of all possible tokens in channels \( c \) and \( d \), i.e. \( Q_{\{c,d\}} = \{(x,v) | x \in \{c,d\} \wedge v \in C_{D_1}(x)\} \). We give \( g \in S_{D_1} \rightarrow S_{D_2} \):

\[
g := \lambda s \in S_{D_1} : s \uplus Q_{\{c,d\}} \uplus \{(e,f)\} \text{ where } f = \lambda n \in \mathbb{N}_0 : s((e,n))
\]

Note that \( g \) is injective. Next, we should prove \( P_2(D_1, D_2, \{c,d\}) \), which in this case equals \( P_3(D_1, D_2, \{c,d\}) \). However, \( P_3(D_1, D_2, \{c,d\}) \) does not hold, because \( D_1 \) and \( D_2 \) are not equivalent! The point is that processor \( q \) of \( D_1 \) can select any token from channel \( e \) to transport to channel \( d \), whereas processor \( q \) of \( D_2 \) always selects the smallest value of database \( e \). Formally, define \( a := \{(e, 1), (e, 3)\} \) and \( a' := \{(e, 1), (d, 3)\} \), then \( a, a' \in S_{D_1} \) and \( (a, a') \in T_{D_1} \) but \( (g(a), g(a')) \notin T_{D_2} \).

\textbf{Attempt 2:} Try Corollary 4.31.

We construct a module \( M_3 = \langle D_3, \{c,d\} \rangle \), a correct function \( g : S_{D_1} \rightarrow S_{D_3} \) and a correct function \( g' : S_{D_2} \rightarrow S_{D_3} \) such that \( P_2(D_1, D_3, \{c,d\}) \) for \( g \) and \( P_2(D_2, D_3, \{c,d\}) \) for \( g' \). Then Corollary 4.31 proves our claim.

\textbf{Module } \( M_3 \):

\[
\begin{array}{ccc}
  \text{c} & \rightarrow & \text{p} \\
  \text{d} & \rightarrow & \end{array}
\]

We illustrate this with a diagram.
5 Application

\[ M_3 = \langle D_3, \{c, d\} \rangle \] with

\[ C_{D_3} = \{(c, N_0), (d, N_0)\}, \quad L_{D_3} = \emptyset, \]

\[ I_{D_3} = \{(p, \{c\})\}, \quad O_{D_3} = \{(p, \{d\})\} \]

and for \( n \in N_0 \), \( R_{D_3}(\{(c, n)\}) = \{(d, n)\} \).

Module \( M_3 \) performs the same task as \( M_1 \) and \( M_2 \), yet without a private channel.

\[
g := \lambda s \in S_{D_1} : s \uplus Q_{\{c,d\}} \uplus [(d, n) \mid (c, n) \in s]
\]

\[
g' := \lambda s \in S_{D_2} : s \uplus Q_{\{c,d\}} \uplus \bigoplus_{(c, a) \in s} \lambda \,(d, a) \in \{d\} \times \text{dom}(f) : f(a)
\]

Remark: \( \text{Dom}(g') \) is not restricted to states with exact one token in channel \( e \).

Proof

It is easy to see that \( g \) and \( g' \) are correct functions. To prove an expression of the form ‘\( A \) simulates \( B \) with \( I \)' the following items must be proven (cf. Definition 2.4):

- \( f(L_A) = L_B \)
- \( f(S_A) = S_B \)
- \( \forall (s, s') \in \hat{T}_A : \langle f(s), f(s') \rangle \in \hat{T}_B \)
- \( \forall (t, t') \in \hat{T}_B, t \neq t' : \forall s_0 \in \hat{f}^{-1}(t) \cap \hat{S}_A : \exists s \in \hat{f}^{-1}(t) : \exists s' \in \hat{f}^{-1}(t') : s_0 \xrightarrow{1_A} s \xrightarrow{1_A} s' . \)

\( P_2(D_1, D_3, \{c, d\}) \):

\( D_1 \) simulates \( D_3 \):

\[
g(L_{D_1}) = g(\emptyset) = \emptyset = L_{D_3} \]
\[
g(S_{D_1}) = g(\emptyset) = \emptyset = S_{D_3} \]

and \( T_{D_1} = T_{D_3} = \{\{(\emptyset), \emptyset\}\} \).

