Discrete event systems : dynamic versus static topology

Discrete Event Systems: Dynamic Versus Static Topology

by

K.M. van Hee P.M.P. Rambags

89/09

December, 1989

COMPUTING SCIENCE NOTES

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 or the editor.

Eindhoven University of Technology
Department of Mathematics and Computing Science
P.O. Box 513
5600 MB EINDHOVEN
The Netherlands

editors: prof.dr.M.Rem
prof.dr.K.M.van Hee.

DISCRETE EVENT SYSTEMS: DYNAMIC VERSUS STATIC TOPOLOGY

K.M. van Hee **P.M.P. Rambags**

Department of Mathematics and Computing Science
Eindhoven University of Technology

Abstract

In this paper, we present two models for discrete event systems: A formal Actor model, based upon Agha [1], and the Des model [5]. The former has a dynamic topology whereas the latter has a fixed interaction structure.
We introduce an equivalence relation for discrete event systems and we construct an equivalent Des for each Actor system. Furthermore, we prove several results, such as the serializability of events in the Actor model and the establishment that every name-generating mechanism is isomorphic to the method introduced by Agha, i.e., to extend an existing name with some element.

1 Introduction

Many existing models for discrete event systems have a fixed topology, i.e. the number of active components and their interaction structure are fixed. The Petri Net model [13] is a well-known example.
There exists also another class of discrete event systems where the topology is dynamic, i.e. new components can be created during the course of the system and the interaction structure may change. The Actor system is one of the most well-known specimen of this class.
In order to derive properties of Actor systems and to manipulate them, we need a formal description. Agha [1] uses recursive equations. It is not obvious to see whether these equations have solutions and if so, which ones. An other approach is based upon

graph grammars [9]. We present in Section 3 a formal Actor model constructed from sets and functions. We have been inspired by [1], which allows us to abandon all time aspects, only order of events is left. We assume only one actor machine of the same actor to be alive at each moment. We adopt the method to generate new names locally. All new names will be an extension of an existing name with a natural number and we prove that every name-generating mechanism is isomorphic to this method.
In Section 4, we briefly introduce another model for discrete event systems, called the *Des model.* It has a fixed interaction structure. We shall not explain the basic ideas behind it, instead we refer to [6].

First, we introduce in Section 2 a very general notion of discrete event systems, called *transition system.* Its purpose is twofold: We use it to describe the *semantics* of more specific models, in this case the Actor and Des models, and to *compare* discrete event systems. For the latter, several similarity relationships on transition systems are defined, of which the strongest is *equivalence.* Thus we are able to compare discrete systems described in completely different frameworks, in this case Actor systems and Des'ses. In Section 5, we first construct a very simple Des which realizes any Actor system only. It has one processor. Consequently, it cannot perform actions in parallel. Next, we construct an equivalent Des for each Actor system. This result may seem remarkable because of the static topology of a Des. The resulting Des, however, has some difficulties, e.g. processors have infinitely many output channels. We solve these problems by means of a two-steps construction. In the end, each processor with infinitely many output channels has been replaced by one or more simple processors with only finitely many output channels. The resulting Des can be implemented in EXSPECT [6,7]. Consequently, we are able to prototype any Actor system.
We start with some notations.

Notations

$\mathbb{N}_0$ is the set of natural numbers including zero and for $i \in \mathbb{N}_0$: $\mathbb{N}_i = \{j \in \mathbb{N}_0 | j \geq i\}$.
For S a set and T a binary relation over S, T^* is the transitive closure of T. We overload the symbol *: For S a set, S^* is the set of all finite rows over S. We denote the empty row by ε and row concatenation by $\circ$. We write $\langle s_1, s_2, \ldots, s_n \rangle$ for a row consisting of the elements $s_1, s_2, \ldots, s_n$, respectively. For $x \in S^*$, $|x|$ is its length and x_i is the i^{th} element of x. Sometimes we use $x \,\&\, a$ as shorthand for $x \circ \langle a \rangle$.
For A and B sets, $A \supseteq B$ iff $B \subseteq A$, $A \rightarrow B$ denotes the set of all total functions from A to B and $A \not\rightarrow B$ the set of all partial functions from A to B. For f a function, $f[a : b] = \lambda\, x \in \mathrm{dom}(f)$: $\underline{\text{if}}\ x = a\ \underline{\text{then}}\ b\ \underline{\text{else}}\ f(x)\ \underline{\text{fi}}$, and for $X \subseteq \mathrm{dom}(f), f(X) = \{f(x) \mid x \in X\}$. We denote function concatenation by $\circ$ and function restriction by $\restriction$. For an injective function, we write f^{-1} for its inverse. For an non-injective function and y some element, $f^{-1}(y) = \{x \in \mathrm{dom}(f) \mid f(x) = y\}$.
For A and $B_1, B_2, \ldots, B_n$ sets, $f \in A \rightarrow B_1 \times B_2 \times \ldots \times B_n$, $a \in A$ and $i \in \{1, \ldots, n\}$: $f_i(a)$ denotes the i^{th} component of $f(a)$.
The symbol 'ω' stands for 'infinite'. For all $n \in \mathbb{N}_0$: $n < \omega$. If X is a set of numbers, then $X^\omega = X \cup \{\omega\}$. For an ordered set D and $d_1, d_2 \in D$: $d_1 \,\underline{\min}\, d_2$ is the minimum of d_1 and d_2 and $d_1 \,\underline{\max}\, d_2$ is the maximum of d_1 and d_2.
If Y is a set of sets, then $\bigcup Y$ denotes the union of all elements of Y. If A is a set,

then $|A|$ is the number of elements in A, $\mathbb{P}(A)$ denotes the set of all subsets of A and $\mathbb{B}(A)$ denotes the set of all multisets (bags) over A, i.e. the set of all functions from A to $\mathbb{N}_0^\omega$. Please note: Infinitely many copies of the same element can appear in a bag and a bag can contain infinite many different elements.
For $b \in \mathbb{B}(A)$ and x some element: $x \in b$ iff $x \in A$ and $b(x) > 0$.
For A, B sets, $x \in \mathbb{B}(A)$ and $y \in \mathbb{B}(B)$:

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

Please notice that $x \cap y = y \cap x$.

Sets can be viewed as bags. If S is a set, then the corresponding bag $\tilde{S} \in \mathbb{B}(S)$ is defined as $\lambda \, s \in S \ : \ 1$. So we can apply the operations above to sets and bags. For a bag-valued function f with finite domain $A = \{a_1, \ \ldots \ , a_n\}$, $\mathrm{mrng}(f)$ denotes the multisetrange of f, defined as $\mathrm{mrng}(f) = f(a_1) \cup \ \ldots \ \cup f(a_n)$. We also write $\bigcup_{a \in A} f(a)$ for $\mathrm{mrng}(f)$.
If x is a bag over A, then $\# x$ is the number of elements in x, i.e. $\# x = \sum_{a \in A} x(a)$.
x is *infinite* iff $\forall k \in \mathbb{N}_0 \ : \ \exists B \subseteq A \ : \ \sum_{a \in B} x(a) > k$, otherwise x is *finite*.

For clarity, we sometimes use the following notation in proofs:

A

$\Leftrightarrow$ $\quad \{\!\!\{$ Hint why A is equivalent with B $\}\!\!\}$

B .

2 Transition system

In this section we formalize a general notion of a discrete system which is called a *transition system*. We shall use transition systems to describe the semantics of the Actor and Des models. We also mention several relationships between transition systems in order to mutually compare them.

A transition system consists of a finite or countable set of states of which some are initial. The system starts with an initial state and then moves from one state to another. Actually, a transition system is a directed graph. Most frameworks incorporate, in some form, a transition system.

Definition 2.1 *Transition system*
A transition system is a triple $\langle S, L, T \rangle$, where:

- S is a finite or 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*.
□

This definition of a transition system can be found in [3]. In literature, also other classes of transition systems are described, for instance see [8,11,12,17]. They differ from ours in mainly two items, 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 $\langle s, a, s' \rangle$ 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.

All these classes of transition systems can be transformed into each other, but we shall not elaborate it here.

A transition system may have several states which are never reachable, i.e. the system can never get there when started in an initial state. In some sense these states are superfluous, yet we do not require all states be reachable. When specifying a transition system one may not know which states are reachable and which are not. It may also be convenient to define the state space too large.