\( \overline{D_1} \) simulates \( \overline{D_3} \):

\[
g(L_{\overline{D}_1}) = g(\{s \in S_{D_1} \mid s \uplus Q_e \text{ is finite}\}) = S_{D_3} = L_{\overline{D}_3}, \]

where \( Q_e = \{(c, n) \mid n \in N_0\} \)

\[
g(S_{\overline{D}_1}) = \# \text{ Lemma 4.9 and the above } \# S_{\overline{D}_3} \]

Let \( \langle s, s' \rangle \in \hat{T}_{\overline{D}_1} \). It suffices to show \( g(s) = g(s') \lor g(s) \lor g(s') \in \hat{T}_{D_3} \).

If \( s = s' \) then we are through, otherwise \( \exists y \in F_{D_1}(s) : \quad s' = s \setminus \text{rng}(y) \cup \bigcup_{i \in \text{dom}(y)} R_{D_1}(i)(y(i)). \)

If \( p \notin \text{dom}(y) \) then \( \text{dom}(y) = \{q\} \) and since \( g(y(q)) = g(R_{D_1}(q)(g(q)), g(s) = g(s'). \)

If \( p \in \text{dom}(y) \) then \( y(p) \in \{(c, n)\} \) for some \( n \in N_0 \) and \( g(R_{D_1}(p)(\{(c, n)\})) = g(\{(c, n)\}) = \{(d, n)\} \). In this case \( g(s') = g(s) \setminus \{(c, n)\} \cup \{(d, n)\} \), hence \( \langle s, g(s'), g(s') \rangle \in T_{D_3} \).
Let \((t, t') \in \tilde{T}_{D_3}, t \neq t'.\) Then \(\exists n \in \mathbb{N}_0 : (c, n) \in t \land t' = t \setminus \{(c, n)\} \cup \{(d, n)\}.

Let \(s_0 \in f^{-1}(t) \cap \tilde{S}_{D_3},\) then \((c, n) \in s_0, (s_0, s_0) \setminus \{(c, n)\} \cup \{(e, n)\}) \in T_{D_1}\) and \(f(s_0) \setminus \{(c, n)\} \cup \{(e, n)\}) = f(s_0) \setminus \{(c, n)\} \cup \{(d, n)\} = t.\)

Last part:
To be proven \(\forall s \in \tilde{S}_{D_1} : \forall b \subseteq g(s) \cap Q_{\{c, d\}} : b\) is finite \(\Rightarrow \exists s' \in \tilde{S}_{D_1} : b \subseteq s' \cap Q_{\{c, d\}} \land s \xrightarrow{g} s'.\)

Let \(s \in \tilde{S}_{D_1}\) and \(b \subseteq g(s) \cap Q_{\{c, d\}}\) such that \(b\) is finite. Define \(b' := s \cap Q_{e}\), then \(b'\) is finite. Let \((e, n_1), \ldots, (e, n_{\#b'})\) be the elements of \(b'\) and define for \(i \in \{0, \ldots, \#b'\},\)

\(s_i := s \setminus \{(e, n_i)\} \cup \{(d, n_i)\}\), then \(s_0 = s, s_{\#b'} = g(s),\)

\(\forall i \in \{1, \ldots, \#b': (s_{i-1}, s_i) \in T_{D_1}\) and \(g(s_i) = g(s)\) for all \(i \in \{0, \ldots, \#b'\}\). Hence \(b \subseteq s_{\#b'}\) and \(s \xrightarrow{g} s_{\#b'}\).