Relationships between transition systems

One of our primary goals is to compare transition systems. We would like to characterize the behaviour of these 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 compare transition systems have been described in literature, e.g. *observation equivalence* [12] and *bisimulation equivalence* [8], but they consider other classes of transition systems and application areas. We shall introduce our own similarity relationships. It would be an interesting topic for further research to relate all classes of transition systems and their comparison techniques.

When developing a new system, we compare several designs with each other and with the existing system. In these cases we have a specification of some transition system and one or more implementations. All implementations can be roughly divided up into three classes: Incorrect implementations, correct but incomplete implementations and both correct and complete implementations. An incorrect implementation can do more than is specified. A correct implementation performs allowed actions only, but it might be incomplete, i.e. the specification allows a larger set of actions than the implementation actually can do. See Figure 2.1.

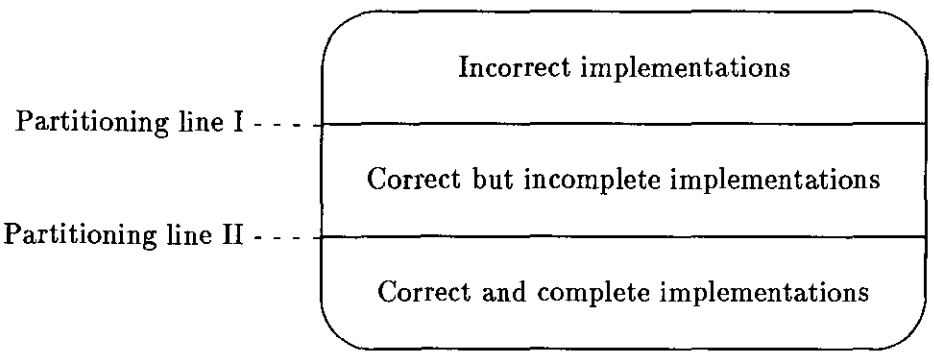


Figure 2.1 *Three classes of implementations*

In this subsection, we introduce some concepts to make the comparisons between discrete systems more precise. We only consider transition systems here. Consequently, if we like to compare two systems described in different frameworks, we have to find representations of them on the level of transition systems first.
We distinguish different transition systems by means of uppercase letters of the beginning alphabet. Their state spaces, initial states and transition relations are distinguished by subscripts.
First, we apply a reduction on any transition system such that all states are reachable and the transition relation is reflexive.

Definition 2.2 *Reduced transition system*
Let $\langle S, L, T \rangle$ be a transition system. Its reduction $\langle \dot{S}, \dot{L}, \dot{T} \rangle$ satisfies:

$$\dot{S} = \{ s \in S \mid \exists n \in I\!N_0 : \exists s_0, \ldots, s_n \in S : s_0 \in L \wedge s_n = s \wedge \forall i \in \{1, \ldots, n\} : \langle s_{i-1}, s_i \rangle \in T \}$$

$$\dot{L} = L$$

$$\dot{T} = \{ \langle s, s' \rangle \in \dot{S} \times \dot{S} \mid s = s' \vee \langle s, s' \rangle \in T \}$$

□

This reduction gives just the information we need to determine the behaviour of a transition system. All similarity relationships will be based hereupon.
We map reachable states of a system A onto reachable states of a system B by a total function $f \in \dot{S}_A \rightarrow \dot{S}_B$. 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 are a measure for the efficiency of system A as compared with system B.
Now we define the first relationship between transition systems: *Realization.*

Definition 2.3 *Realization*
Let A and B be transition systems. A realizes B with respect to function f iff

- $f \in \dot{S}_A \rightarrow \dot{S}_B$
- $f(\dot{L}_A) \subseteq \dot{L}_B$
- $\forall \langle s, s' \rangle \in \dot{T}_A : \langle f(s), f(s') \rangle \in \dot{T}_B$

□

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. In terms of Figure 2.1, the realization relation formalizes partitioning line I.
Without proof we mention that the realization relation is transitive:

Lemma 2.1
Let A realize B w.r.t. function f and let B realize C w.r.t. function g.
Then A realizes C w.r.t. $g \circ f$.
□

The definition of the realization relation as such is not useful in many practical cases. The point is that it has been based on reachable states and we do not know in advance which states are reachable and which are not. However, it can often be proven that all reachable states have certain properties in common. One might speak of *system invariants.* In the following lemma, S can be regarded as a set of states for which these invariants hold. It can be used in several practical cases.

Lemma 2.2
Let S be a set and f a function with the following properties:

$$- L_A \subseteq S \subseteq S_A \tag{1}$$

$$- \forall s \in S \ : \ \forall s' \in S_A \ : \ \langle s, s' \rangle \in T_A \ \Rightarrow \ s' \in S \tag{2}$$

$$- f \in S \rightarrow S_B \tag{3}$$

$$- f(L_A) \subseteq L_B \tag{4}$$

$$- \forall s, s' \in S \ : \ \langle s, s' \rangle \in T_A \ \Rightarrow \ f(s) = f(s') \ \vee \ \langle f(s), f(s') \rangle \in T_B \tag{5}$$

Then A realizes B with f.

Proof: See appendix.
□

If all transitions of system A correspond to a transition of B, then A realizes B. This is a special case of Lemma 2.2.

Lemma 2.3
Let $f \in S_A \rightarrow S_B$, $f(L_A) \subseteq L_B$ and suppose

$$\forall \langle s, s' \rangle \in T_A \ : \ \langle f(s), f(s') \rangle \in T_B \ .$$

Then A realizes B with function f.
□

Consequently,

Lemma 2.4
If $S_A = S_B$, $L_A \subseteq L_B$ and $T_A \subseteq T_B$, then A realizes B with the identical function.
□

Notice that A may realize B even if the transition relation of A is empty. If A realizes B then we consider B at least as powerful as A.
We are now introducing a stronger relationship: *Simulation*, in which we also require that behaviour of B is, under the image of f, 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
- $f(L_A) = L_B$
- $\forall \langle t, t' \rangle \in \dot{T}_B \ : \ \forall s_0 \in f^{-1}(t) \ : \ \exists n \in I\!N_0 \ : \ \exists s_1, \ldots, s_n \in f^{-1}(t) \ : \ \exists s_{n+1} \in f^{-1}(t') \ : \ \forall i \in \{0, \ldots, n\} \ : \ \langle s_i, s_{i+1} \rangle \in \dot{T}_A$

□

Without proof we mention that the simulation relation is transitive, too:

Lemma 2.5
Let A simulate B w.r.t. function f and let B simulate C w.r.t. function g.
Then A simulates C w.r.t. $g \circ f$.
□

Lemma 2.2 can be extended for the simulation relation.

Lemma 2.6
Let S be a set and f a function with additional properties:

- $L_A \subseteq S \subseteq S_A$ (1)
- $\forall s \in S \ : \ \forall s' \in S_A \ : \ \langle s, s' \rangle \in T_A \ \Rightarrow \ s' \in S$ (2)
- $f \in S_A \rightarrow S_B$ (3)
- $f(L_A) = L_B$ (4)
- $\forall s, s' \in S \ : \ \langle s, s' \rangle \in T_A \ \Rightarrow \ f(s) = f(s') \ \vee \ \langle f(s), f(s') \rangle \in T_B$ (5)
- $\forall \langle t, t' \rangle \in T_B \ : \ \forall s_0 \in f^{-1}(t) \cap S \ : \ \exists s_1, \ldots, s_n \in f^{-1}(t) \ : \ \exists s_{n+1} \in f^{-1}(t') \ : \ \forall i \in \{0, \ldots, n\} \ : \ \langle s_i, s_{i+1} \rangle \in T_A$ (6)

Then A simulates B with f.

Proof: See appendix.
□

Simulation with an injective function implies that both systems simulate each other. We omit the proof.

Lemma 2.7
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 transition systems. Finally, we have come to the strongest relationship: *Equivalence.*

Definition 2.5 *Equivalence*
Two transition systems A and B are equivalent iff a bijective function $f : \dot{S}_A \to \dot{S}_B$ exists with the following properties:

- $f(\dot{L}_A) = \dot{L}_B$
- $\forall s, s' \in \dot{S}_A : \langle s, s' \rangle \in \dot{T}_A \Leftrightarrow \langle f(s), f(s') \rangle \in \dot{T}_B$.

Notation: $A \cong B$.
□

It is straightforward to see that equivalence equals simulation with an injective function.

Lemma 2.8
$A \cong B$ iff an injective function f exists such that A simulates B with f.
□

We shall use the following lemma in a subsequent section to prove the equivalence of two systems.