\(P_2(D_2, D_3, \{c, d\})\):

\(D_2\) simulates \(D_3:\)

\[g'(L_{D_2}) = g'\{(\emptyset)\} = \emptyset = L_{D_2}\]

\[g'(\hat{S}_{D_2}) = g'\{(\emptyset)\} = \emptyset = \hat{S}_{D_2}\]

and \(T_{D_2} = T_{D_3} = \{\{(\emptyset), (\emptyset)\}\}.

\(\bar{D}_2\) simulates \(D_3:\)

\[g'(L_{D_2}) = g'\{(s \in S_{D_1} | \exists f \in \mathbb{N}_0 \neq \mathbb{N}_1 : s \cap Q_e = \{(e, f)\} \land \text{dom}(f) \text{ is finite}\}) = S_{D_3} = L_{D_3} \] where \(Q_e = \{(e, f) \mid f \in \mathbb{N}_0 \neq \mathbb{N}_1\}\)

\[g'(\hat{S}_{D_2}) = \# \text{Lemma 4.9 and the above} \# \hat{S}_{D_3}\]

Let \((s, s') \in \tilde{T}_{D_2}.\) We prove \(g'(s) = g'(s') \lor (g'(s), g'(s')) \in T_{D_3}.\) If \(s = s'\) then we are through. For \(s \neq s', \exists y \in \mathbb{F}_{D_2}(s) : s' = s \setminus \text{dom}(y) \cup \bigcup_{i \in \text{dom}(y)} R_{D_2}(y)(i)(y(i)).\)

If \(q \in \text{dom}(y)\) then \(g'(y(q)) = g'(R_{D_2}(q)(y(q))),\) hence if \(p \notin \text{dom}(y)\) then \(g'(s) = g'(s').\)

If \(p \in \text{dom}(y)\) then for some \(n \in \mathbb{N}_0\) and \(f \in \mathbb{N}_0 \neq \mathbb{N}_1, y(p) = \{(c, n), (e, f)\}\) and \(R_{D_2}(p)(y(p)) = \left\{ (e, f \cup \{(n, 1)\}), \quad n \notin \text{dom}(f) \right\} \cup \left\{ (e, f \cup \{(n, f(n) + 1)\}), \quad n \in \text{dom}(f) \right\}.\)

In this case, \(g'(R_{D_2}(p)(y(p))) = g'(y(p)) \setminus \{(c, n)\} \cup \{(d, n)\},\) hence also \(g'(s') = g'(s) \setminus \{(c, n)\} \cup \{(d, n)\},\) i.e. \(g'(s), g'(s') \in T_{D_3}.\)

Last part:
To be proven \(\forall s \in \tilde{S}_{D_2} : \forall b \subseteq g(s) \cap Q_{\{c, d\}} : b\) is finite \(\Rightarrow \exists s' \in \tilde{S}_{D_2} : b \subseteq s' \cap Q_{\{c, d\}} \land s \xrightarrow{g} s'.\)

Let \(s \in \tilde{S}_{D_2}\) and \(b \subseteq g(s) \cap Q_{\{c, d\}}\) such that \(b\) is finite. Now \(s \cap Q_e = \{(e, f)\}\) for some \(f \in \mathbb{N}_0 \neq \mathbb{N}_1 \) with \(\text{dom}(f)\) finite. Define \(n := \sum_{j \in \text{dom}(f)} f(j), s_0 := s, f_0 := f\) and for \(i \in \{1, \ldots, n\}, s_i := s_{i-1} \setminus \{(e, f_{i-1})\} \cup R_{D_2}(g)(\{(e, f_{i-1})\})\) and \(f_i\) is the unique \(f' \in \mathbb{N}_0 \neq \mathbb{N}_1\) satisfying \(s_i \cap Q_e = \{(e, f')\}.\) Then \(\forall i \in \{1, \ldots, n\} : (s_{i-1}, s_i) \in T_{D_2} \land g'(s_{i-1}) = g'(s_i),\) i.e. \(s \xrightarrow{g} s_n.\) \(\text{Dom}(f_n) = \emptyset,\) hence \(s_n = g'(s)\)
and \( b \subseteq s_n \).

\[ \square \]

6 Concluding Remarks

We introduced modules in the DES model, a high-level CPN-like Petri Net model and we gave a semantics for them. Our notion of modules is general, as it does not have structural or behavioural restrictions.