Lemma 2.9
Let $f \in S_A \to S_B$ be an injective, not necessary surjective function. Assume furthermore:

- $\forall \langle t, t' \rangle \in T_B : t \in \mathrm{rng}(f) \Rightarrow t' \in \mathrm{rng}(f)$ (1)
- $f(L_A) = L_B$ (2)
- $\forall s, s' \in S_A : \langle s, s' \rangle \in T_A \Leftrightarrow \langle f(s), f(s') \rangle \in T_B$ (3)

Then $A \cong B$.

Proof: See appendix.
□

In the next section, we present a formal model for Actor systems.

3 Actor model

In this section, we present a formal model for Actor systems. It has been based upon [1], but we assume only one actor machine of each actor to be present at each moment.

Agha [1] has already introduced a formal model, he however uses recursive domain equations. This method has some disadvantages, for instance it is not immediately clear whether such an equation has solutions, and if so, which ones. Often a complex fixpoint theory is needed to find a minimal solution (e.g. see [2,14,15,16]). Therefore we choose a different approach.
In order to give a non-recursive definition for actors, we introduce so-called *local states*, i.e. for every actor some internal memory. We have thus introduced over-specification, a designer might not want to bother about local actor states as we force him to do. We do not regard this as a disadvantage, on the contrary we feel that our model is easier to understand and use. It has been illustrated with some examples.

Agha introduced a method to generate different names for new communications and new actors. In this method, each new name is an extension of an existing communication name with some element. All initial configurations should be such that no name of an actor or communication is a prefix of another name. He proved that this property remains invariant when the Actor system evolves. Consequently, no new name can equal an already existing name.
This method has a nice feature: All new names can be computed locally. To justify its usage, we show that all name-giving mechanisms are isomorphic to it, before we formalize Actor systems.

Index problem

Let I be a countable set of indices (names) and $f \in I \rightarrow \mathbb{P}(I)$ a function that assigns to each index a set of new indices, such that:

$$\forall i,j \in I \ : \ i \neq j \ \Rightarrow \ f(i) \cap f(j) = \emptyset \tag{1}$$

$$\forall i \in I \ : \ \exists i_0 \in I \setminus \cup \, \mathrm{rng}(f) \ : \ \exists n \in \mathbb{N}_0 \ : \ i \in f^n(i_0) \tag{2}$$

According to Eq. 1, all images under f are disjunct. Eq. 2 implies that each index can be reached from an initial index in finitely many steps. This is the most general form of a name-giving mechanism.
We construct an isomorphic set of indices X, $X \subseteq \mathbb{N}_0{}^*$, and a function $g \in X \rightarrow \mathbb{P}(X)$. Here, each index will be a finite string of natural numbers and every new index will be an extension of the original index with one number. Let:

- $I = \{i_0, i_1, \ldots\}$ be an enumeration of I with order, i.e. $i_0 < i_1 < \ldots$
- $I_0 = I \setminus \cup \, \mathrm{rng}(f)$ (all initial indices)
- $f' \in (I \setminus I_0) \rightarrow I$ such that $f'(j) = i$ iff $j \in f(i)$, $\forall i,j$.
  Please note: f' exists and has been defined unambiguously.

- $r \in I \to \mathbb{N}_0$ with $r = \lambda\, i \in I\ :\ \underline{\text{if}}\ i \in I_0\ \underline{\text{then}}\ \#\{\, j \in I_0 \mid j < i \,\}$
  $\underline{\text{else}}\ \#\{\, j \in f(f'(i)) \mid j < i \,\}\ \underline{\text{fi}}$

  Function r assigns to each $i \in I$ a local number.

Now we construct a function h which maps every $i \in I$ on a string of natural numbers:

- $h\ :=\ \lambda\, i \in I\ :\ \underline{\text{if}}\ i \in I_0\ \underline{\text{then}}\ \langle r(i) \rangle\ \underline{\text{else}}\ h(f'(i)) \,\&\, r(i)\ \underline{\text{fi}}$
- $X\ :=\ \text{rng}(h)$

Hence, $h \in I \to X$. Function h has been defined recursively and due to Eq. 2, the recursion is finite. Consequently, $X \subseteq \mathbb{N}_0{}^*$.

Lemma 3.1
h is injective.

Proof.
Let $i, j \in I$ and suppose $h(i) = h(j)$. We prove $i = j$. Since $|h(i)| = |h(j)|$ we have $i, j \in I_0$ or $i, j \in I \setminus I_0$. If $i, j \in I_0$ then $r(i) = r(j)$ and from the definition of r it follows that $i = j$. Next, assume $i, j \in I \setminus I_0$. Then $h(f'(i)) = h(f'(j))$ and $r(i) = r(j)$. Consequently, $|h(f'(i))| = |h(f'(j))| = |h(i)| - 1$ and with induction to the length of $h(i)$ it follows that $f'(i) = f'(j)$. Hence, $i, j \in f(f'(i))$ and they have the same local number, so $i = j$.
□

This means that the inverse of function h, h^{-1}, exists. We now define function g:

- $g\ :=\ \lambda\, x \in X\ :\ \{\, x \,\&\, n \mid n \in \mathbb{N}_0 \wedge n < \#f(h^{-1}(x)) \,\}$

Theorem 3.1 *Index problem*
$\langle I, f \rangle$ is isomorphic to $\langle X, g \rangle$ with respect to h.

Proof.
Function h is a bijection from I to X. What remains to prove is that for all $i \in I$: $h(f(i)) = g(h(i))$.

$$
\begin{aligned}
h(f(i)) \ =\ & \{\, h(j) \mid j \in f(i) \,\} \ =\ \{\, h(f'(j)) \,\&\, r(j) \mid j \in f(i) \,\} \ = \\
& \{\, h(i) \,\&\, n \mid n < \#f(i) \,\} \ =\ \{\, h(i) \,\&\, n \mid n < \#f(h^{-1}(h(i))) \,\} \ = \\
& g(h(i))\ .
\end{aligned}
$$

□

Now we formalize the Actor system.

Formal model

The readers unfamiliar with the Actor model should read the informal description in [1].
We construct an Actor system out of three sets and a complex function over these sets. The sets specify, respectively,

- tags, i.e. communication names and actor names,
- values of messages,
- states of actors.

We assign to each actor a state. This state can be seen as some local information. For example, an actor that forwards the previous message when the current message arrives, should be able to remember one message. If all messages are natural numbers, then its state is a natural number.
We explicitly mention that the set of message values and the set of actor states do not deal with acquaintances. A communication will consist of a message value and a possibly empty set of acquaintance names. An actor always has a state and a set of acquaintances.

Definition 3.1 *Actor system*
An Actor system is a quadruple $\langle I, M, A, B \rangle$, where:

- I is a finite, ordered set
  I^* is the set of all possible tags, which has been ordered lexicographically. Since a tag belongs to a communication or an actor, we define $\mathcal{A} := I^*$ and $\mathcal{T} := I^*$ and we use $\mathcal{A}$ for actor names and $\mathcal{T}$ for communication names.
- M is a countable set
  M is the set of message values.
  $\widetilde{M} := M \times I\!P(\mathcal{A})$, the set of all nameless communications.
- A is a countable set
  A is the set of actor states.
  $\widetilde{A} := A \times I\!P(\mathcal{A})$, actor states with acquaintances.
- $B \in \widetilde{A} \times \mathcal{T} \times \widetilde{M} \rightarrow (\mathcal{T} \not\rightarrow \mathcal{A} \times \widetilde{M}) \times (\mathcal{A} \not\rightarrow \widetilde{A}) \times \widetilde{A}$
  B is the behaviour function. It assigns to an actor state ($\widetilde{A}$), a communication name ($\mathcal{T}$) and a communication ($\widetilde{M}$): A set of named communications to actors ($\mathcal{T} \not\rightarrow \mathcal{A} \times \widetilde{M}$), some new actors ($\mathcal{A} \not\rightarrow \widetilde{A}$) and a replacement behaviour ($\widetilde{A}$).

For B, the following constraints should hold: (All free variables universally quantified)

If $B(\langle a, V\rangle, t, \langle m, W\rangle) = \langle f, g, \langle aa, VV\rangle\rangle$, then:

- $\mathrm{dom}(f) \cap \mathrm{dom}(g) = \emptyset$
  New actors and new communications should be named differently.

- $\mathrm{dom}(f)$ and $\mathrm{dom}(g)$ are finite
  An actor should not produce an infinite amount of new communications or actors.

- $t' \in \mathrm{dom}(f) \cup \mathrm{dom}(g) \Rightarrow \exists i \in I \ : \ t' = t \,\&\, i$
  New names are an extension of the received communication name.

- $\langle \alpha', \langle m', W'\rangle\rangle \in \mathrm{rng}(f) \Rightarrow \alpha' \in V \cup W \cup \mathrm{dom}(g)$
  $\wedge \ W' \subseteq V \cup W \cup \mathrm{dom}(g)$
  Messages can be sent to well-known actors only and such a message may contain no unknown actor names. Please note:
  V = Acquaintances of the actor itself;
  W = Acquaintances of the received communication;
  $\mathrm{dom}(g)$ = Newly created actor names.

- $\langle a', V'\rangle \in \mathrm{rng}(g) \Rightarrow V' \subseteq V \cup W \cup \mathrm{dom}(g)$
  The acquaintances of newly created actors should be well-known.

- $VV \subseteq V \cup W \cup \mathrm{dom}(g)$
  All new acquaintances of the actor itself should be well-known, too.

□

We use partial functions over a set of names to denote sets of named elements. For example, let $f \in \mathcal{T} \not\rightarrow \widetilde{M}$, then f corresponds with a set of named communications. A configuration of an Actor system consists of a finite set of actor names and for each actor name:

- An actor state with acquaintances ($\widetilde{A}$);

- A set of named communications ($\mathcal{T} \not\rightarrow \widetilde{M}$).

A configuration can thus be described as an element of $\mathcal{A} \not\rightarrow \widetilde{A} \times (\mathcal{T} \not\rightarrow \widetilde{M})$. We call the set of all possible configurations of an Actor system the *state space.*

Definition 3.2 *State space*
Let an Actor system be given. Its state space S satisfies:

$$
\begin{aligned}
S \;=\; \{\, & s \in \mathcal{A} \nrightarrow \widetilde{A} \times (\mathcal{T} \nrightarrow \widetilde{M}) \\
& \mid \forall \langle\langle a, V\rangle, f\rangle \in \mathrm{rng}(s) \;:\; V \subseteq \mathrm{dom}(s) \\
& \qquad\qquad \wedge\; \forall \langle m, W\rangle \in \mathrm{rng}(f) \;:\; W \subseteq \mathrm{dom}(s) \\
& \quad \wedge\; \forall N, N' \in \mathrm{Names}(s) \;:\; N \neq N' \Rightarrow N \cap N' = \emptyset \\
& \quad \wedge\; \forall n, n' \in \bigcup \mathrm{Names}(s) \;:\; ((\exists u \in I^* \;:\; n \circ u = n') \Rightarrow n = n') \\
& \quad \wedge\; \textstyle\bigcup \mathrm{Names}(s) \text{ is finite} \\
& \}
\end{aligned}
$$

$$
\text{where } \mathrm{Names}(s) = \{\mathrm{dom}(s)\} \cup \bigcup_{\alpha \in \mathrm{dom}(s)} \{\mathrm{dom}(s_2(\alpha))\}
$$

□

Names(s) is the union of all sets of names in configuration s.
According to the first constraint on S, all acquaintances should be part of the configuration. The second constraint asserts that all names are different and because of the third constraint, no name is a prefix of another one. Finally, the last restriction states that the number of communications and actors is always finite.
In a given configuration, each actor that has a communication may consume it, produce new communications and actors and replace its behaviour. We call an assignment of communications to actors an *event*.

Definition 3.3 *Event function*
The event function E of an Actor system with state space S satisfies:

$$
\begin{aligned}
E \;=\; \lambda\, s \in S \;:\; \{\, e \in \mathcal{A} \nrightarrow \mathcal{T} \;\mid\; & \mathrm{dom}(e) \neq \emptyset \;\wedge\; \mathrm{dom}(e) \subseteq \mathrm{dom}(s) \\
& \wedge\; \forall \alpha \in \mathrm{dom}(e) \;:\; e(\alpha) \in \mathrm{dom}(s_2(\alpha))\,\} \;.
\end{aligned}
$$

□

For any configuration s, the event function gives all possible events. Such an event $e \in E(s)$ unambiguously determines another configuration $Q(s, e)$, namely the resulting one when all communications are processed. Before we give $Q(s, e)$, we introduce two auxiliary functions.

Definition 3.4 β, μ
Let $s \in S$, $e \in E(s)$ and $\alpha \in \mathrm{dom}(e)$.

- $\beta(s, e, \alpha) \;:=\; B(s_1(\alpha), e(\alpha), s_2(\alpha)(e(\alpha)))$
- $\mu(s, e) \;:=\; \lambda\, \alpha \in \mathcal{A} \;:\; \{\, \langle t, \tilde{m}\rangle \mid \exists \alpha' \in \mathrm{dom}(e) \;:\; (t, \langle \alpha, \tilde{m}\rangle) \in \beta_1(s, e, \alpha')\,\}$

□

In this definition, $\beta(s,e,\alpha)$ is the behaviour of actor α when processing communication $e(\alpha)$ in configuration s and $\mu(s,e)$ gives for every actor name $\alpha \in \mathcal{A}$ the set of new communications with tags sent to α when event e occurs in configuration s.

In configuration $Q(s,e)$, each actor $\alpha \in \mathcal{A}$ has a set $\mu(s,e)(\alpha)$ of additional named communications. Moreover, $Q(s,e)$ differs from s in the following two aspects:

- Each $\alpha \in \mathrm{dom}(e)$ gets a new actor state with acquaintances, namely $\beta_3(s,e,\alpha)$. Furthermore, communication $e(\alpha)$ has been disappeared.
- For each $\alpha \in \mathrm{dom}(e)$ we have a set $\mathrm{dom}(\beta_2(s,e,\alpha))$ of new actors. Each new actor $\alpha' \in \mathrm{dom}(\beta_2(s,e,\alpha))$ gets $\beta_2(s,e,\alpha)(\alpha')$ as state with acquaintances.

These considerations give rise to next definition.

Definition 3.5 *Transition function, transition relation*
The transition function Q of an Actor system satisfies:

$$Q \in S \times \mathrm{rng}(E) \not\to S \quad \text{such that } \mathrm{dom}(Q) = \{\, \langle s,e\rangle \mid s \in S \wedge e \in E(s) \,\}$$

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

$$\begin{aligned} Q(s,e) \;=\; & \lambda\, \alpha \in \mathrm{dom}(e) \;:\; \langle \beta_3(s,e,\alpha), \\ & \qquad s_2(\alpha) \setminus \{(e(\alpha), s_2(\alpha)(e(\alpha)))\} \cup \mu(s,e)(\alpha)\rangle \\ & \cup\; \lambda\, \alpha \in \mathrm{dom}(s) \setminus \mathrm{dom}(e) \;:\; \langle s_1(\alpha),\, s_2(\alpha) \cup \mu(s,e)(\alpha)\rangle \\ & \cup\; \bigcup \{\, \lambda\, \alpha' \in \mathrm{dom}(\beta_2(s,e,\alpha)) \;:\; \langle \beta_2(s,e,\alpha)(\alpha'),\, \mu(s,e)(\alpha')\rangle \\ & \qquad \mid \alpha \in \mathrm{dom}(e) \,\} \end{aligned}$$

The transition relation T of an Actor system satisfies:

$$\begin{aligned} & T \subseteq S \times S \quad \text{and} \\ & T \;=\; \{\, \langle s, Q(s,e)\rangle \mid s \in S \wedge e \in E(s) \,\}\,. \end{aligned}$$

□

We do not formally prove the correctness of our transition function, i.e., $Q(s,e) \in S$. Instead we remark that for every new configuration, all acquaintances must be part of it since an actor sends messages to well-known actors only and these messages have no unknown actor names. Moreover, no name can be a prefix of another name because we adopted the name-generating mechanism of Agha. At last, the number of communications and actors must be finite, as an actor can only produce a finite amount of new communications and actors.
Please note: If L is a set of initial configurations of an Actor system with state space S and transition relation T, i.e. $L \subseteq S$, then the triple $\langle S, L, T\rangle$ is a transition system. This property enables us to compare an Actor system with a system described in a completely different model that also specifies a transition system, using the relationships

from Section 2. In Section 5, we implement any Actor system in the Des model and we compare the resulting Des with the original Actor system in this way.
We proceed with three examples of simple Actor systems.