When a module is replaced with another one, the behaviour of the overall system should not change. To this purpose, we gave sufficient and necessary conditions in terms of the modules only, where we used the notions realization, simulation and equivalence [Section 2] to express similar behaviours. For modules, we defined a notion of observation equivalence and for a module with an environment, we gave a set of observation equivalent modules, among which the environment cannot detect any difference. This was illustrated with an example.

Further work

The last-mentioned set of observation equivalent modules is, in general, not as large as possible. E.g. for an empty environment, all modules are equal. But we want to investigate the case of maximal environments further.

In this work we started from a worst-case (maximal) environment to determine predicates \( P_1 \) to \( P_3 \). The environment was allowed to arbitrarily interact with the modules.

In practical cases, usually some kind of communication protocol exists, e.g. the environment might wait for module output before offering new input. Predicates \( P_1 \) to \( P_3 \) may be weakened w.r.t. such protocols. Now several questions arise, such as: What formalism is suited to express that a Petri Net obeys a protocol? We have the idea that process algebras (e.g., ACP [1]) will do. Next, what is the semantics of a module with a protocol? In general, a protocol (which is itself a limited communication behaviour) leads to a smaller set of reachable states, so a different semantics applies. Finally, we intend to illustrate the presented theory with more examples.

References


REFERENCES


A Notations

$\mathbb{N}_0$ is the set of natural numbers and for $i \in \mathbb{N}_0$, $\mathbb{N}_i = \{ j \in \mathbb{N}_0 \mid j \geq i \}$. The symbol '∞' stands for 'infinite'. For any ordered set $S$ and $s \in S : s < \infty$.

For $A$ and $B$ sets, $A \supseteq B$ iff $B \subseteq A$, the set of all total functions from $A$ to $B$ is denoted by $A \rightarrow B$ and $A \not\rightarrow B$ is the set of all partial functions from $A$ to $B$. We denote function restriction by $\upharpoonright$ and for a set of functions $F$, $F \upharpoonright A = \{ f \upharpoonright A \mid f \in F \}$.

$I^P(A)$ denotes the set of all subsets of $A$ and $I^B(A)$ denotes the set of all multisets (bags) over $A$, i.e. the set of all total functions from $A$ to $\mathbb{N}_0 \cup \{ \infty \}$. Please note: Infinitely many copies of the same element can appear in a bag and a bag can contain infinitely many different elements. For $b \in I^B(A)$ and $x$ some element: $x \in b$ iff $x \in A$ and $b(x) > 0$. The number of elements in bag $b$ is denoted by $\#b$, i.e. $\#b = \sum_{a \in A} b(a)$. $b$ is infinite iff $\forall k \in \mathbb{N}_0 : \exists B \subseteq A : \sum_{a \in B} b(a) > k$, otherwise $b$ is finite. For $f$ a function with $\text{dom}(f) = A$ we have an analogous bag notation for $\{ f(x) \mid x \in A \}$: $\{ f(x) \mid x \in b \}$ is defined as $\lambda x \in \{ f(a) \mid a \in A \} : \sum_{a \in A} f(a) = x b(a)$.

For $x \in I^B(A)$ and $y \in I^B(B)$:

- $x \subseteq y$ iff $\forall a \in x : a \in B \land x(a) \leq y(a)$.
- $x = y$ iff $x \subseteq y \land y \subseteq x$.
- $x \uplus y = \lambda a \in A \cup B :$ if $a \in A \setminus B$ then $x(a)$ else if $a \in B \setminus A$ then $y(a)$ else $x(a) + y(a)$.
- $x \setminus y = \lambda a \in A :$ if $a \in B$ then $\max(0, x(a) - y(a))$ else $x(a)$.
- $x \cap y = \lambda a \in A :$ if $a \in B$ then $\max(0, x(a) - y(a))$ else $x(a)$.


The symbol ‘∪’ stands for ‘bag union.’
Please notice that $x \cap y = y \cap x$.

Sets can be viewed as bags. If $S$ is a set, then the corresponding bag $\tilde{S} \in B(S)$ is defined as $\lambda s \in S : 1$. So we can apply the operations above to sets and bags.

Remark: For a non-empty, finite set $S$, $S \cup S \neq S \cup S$.

For a bag-valued function $f$ with finite domain $A = \{a_1, ..., a_n\}$, $\text{mrng}(f)$ denotes the multiset-range of $f$, defined as $\text{mrng}(f) = f(a_1) \cup ... \cup f(a_n)$.

We shall use the following abbreviations. See Appendix B for the concepts ‘token’ and ‘DES’.

For $Q$ a set of tokens; $s, s' \in B(Q)$; $D$ a DES and $n \in \mathbb{N}_0$:

$$s \xrightarrow{D}{n} s' \overset{\text{def}}{=} s \upharpoonright (Q \setminus Q_D) = s' \upharpoonright (Q \setminus Q_D) \quad \text{and} \quad \exists s_0, ..., s_n \in S_D : s_0 = s \upharpoonright Q_D \land s_n = s' \upharpoonright Q_D \land \forall i \in \{1, ..., n\} : \{s_{i-1}, s_i\} \in T_D$$

Furthermore, if $f$ is a function over $S_D$ (i.e. $\text{dom}(f) = S_D$):

$$s \xrightarrow{f} s' \overset{\text{def}}{=} s \upharpoonright (Q \setminus Q_D) = s' \upharpoonright (Q \setminus Q_D) \quad \text{and} \quad \exists m \in \mathbb{N}_0 : \exists s_0, ..., s_m \in f^{-1}(s \upharpoonright Q_D) : s_0 = s \upharpoonright Q_D \land s_m = s' \upharpoonright Q_D \land \forall i \in \{1, ..., m\} : \{s_{i-1}, s_i\} \in T_D$$

Intuitively, $s \xrightarrow{D}{n} s'$ means that $D$ can go from state $s$ to state $s'$ in $n$ steps and $s \xrightarrow{f} s'$ means that it is possible to go from state $s$ to $s'$ within the same class. (Note: States $s$ and $s'$ are in the same class iff $f(s) = f(s')$.)

We abbreviate $s \xrightarrow{D}{n} s' \land s' \xrightarrow{D}{n} s''$ with $s \xrightarrow{D}{m+n} s''$, similarly for $\xrightarrow{f}$.

Without proof we mention several properties:
(For concepts $\text{fit}$ and $\oplus$ see Section 3; $m, n \in \mathbb{N}_0$; $Q$ a set of tokens and $s, s', t, t' \in B(Q)$; $D, D'$ DES’bes and $f$ a function over $S_D$)

- If $s \xrightarrow{D}{n} s'$ then $\exists b \subseteq s : \exists b' \in S_D : b, b'$ are finite $\land s' = (s \setminus b) \cup b'$.
- $(\exists s'' \in B(Q) : s \xrightarrow{D}{m} s'' \xrightarrow{D}{n} s') \iff s \xrightarrow{D}{m+n} s'$.
- If $s \xrightarrow{D}{n} s'$ then $s \cup t \xrightarrow{D}{n} s' \cup t$.
- If $D \text{fit} D'$, $s \xrightarrow{D}{n} s'$ and $t \xrightarrow{D'}{n} t'$ then $s \cup t \xrightarrow{D \oplus D'}{\text{max}(m,n)} s' \cup t'$.
- The set of reachable states $\hat{S}_D$ equals

$$\hat{S}_D = \{ s \in S_D \mid \exists l \in L_D : \exists n \in \mathbb{N}_0 : l \xrightarrow{D}{n} s \}$$

and if $s \in \hat{S}_D$ and $s \xrightarrow{D}{n} s'$ then $s' \in \hat{S}_D$.
- If $s \xrightarrow{f} s'$ then $f(s) = f(s')$. 
• $\rightarrow$ is reflexive and transitive.
• If $s \rightarrow s'$ then $s \cup t \rightarrow s' \cup t$.
• If $s \rightarrow s'$ then $\exists n \in \mathbb{N}_0 : s \rightarrow^n s'$.