Example 3.1 *Changing behaviour*
Consider the following informal description of a behaviour. For $n \in \mathbb{N}_0$:

$$\delta_n = \text{"Accept a natural number } m\text{, return to sender } m * n \text{ and become } \delta_{n+1}\text{."}$$

We construct an Actor system $\langle I, M, A, B \rangle$ in which actors with this behaviour may appear.

$$\begin{aligned} I &:= \{0,1,2\} \\ M &:= \mathbb{N}_0 \\ A &:= \mathbb{N}_0 \cup \{user\} \end{aligned}$$

We do not describe B for *user*. *User* is a so-called *external actor*.
We assume: An actor with the above described behaviour has no acquaintances and every communication sent to it has exactly one acquaintance, namely the sender.
For $n \in A \setminus \{user\}$; $t, \alpha \in I^*$ and $m \in M$:

$$\begin{aligned} B(\langle n, \emptyset \rangle, t, \langle m, \{\alpha\} \rangle) := \langle\ & \{(t \,\&\, 0, \langle \alpha, \langle n * m, \emptyset \rangle \rangle)\}, \\ & \emptyset, \\ & \langle n+1, \emptyset \rangle \\ \rangle\ . & \end{aligned}$$

We can play a transition game for this Actor system. Our configurations are constituted from partial functions and we have no method to picture them yet. Instead, we simply describe configurations by enumerating all partial functions involved.
Suppose we have an initial configuration s_0 with one user u and one actor b with behaviour δ_3, while b has just received two communications from u: A seven and an eight.

$$\begin{aligned} s_0 = \{ & (b,\ \langle \langle 3, \emptyset \rangle,\ \{(\langle 0 \rangle, \langle 7, \{u\} \rangle),\ (\langle 1 \rangle, \langle 8, \{u\} \rangle)\}\rangle), \\ & (u,\ \langle \langle user, \{b, u\} \rangle,\ \emptyset \rangle) \} \end{aligned}$$

Please notice that u has two acquaintances, namely b and itself.
Dependent on the selected communication, there are two possibilities for the next configuration s_1:

$$\begin{aligned} s_1 = \{ & (b,\ \langle \langle 4, \emptyset \rangle,\ \{(\langle 1 \rangle, \langle 8, \{u\} \rangle)\}\rangle), \\ & (u,\ \langle \langle user, \{b, u\} \rangle,\ \{(\langle 0, 0 \rangle, \langle 21, \emptyset \rangle)\}\rangle) \} \end{aligned}$$

or

$$\begin{aligned} s_1 = \{ & (b,\ \langle \langle 4, \emptyset \rangle,\ \{(\langle 0 \rangle, \langle 7, \{u\} \rangle)\}\rangle), \\ & (u,\ \langle \langle user, \{b, u\} \rangle,\ \{(\langle 1, 0 \rangle, \langle 24, \emptyset \rangle)\}\rangle) \}\ . \end{aligned}$$

Assuming only b consumes a communication, we have two possibilities for the last configuration s_2, too:

$$s_2 = \{(b,\ \langle\langle 5,\emptyset\rangle,\ \emptyset\rangle),\\ (u,\ \langle\langle user,\{b,u\}\rangle,\ \{(\langle 0,0\rangle,\langle 21,\emptyset\rangle),\ (\langle 1,0\rangle,\langle 32,\emptyset\rangle)\}\rangle)\}$$

or $$s_2 = \{(b,\ \langle\langle 5,\emptyset\rangle,\ \emptyset\rangle),\\ (u,\ \langle\langle user,\{b,u\}\rangle,\ \{(\langle 0,0\rangle,\langle 28,\emptyset\rangle),\ (\langle 1,0\rangle,\langle 24,\emptyset\rangle)\}\rangle)\}\ .$$

In the end, u has received either a 21 and a 32 or a 28 and a 24. *Nondeterminism* has come up.
□

The following example has been taken from [1]:

Example 3.2 *Recursive factorial*
We construct an Actor system $\langle I, M, A, B\rangle$ with three kinds of actors:

- Users;
- Factorial actors;
- Customers.

Users send communications to factorial actors. Again, users are external actors whose behaviour we shall not formalize.

$$\begin{aligned} I &:= \{0,1\} \\ M &:= I\!N_0 \qquad \text{(Only numbers need be send.)} \\ A &:= \{user, fac\} \cup I\!N_1 \end{aligned}$$

Factorial actors have one acquaintance, namely their own name. Communications arriving at a factorial actor have merely the sender as acquaintance.
For $m, t, \alpha \in I^*$ and $n \in M$:

$$\begin{aligned} B(\langle fac,\{m\}\rangle,\ t,\ \langle n,\{\alpha\}\rangle) \ :=\ & \underline{\text{if}}\ n = 0 \\ & \underline{\text{then}}\ \langle\ \{(t\,\&\,0, \langle\alpha, \langle 1,\emptyset\rangle\rangle)\}, \\ & \qquad \emptyset, \\ & \qquad \langle fac,\{m\}\rangle \\ & \qquad\ \rangle \\ & \underline{\text{else}}\ \langle\ \{(t\,\&\,0, \langle m, \langle n-1, \{t\,\&\,1\}\rangle\rangle)\}, \\ & \qquad \{(t\,\&\,1, \langle n,\{\alpha\}\rangle)\}, \\ & \qquad \langle fac,\{m\}\rangle \\ & \qquad\ \rangle \\ & \underline{\text{fi}}\ . \end{aligned}$$

A customer also has one acquaintance, namely the actor which a response must be sent to. Communications destined for customers have no acquaintances. For $n \in \mathbb{N}_1$, $k \in M$ and $\alpha, t \in I^*$:

$$B(\langle n, \{\alpha\}\rangle, t, \langle k, \emptyset\rangle) := \langle \{(t \& 0, \langle \alpha, \langle n * k, \emptyset\rangle\rangle)\}, \emptyset, \langle n, \{\alpha\}\rangle \rangle .$$

Please note: The replacement behaviour of a customer is irrelevant, for a customer receives no communications anymore after the first one.
Let us play the transition game again. Assume we have an initial configuration s_0 with an user u and a factorial actor m, where u has just send a 3 to m.

$$\begin{aligned} s_0 = \{&(m, \langle\langle fac, \{m\}\rangle, \{(t, \langle 3, \{u\}\rangle)\}\rangle), \\ &(u, \langle\langle user, \{m, u\}\rangle, \emptyset\rangle)\} \end{aligned}$$

The system then evolves to final configuration s_7.

$$\begin{aligned} s_7 = \{&(m, \langle\langle fac, \{m\}\rangle, \emptyset\rangle), \\ &(t \circ \langle 0,0,1\rangle, \langle\langle 1, \{t \circ \langle 0,1\rangle\}\rangle, \emptyset\rangle), \\ &(t \circ \langle 0,1\rangle, \langle\langle 2, \{t \circ \langle 1\rangle\}\rangle, \emptyset\rangle), \\ &(t \circ \langle 1\rangle, \langle\langle 3, \{u\}\rangle, \emptyset\rangle), \\ &(u, \langle\langle user, \{m, u\}\rangle, \{(t \circ \langle 0,0,0,0,0,0,0\rangle, \langle 6, \emptyset\rangle)\}\rangle)\} . \end{aligned}$$

Configuration s_7 has three redundant actors: $t \circ \langle 1\rangle$, $t \circ \langle 0,1\rangle$ and $t \circ \langle 0,0,1\rangle$. These actors could as well be deleted from the system.
□

Our model allows an actor to deterministically select among acquaintance names, because the set of names has been ordered lexicographically. In Example 3.3 we have constructed a behaviour *sel* for returning messages to the acquaintance with lowest name.

Example 3.3 *Acquaintance selection*
Let $\langle I, M, A, B\rangle$ be an Actor system with additional properties:

$$0 \in I ;$$
$$sel \in A$$

and for $t \in I^*$, $m \in M$ and $V \in \mathbb{P}(I^*)$,

$$B(\langle sel, \emptyset\rangle, t, \langle m, V\rangle) = \underline{\text{if}}\ V = \emptyset \ \underline{\text{then}}\ \langle \emptyset, \emptyset, \langle sel, \emptyset\rangle \rangle$$

$$
\begin{array}{ll}
\underline{\text{else}} & \langle\ \{(t \,\&\, 0,\ \langle min(V), \langle m, V\rangle\rangle)\},\\
& \quad \emptyset,\\
& \quad \langle sel, \emptyset\rangle\\
& \rangle\\
\underline{\text{fi}}\ . &
\end{array}
$$

We assume an actor with local state *sel* to have no acquaintances itself. Such an actor selects the lowest actor name out of the set of received acquaintance names and returns the communication to it.
□

Serializability of events

An event in which several actors process communications can be split up into a number of events, all occurring after each other. The resulting configuration is the same in both cases. We call this property *serializability of events*. In order to prove it, we first introduce several lemmas.