B  DES Model

In this appendix we describe the mathematical model for discrete event systems used in this paper. It is called the DES model. For a detailed description we refer to [13, 15].

**Definition B.1 Discrete Event System**

A Discrete Event System (DES) is a quintuple $\langle R, C, I, O, L \rangle$, where $R$ is a function-valued function, $C$ and $O$ are set-valued functions, $I$ is a bag-valued function and $L$ is a set of bags, such that:

- $\text{dom}(I) = \text{dom}(O) = \text{dom}(R)$, countable sets
- $\text{dom}(C)$ is countable
- $\forall p \in \text{dom}(R) : I(p) \in \mathcal{B}(\text{dom}(C)) \land I(p) \neq \emptyset \land I(p)$ is finite
  \quad $\land O(p) \in \mathcal{P}($\text{dom}(C)$)$
- $\forall p \in \text{dom}(R) : R(p) \in \{b \in \mathcal{B}(\{(c, w) \mid c \in \text{dom}(C) \land w \in C(c)\}) : \forall c \in \text{dom}(C) : \sum_{w \in C(c)} b((c, w)) = I(p)(c)\}$
  \quad $\rightarrow \mathcal{B}(\{(c, w) \mid c \in O(p) \land w \in C(c)\})$
- $\forall p \in \text{dom}(R) : \forall b \in \text{dom}(R(p)) : R(p)(b)$ is finite.

$\text{Dom}(R)$ is called the set of *processor indices*, denoted by $P$, $\text{dom}(C)$ is called the set of *channel indices*, denoted by $K$ and for all $p \in P, c \in K$:

$I(p)$ is called the bag of input channels of $p$, where $I(p)(c)$ is the multiplicity of channel $c$;
$O(p)$ is the set of output channels of $p$;
$R(p)$ is the reaction function of $p$;
$C(c)$ is the type of channel $c$.

The set $Q := \{(c, w) \mid c \in K \land w \in C(c)\}$ is the *token set* and the set $S := \mathcal{B}(Q)$ is the *state space*. $L \subseteq S$. $L$ is a collection *initial states*. $\square$
We shall use these symbols strictly for the concepts defined. When we consider different DES'ses we distinguish them by means of subscripts.

To indicate the processors and channels and their input and output relations, we use a diagram technique where processors are represented by squares and channels by circles. For input/output relations we use arrows and lines (lines represent bidirectional relations) and for each input channel we mention its multiplicity, except for input channels with multiplicity one, which is the default value. Stores, i.e. channels always having one token, are denoted with a dot (●). See Figure B.1 for an example.

![Figure B.1: Example.](image)

The remainder of this appendix is concerned with a semantics of DES'ses.

**Definition B.2 Event set, event function**

The event set $E$ of a DES satisfies:

$$
E = \{ e \in P \rightarrow B(Q) \mid \text{dom}(e) \neq \emptyset, \text{dom}(e) \text{ is finite and } \forall p \in \text{dom}(e) : e(p) \in \text{dom}(R(p)) \} .
$$

The event function $F$ of a DES satisfies:

$$
F \in S \rightarrow \mathcal{P}(E) \text{ and } \\
\forall s \in S : F(s) = \{ e \in E \mid \text{mrng}(e) \subseteq s \} .
$$

Please note that an event is an assignment of a bag of tokens to a processor such that for each input channel $e$ with multiplicity $m$ exactly $m$ tokens are chosen. An event $e$ is in $F(s)$ only if $s$ contains enough tokens to supply all the processors of
dom(e). Note that $e \in F(\text{mrng}(e))$, for all events $e \in E$. It is easy to verify that for all $s, t$ in $S$: If $s \subseteq t$ then $F(s) \subseteq F(t)$.

The transition function assigns a new state to a state $s$ and an event $e \in F(s)$.

**Definition B.3** Transition function, transition relation

The transition function $T$ of a DES satisfies:

$$T \in S \times E \nexists S$$ such that $\text{dom}(T) = \{(s, e) \mid s \in S \land e \in F(s)\}$

and for $s \in S, e \in F(s)$:

$$T(s, e) = s \setminus \text{mrng}(e) \cup \bigcup_{p \in \text{dom}(e)} R(p)(e(p)).$$

The transition relation $T$ of a DES is:

$$\{(s, t) \in S \times S \mid \exists e \in F(s) : T(s, e) = t\}.$$  

Remark: The symbol $T$ has been overloaded.

Elements of the transition relation are called transitions.

Next definition gives a non-interleaving semantics of DES'ses.

**Definition B.4** Semantics of a DES

The semantics of a DES is the unlabeled transition system $(S, L, T)$, where $S$, $L$ and $T$ are the state space, initial states and transition relation of the DES, respectively.

For more results on the DES model, see [13, 15].
In this series appeared:

89/1 E.Zs.Lepoeter-Molnar
Reconstruction of a 3-D surface from its normal vectors.

89/2 R.H. Mak
P.Struik
A systolic design for dynamic programming.

89/3 H.M.M. Ten Eikelder
C. Hemerik
Some category theoretical properties related to a model for a polymorphic lambda-calculus.

89/4 J.Zwiers
W.P. de Roever
Compositionality and modularity in process specification and design: A trace-state based approach.

89/5 Wei Chen
T.Verhoeff
J.T.Udding
Networks of Communicating Processes and their (De-)Composition.

89/6 T.Verhoeff
Characterizations of Delay-Insensitive Communication Protocols.

89/7 P.Struik
A systematic design of a parallel program for Dirichlet convolution.

89/8 E.H.L.Aarts
A.E.Eiben
K.M. van Hee
A general theory of genetic algorithms.

89/9 K.M. van Hee
P.M.P. Rambags
Discrete event systems: Dynamic versus static topology.

89/10 S.Ramesh
A new efficient implementation of CSP with output guards.

89/11 S.Ramesh
Algebraic specification and implementation of infinite processes.

89/12 A.T.M.Aerts
K.M. van Hee
A concise formal framework for data modeling.

89/13 A.T.M.Aerts
K.M. van Hee
M.W.H. Hesen
A program generator for simulated annealing problems.

89/14 H.C.Haesen
ELDA, data manipulatie taal.

89/15 J.S.C.P. van der Woude
Optimal segmentations.

89/16 A.T.M.Aerts
K.M. van Hee
Towards a framework for comparing data models.

89/17 M.J. van Diepen
K.M. van Hee
A formal semantics for Z and the link between Z and the relational algebra.
Formal methods and tools for the development of distributed and real time systems, p. 17.

Dynamic process creation in high-level Petri nets, pp. 19.

Foundations of Compositional Program Refinement - safety properties -, p. 38.

Decomposition of delay-insensitive circuits, p. 25.

On the delay-sensitivity of gate networks, p. 23.


A logic for one-pass, one-attributed grammars, p. 14.

Receptive Process Theory, p. 16.

Combining the functional and the relational model, p. 15.

A formal semantics for Z and the link between Z and the relational algebra, p. 30. (Revised version of CSNotes 89/17).

A proof system for process creation, p. 84.

A proof theory for a sequential version of POOL, p. 110.

Proving termination of Parallel Programs, p. 7.

A proof system for the language POOL, p. 70.

Compositionality in the temporal logic of concurrent systems, p. 17.