Events that may occur in a given configuration might also occur if new actors and new communications are added. In both cases, the behaviour of an actor being part of such an event is equal. This is expressed in Lemma 3.2.

Lemma 3.2
$\forall s, s' \in S\ :\ \forall e \in E(s)\ :$
If $\mathrm{dom}(s) \subseteq \mathrm{dom}(s')$ and $\forall \alpha \in \mathrm{dom}(e)\ :\ s_1(\alpha) = s'_1(\alpha)\ \wedge\ s_2(\alpha) \subseteq s'_2(\alpha)$, then:

- $e \in E(s')$;
- $\forall \alpha \in \mathrm{dom}(e)\ :\ \beta(s, e, \alpha) = \beta(s', e, \alpha)$;
- $\forall \alpha \in \mathcal{A}\ :\ \mu(s, e)(\alpha) = \mu(s', e)(\alpha)$.

Proof.
The first point directly follows from Definition 3.3.
The second property can be directly derived from the definition of β and the last assertion is an immediate consequence of the second one.
□

Next lemma is a direct consequence of the definition of β:

Lemma 3.3
$\forall s \in S\ :\ \forall e, e' \in E(s)\ :\ \forall \alpha \in \mathrm{dom}(e \cap e')\ :\ \beta(s, e, \alpha) = \beta(s, e', \alpha)$.
□

Assume we split up an event e of a configuration s into two events e' and e''. The set of new communications that results when e is processed in s, equals the union of these sets if e' and e'' were processed in s.

Lemma 3.4
$\forall s \in S \ : \ \forall e \in E(s) \ : \ \forall D_1, D_2 \in \mathrm{dom}(e) \ :$
If $D_1 \cup D_2 = \mathrm{dom}(e)$ and $D_1 \cap D_2 = \emptyset$ and $D_1, D_2 \neq \emptyset$, then:

- $e \lceil D_1 \in E(s)$ and $e \lceil D_2 \in E(s)$;
- $\forall \alpha \in \mathcal{A} \ : \ \mu(s,e)(\alpha) = \mu(s, e \lceil D_1)(\alpha) \cup \mu(s, e \lceil D_2)(\alpha)$.

Proof: See appendix.
□

If actor α creates a new actor α' in event e then α is the only one in e being able to send communications to α'. This property is a special case of next lemma, that is, if D is a singleton.

Lemma 3.5
$\forall s \in S \ : \ \forall e \in E(s) \ : \ \forall D \subseteq \mathrm{dom}(e) \ :$
$\forall \alpha \in D \ : \ \forall \alpha' \in \mathrm{dom}(\beta_2(s,e,\alpha)) \ : \ \mu(s,e)(\alpha') = \mu(s, e \lceil D)(\alpha')$.

Proof: See appendix.
□

We are now able to prove the serializability property.

Theorem 3.2 *Serializability of events*
Let $s \in S$, $e \in E(s)$, D_1 and $D_2 \subseteq \mathrm{dom}(e)$ and suppose $\mathrm{dom}(e) = D_1 \cup D_2$, $D_1 \cap D_2 = \emptyset$, $D_1 \neq \emptyset$ and $D_2 \neq \emptyset$.
Then $e \lceil D_1 \in E(s)$, $e \lceil D_2 \in E(Q(s, e \lceil D_1))$ and

$$Q(s,e) \ = \ Q(Q(s, e \lceil D_1), e \lceil D_2) \ .$$

Proof: See appendix.
□

As an Actor system evolves, some present actors might get superfluous. The last configuration of Example 3.2 included three customers which would never receive a communication anymore. Another example regarding redundant actors is concerned with *forwarders* [1], i.e. actors that only forward received communications to other ones. An interesting topic for future research would be to give sufficient conditions for deleting an actor from the system. Such an Actor system might ultimately reach an empty configuration, i.e. a configuration without actors.
In the next section we introduce briefly our model of discrete event systems, called *Des model.*

4 Des model

In this section we describe a mathematical model for discrete event systems, called the *Des model*. We shall not extensively discuss its motivations and properties, instead we refer to [5].
Like Predicate/Transition nets [4] and Coloured nets [10], the Des model is an extension of the elementary Petri Net model [17]. A major difference with the Actor model from the previous section is the topology: In a Des, the number of components and their interaction structure are fixed, whereas an Actor system exhibits dynamic process creation. We have developed a tool called EXSPECT [6,7] to specify and simulate a Des. Hence, specifications of systems described as Des are suitable for prototyping. We start with an informal treatment.

A Des consists of two kinds of components: *Processors* and *channels*, which correspond to transitions and places in Petri nets. A processor is connected with several input and output channels. To each channel a type is associated and to each processor a function. The signature of the function of a processor is such that the types of the input parameters are identical to the types of the input channels and the types of the output parameters of the function correspond to the types of the output channels. A channel may be shared by several processors as input or output channel. The channels may contain so-called *triggers* (tokens in Petri nets). A trigger has a value that belongs to the type of the channel. More than one trigger of the same value may reside in a channel, so a channel contains a bag over its type (cf. notations).
For each processor, every input channel has a multiplicity, which gives the number of triggers the processor needs from that channel to operate. At each moment a *transition* can occur, which means that the configuration of triggers in the channels may change. Such a transition happens instantaneously and is *executed* by the processors. A processor that has enough triggers in each input channel may pick as many triggers as it needs and produce a finite amount of new triggers for its output channels, according to its function. Several processors may produce triggers for the same output channel. A channel having always exactly one trigger can be seen as a memory of the system. We call such a channel a *store*. If a processor wants to use the store, it picks out the trigger and instantaneously places back a new trigger into the store. In fact, it replaces the trigger. A store can even be a database. In this case, its type is very complex.
A configuration of triggers distributed over channels is called a *state*.

The Des model actually resembles Coloured nets very much. The selection of tokens differs, in our model a processor has to accept all triggers, whatever their values may be, whereas in Coloured nets a transition might refuse certain tokens because of their colours. But we can simulate such a refusal by simply placing back undesirable triggers.

The systems we consider may be part of another, much larger system. In this case, communications take place from our system to its environment and vice versa. We regard the environment as a Des too, i.e. as a system with processors and channels, but we do not know the specification of those processors and the channel structure. Processors with an unknown specification are called *black box processors*. A black box processor which only produces triggers for our well-known system may be modeled as a single channel containing an infinite amount of triggers. Such a channel represents an *input stream*. It will never get empty, since a processor consumes only a finite amount

of triggers in a finite time interval.
To indicate the processors and channels and their input and output relations, we use a diagram technique where processors are represented by triangles and channels by circles. For input/output relations we use arrows and for each input channel we mention its multiplicity, except for input channels with multiplicity one, which is the default value. See Figure 4.1 for an example.
We now formalize the Des framework.

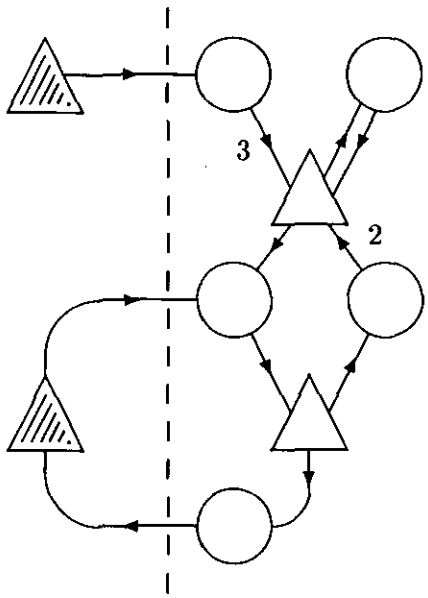


Figure 4.1

Definition 4.1 *Discrete event system*
A discrete event system (Des) is a quadruple $\langle R, C, I, O \rangle$, where R is a function-valued function, C and O are set-valued functions and I is a bag-valued function, such that:

- $\mathrm{dom}(I) = \mathrm{dom}(O) = \mathrm{dom}(R)$, finite or countable sets.
- $\forall i \in \mathrm{dom}(R) \ : \ I(i) \in I\!B(\mathrm{dom}(C)) \wedge I(i) \neq \emptyset \wedge I(i)$ is finite
  $\wedge \ O(i) \in I\!P(\mathrm{dom}(C))$.
- $\forall i \in \mathrm{dom}(R) \ : \ R(i) \in \{b \in I\!B(\{\langle c, w \rangle \mid c \in \mathrm{dom}(C) \wedge w \in C(c)\})$
  $\mid \forall c \in \mathrm{dom}(C) \ : \ \sum_{w \in C(c)} b(\langle c, w \rangle) = I(i)(c)\}$
  $\rightarrow I\!B(\{\langle c, w \rangle \mid c \in O(i) \wedge w \in C(c)\})$.
- $\forall i \in \mathrm{dom}(R) \ : \ \forall b \in \mathrm{dom}(R(i)) \ : \ R(i)(b)$ is finite.
- $\forall i \in \mathrm{dom}(C) \ : \ C(i)$ is finite or countable.