A fully abstract model for concurrent logic languages, p. 23.

On the asynchronous nature of communication in logic languages: a fully abstract model based on sequences, p. 29.
Design and implementation aspects of remote procedure calls, p. 15.

Two Case Studies in ExSpect, p. 24.

The Nature of Delay-Insensitive Computing, p. 18.

Data, Process and Behaviour Modelling in an integrated specification framework, p. 37.


Implication. A survey of the different logical analyses "if...,then...", p. 26.

Parallel Programs for the Recognition of P-invariant Segments, p. 16.

Performance Analysis of VLSI Programs, p. 31.

An Implementation Model for GOOD, p. 18.

SPECIFICATIEMETHODEN, een overzicht, p. 20.

CPO-models for second order lambda calculus with recursive types and subtyping, p. 49.

Terminology and Paradigms for Fault Tolerance, p. 25.

Interval Timed Petri Nets and their analysis, p. 53.

POLYNOMIAL RELATORS, p. 52.

Relational Catamorphism, p. 31.


A note on Extensionality, p. 21.

The PDB Hypermedia Package. Why and how it was built, p. 63.

An example of proving attribute grammars correct: the representation of arithmetical expressions by DAGs, p. 25.

Transforming Functional Database Schemes to Relational Representations, p. 21.

Transformational Query Solving, p. 35.

Some categorical properties for a model for second order lambda calculus with subtyping, p. 21.


Assertional Data Reification Proofs: Survey and Perspective, p. 18.


Z and high level Petri nets, p. 16.

Formal semantics for BRM with examples, p. 25.

A compositional proof system for real-time systems based on explicit clock temporal logic: soundness and completeness, p. 52.

The GOOD based hypertext reference model, p. 12.

Embedding as a tool for language comparison: On the CSP hierarchy, p. 17.

A compositional proof system for dynamic process creation, p. 24.

Correctness of Acceptor Schemes for Regular Languages, p. 31.

An Algebra for Process Creation, p. 29.
<table>
<thead>
<tr>
<th>Year/Issue</th>
<th>Authors</th>
<th>Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>91/31</td>
<td>H. ten Eikelder</td>
<td>Some algorithms to decide the equivalence of recursive types, p. 26</td>
</tr>
<tr>
<td>91/32</td>
<td>P. Struik</td>
<td>Techniques for designing efficient parallel programs, p. 14</td>
</tr>
<tr>
<td>91/33</td>
<td>W. v.d. Aalst</td>
<td>The modelling and analysis of queueing systems with QNM-ExSpect, p. 23</td>
</tr>
<tr>
<td>91/34</td>
<td>J. Coenen</td>
<td>Specifying fault tolerant programs in deontic logic, p. 15</td>
</tr>
<tr>
<td>92/01</td>
<td>J. Coenen, J. Zwiers, W.-P. de Roever</td>
<td>A note on compositional refinement, p. 27.</td>
</tr>
<tr>
<td>92/02</td>
<td>J. Coenen, J. Hooman</td>
<td>A compositional semantics for fault tolerant real-time systems, p. 18</td>
</tr>
<tr>
<td>92/03</td>
<td>J.C.M. Baeten, J.A. Bergstra</td>
<td>Real space process algebra, p. 42.</td>
</tr>
<tr>
<td>92/05</td>
<td>J.P.H.W.v.d.Eijnde</td>
<td>Conservative fixpoint functions on a graph, p. 25.</td>
</tr>
<tr>
<td>92/06</td>
<td>J.C.M. Baeten, J.A. Bergstra</td>
<td>Discrete time process algebra, p.45.</td>
</tr>
<tr>
<td>92/07</td>
<td>R.P. Nederpelt</td>
<td>The fine-structure of lambda calculus, p. 110.</td>
</tr>
<tr>
<td>92/10</td>
<td>P.M.P. Rambags</td>
<td>Composition and decomposition in a CPN model, p. 55.</td>
</tr>
</tbody>
</table>