$\mathrm{Dom}(R)$ is called the set of *processor indices*, denoted by P,

$\mathrm{dom}(C)$ is called the set of *channel indices*, denoted by K
and for all $p \in P$, $k \in K$:

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

□

We shall use these symbols strictly for the concepts defined. When we consider different Des'ses we distinguish them by means of subscripts.

Definition 4.2 *Trigger set, state space, event set*
Let a Des be given. Then

$$
\begin{aligned}
Q &:= \{\langle k, w\rangle \mid k \in K \wedge w \in C(k)\} \\
S &:= \mathbb{B}(Q) \\
E &:= \{e \in P \not\to \mathbb{B}(Q) \mid \mathrm{dom}(e) \neq \emptyset,\ \mathrm{dom}(e) \text{ is finite} \\
&\qquad\qquad \text{and } \forall p \in \mathrm{dom}(e) \ :\ e(p) \in \mathrm{dom}(R(p))\}\ .
\end{aligned}
$$

Q is called the *trigger set*, S is called the *state space* and E is called the *event set.*
□

Please note that an event is an assignment of a bag of triggers to a processor such that for each input channel k with multiplicity m exactly m triggers are chosen.

Definition 4.3 *Event function*
The event function F of a Des satisfies:

$$
\begin{aligned}
&F \in S \to \mathbb{P}(E) \quad \text{and} \\
&\forall s \in S \ :\ F(s) = \{e \in E \mid \mathrm{mrng}(e) \subseteq s\}\ .
\end{aligned}
$$

□

Hence, $e \in F(s)$ only if s contains enough triggers to supply all the processors of $\mathrm{dom}(e)$. Note that $e \in F(\mathrm{mrng}(e))$, for all events $e \in E$. It is easy to verify that for all s, t in S: If $s \subseteq t$ then $F(s) \subseteq F(t)$.
We next define the *transition function*, which assigns to a state s and an event $e \in F(s)$ a new state.

Definition 4.4 *Transition function, transition relation*
The transition function T of a Des satisfies:

$$T \in S \times E \nrightarrow S \quad \text{such that } \mathrm{dom}(T) = \{\langle s, e\rangle \mid e \in F(s)\}$$
$$\text{and for } s \in S,\ e \in F(s):$$
$$T(s,e) = s \setminus \mathrm{mrng}(e) \cup \bigcup_{p \in \mathrm{dom}(e)} R(p)(e(p)) .$$

The transition relation of a Des is:

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

Elements of the transition relation are called *transitions.*
□

We also use the symbol T to denote the transition relation. It is easy to verify that the graph $\langle S, L, T\rangle$, where S is the state space of a Des, L a collection initial states and T its transition relation, forms a transition system.
A Des cannot evolve from a finite state (i.e. a state with finitely many triggers) to an infinite state or vice versa. This is expressed by Lemma 4.1.

Lemma 4.1
Let T be the transition relation of a Des. Then

$$\forall \langle s, s'\rangle \in T : s \text{ is finite} \Leftrightarrow s' \text{ is finite} .$$

Proof.
Each processor consumes and produces a finite amount of triggers and in an event only finitely many processors perform an action.
□

We have true parallelism in our model: Processors may execute simultaneously. As in an Actor system, we have serializability of events, too. Without proof we mention that it is always possible to split up an event for more than one processor into other events, such that the successive execution of these events, in any order, ends in the same state as the original compound event.

Theorem 4.1 *Serializability of events*
Let a Des be given and let $s \in S$ and $e \in F(s)$ such that $|\mathrm{dom}(e)| \geq 2$. Let further $D_1, D_2 \subset \mathrm{dom}(e)$ such that $|D_1| \geq 1$, $|D_2| \geq 1$, $D_1 \cap D_2 = \emptyset$ and $D_1 \cup D_2 = \mathrm{dom}(e)$. Then $e \restriction D_1 \in F(s)$, $e \restriction D_2 \in F(T(s, e \restriction D_1))$ and

$$T(T(s, e \restriction D_1), e \restriction D_2) = T(s,e) .$$

□

Please notice the resemblance with Theorem 3.2. For more results on the Des model, see [5]. In the next section we show that any Actor system can be implemented in an equivalent Des.

5 Implementation of Actor systems as Des'ses

In this section, we present two implementations of any Actor system as Des. The first one is really very simple, we use only one processor here. Consequently, all parallelism is lost and this Des only realizes the original Actor system. But it might be useful in order to very quickly obtain a prototype of the Actor system. The second one will be equivalent with the Actor system, thus maintaining all parallelism, but it has processors with infinitely many output channels and therefore it needs some adjustments to make it suitable for EXSPECT. This will be done in two steps, yielding a Des that simulates the original Actor system.

5.1 First implementation

In this subsection, we implement an arbitrary Actor system in a simple Des. Let an Actor system $\langle I, M, A, B\rangle$ with transition system $\langle S, L, T\rangle$ be given. We shall construct a Des $\langle RR, CC, II, OO\rangle$ with transition system $\langle SS, LL, TT\rangle$ that realizes the Actor system. To avoid ambiguities we denote symbols referring to the Actor system with one character and symbols referring to the Des with two identical characters.

The Des consists of one processor and two channels. See Figure 5.1. Channel *com* contains communications. Channel *act* is a store for a table of actor names with corresponding behaviours (local states). Processor p repeatedly takes a single communication from *com*. This communication is destined for some actor α in the table of *act*. Then p processes the communication: New communications are send to *com*, the behaviour of α is updated and possible new actors are added to the table.

Figure 5.1 *The des*

Now we formalize Des $\langle RR, CC, II, OO\rangle$. It is constructed from the Actor system $\langle I, M, A, B\rangle$:

$$CC = \{(com,\ \mathcal{T} \times (\mathcal{A} \times \widetilde{M})),\ (act,\ \mathcal{A} \not\to \tilde{A})\}\ ,$$
$$II = OO = \{(p,\ \{com,\ act\})\}$$

and for $t \in \mathcal{T}$, $\alpha \in \mathcal{A}$, $\tilde{m} \in \widetilde{M}$ and $f \in \mathcal{A} \not\to \tilde{A}$:

$$RR(p)(\{(com,\ \langle t, \langle \alpha, \tilde{m}\rangle\rangle),\ (act, f)\}) =$$
$$\{(com,\ B_1(f(\alpha), t, \tilde{m})),$$
$$(act,\ f[\alpha : B_3(f(\alpha), t, \tilde{m})] \cup B_2(f(\alpha), t, \tilde{m})\}$$

Please review Section 3 for those symbols not defined here. Please note that a (partial) function can be considered as a set of tuples.

The Des has been defined completely apart from its initial states. It implicitly defines a state space SS and a transition relation TT. However, not every possible state $s \in SS$ is allowed. Communications from *com* have a communication name and the name of the destination actor. Every such actor name should of course be present in the table of *act*. The same should hold for the names of acquaintances, both actor acquaintances and communication acquaintances. Furthermore, no single (actor or communication) name may be the prefix of another name, particularly no name may be present more than once. Next, the number of actors and communications should be finite. Finally, *act* should store exactly one partial function. We shall choose the initial states such that all these constraints are met. From the definition of Actor systems it follows that all these properties remain invariant when the Des evolves, if it is initiated correctly. For the Des we define a set AS of allowed states:

$$
\begin{aligned}
AS \;=\; \{\; & \{(com,\, b),\ (act,\, \{f\})\} \in SS \\
& \mid\; b \in I\!B(T \times (\mathcal{A} \times \widetilde{M})),\ f \in \mathcal{A} \not\rightarrow \widetilde{A} \ \text{ and} \\
& \quad - \ \forall \langle t, \langle \alpha, \langle m, V \rangle \rangle \rangle \in b \ :\ \alpha \in \mathrm{dom}(f) \ \wedge\ V \subseteq \mathrm{dom}(f) \\
& \quad - \ \forall \langle a, V \rangle \in \mathrm{rng}(f) \ :\ V \subseteq \mathrm{dom}(f) \\
& \quad - \ b \text{ and } \mathrm{dom}(f) \text{ are finite} \\
& \quad - \ \forall \alpha, \alpha' \in \mathrm{dom}(f) \ :\ \forall \langle t, \tau \rangle,\ \langle t', \tau' \rangle \in b \ : \\
& \qquad\qquad \alpha \neq \alpha' \ \wedge\ t \neq t' \ \Rightarrow \\
& \qquad\qquad \neg \exists x \in I^* \ :\ \alpha \circ x = \alpha' \ \vee\ t \circ x = t' \ \vee\ \alpha \circ x = t \ \vee\ t \circ x = \alpha \\
& \}
\end{aligned}
$$

We also define a mapping $g \in AS \rightarrow S$, where S is the state space of the Actor system:

$$
\begin{aligned}
g \;:=\; & \lambda\, \{(com,\, b),\ (act,\, \{f\})\} \in AS \ : \\
& \qquad (\ \lambda\, \alpha \in \mathrm{dom}(f) \ :\ \langle f(\alpha),\ \{(t, \widetilde{m}) \mid \langle t, \langle \alpha, \widetilde{m} \rangle \rangle \in b\} \rangle\ )\ .
\end{aligned}
$$

It can easily be verified that g is a total, surjective and injective function.
We finally define the set of initial states LL of the Des:

$$
LL \;:=\; g^{-1}(L)\ .
$$

Hence, $LL \subseteq AS$ and $g(LL) = L$. Without proof we mention that the Des cannot escape from the set of allowed states. It follows from the construction of the Des and the constraints on B (see Definition 3.1).

Lemma 5.1
$\forall s \in AS \ :\ \forall s' \in SS \ :\ \langle s, s' \rangle \in TT \Rightarrow s' \in AS.$
□

The Des realizes the Actor system.

Theorem 5.1
$\langle SS, LL, TT\rangle$ realizes $\langle S, L, T\rangle$ with g.

Proof: See appendix.
□

Consequently, all the actions of the Des are correct. Actually, the Des can perform each one-step action of the Actor system, i.e. every action of a single actor. As any Actor system has serializability of events, the Des can reach every possible state the Actor system might evolve to. So this prototype is not too bad, despite of the lost of parallelism. It's no use adding one or more processors in parallel with p, since only one processor at a time can use store *act*.
We proceed with a better implementation of the Actor model into the Des model, where all parallelism will be maintained.

5.2 Second implementation

Again, we consider an Actor system $\langle I, M, A, B\rangle$ with transition system $\langle S, L, T\rangle$. We construct a Des $D' = \langle R', C', I', O'\rangle$ with transition system $\langle S', L', T'\rangle$ and we shall prime every symbol referring to D'.
D' will be a chain consisting of a countable number of processors and channels. See Figure 5.2. Each processor has a name α and two input channels with multiplicity one. One of them, c_α, is used to collect communications for α and the other one, b_α, is a kind of pseudo-store for the behaviour of α. This channel will contain at most one trigger. In case b_α is empty, α is not (yet) present in the Actor system, otherwise α really exists. The actor population will always be finite when the system starts with a finite one, because an actor produces only finitely many new actors.
Each processor has every channel in the system as output channel, so it can send communications to every processor in the system and it can initiate another processor α' by sending a trigger to $b_{\alpha'}$. Of course, $b_{\alpha'}$ must then be empty, but this is guaranteed by construction. The equivalence of both systems will be proved.
We start with a formal specification of D'.

$D' := \langle R', C', I', O'\rangle$, where:

$$\mathrm{dom}(R') = \mathcal{A} = I^* ; \quad \mathrm{dom}(C') = \bigcup_{\alpha\in\mathcal{A}} \{c_\alpha, b_\alpha\} ;$$

$$C' = \bigcup_{\alpha\in\mathcal{A}} \{(c_\alpha, \mathcal{T}\times\widetilde{M}), (b_\alpha, \widetilde{A})\} ;$$

$$I' = \bigcup_{\alpha\in\mathcal{A}} \{(\alpha, \{c_\alpha, b_\alpha\})\} ;$$

$$O' = \bigcup_{\alpha\in\mathcal{A}} \{(\alpha, \mathrm{dom}(C'))\}$$

and for $\alpha \in \mathcal{A}$, $t \in \mathcal{T}$, $\tilde{m} \in \widetilde{M}$ and $\tilde{a} \in \widetilde{A}$:

$$
\begin{aligned}
R'(\alpha)(\{\langle c_\alpha, \langle t, \tilde{m}\rangle\rangle,\ \langle b_\alpha, \tilde{a}\rangle\}) \ = \\
\quad \{\langle c_{\alpha\alpha}, \langle tt, \widetilde{m\,m}\rangle\rangle \mid (tt, \langle \alpha\alpha, \widetilde{m\,m}\rangle) \in B_1(\tilde{a}, t, \tilde{m})\} \cup \\
\quad \{\langle b_{\alpha\alpha}, \widetilde{a\,a}\rangle \mid (\alpha\alpha, \widetilde{a\,a}) \in B_2(\tilde{a}, t, \tilde{m})\} \cup \\
\quad \{\langle b_\alpha, B_3(\tilde{a}, t, \tilde{m})\rangle\}
\end{aligned}
$$

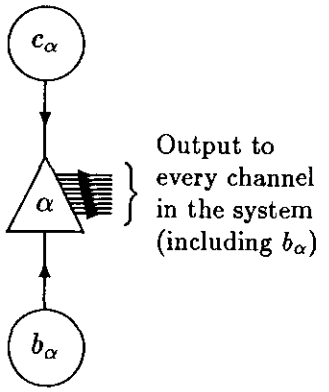
Dom(R') is countable, because I is finite. D' implicitly defines a state space S', an event function F' and a transition relation T'. Before we define a set of initial states for D', we give a function $g \in S \to S'$ which maps every configuration of the Actor system onto a state of the Des.

$$
\begin{aligned}
g \ := \ \lambda\, s \in S \ : \ & \{\langle b_\alpha, s_1(\alpha)\rangle \mid \alpha \in \mathrm{dom}(s)\} \cup \\
& \{\langle c_\alpha, \langle t, \tilde{m}\rangle\rangle \mid \alpha \in \mathrm{dom}(s) \wedge (t, \tilde{m}) \in s_2(\alpha)\} \ .
\end{aligned}
$$

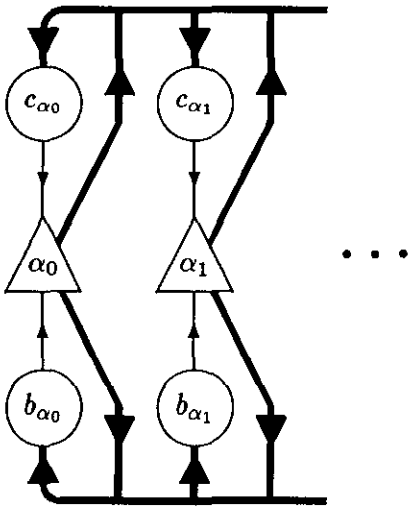
The set of initial states L' will then be:

$$L' \ := \ g(L) \ .$$

D' consists of countable many identical processors. Figure 5.2 shows a schematic reproduction. A thick line represents an infinite number of input/output relations.



A single processor



Des D'

Figure 5.2

Lemma 5.2
g is injective.

Proof.
This can easily be seen by interpreting g: For each actor α in the configuration, its behaviour is mapped on pseudo-store b_α and its communications are mapped on channel

c_α. For each actor name α' that is not part of the configuration, $b_{\alpha'}$ and $c_{\alpha'}$ are empty. Hence, no two different configurations of the Actor system are mapped onto the same state of D'.
□

Events of the Actor system can be transformed into events of the Des and vice versa.

Lemma 5.3
$\forall s \in S \ : \ \exists h \in E(s) \rightarrow F'(g(s)) \ : \ h$ is bijective.

Proof: See appendix.
□

Now we can prove the equivalence of both systems.

Theorem 5.2
$\langle S, L, T \rangle \cong \langle S', L', T' \rangle$.

Proof: See appendix.
□

We have constructed an equivalent Des D' for each Actor system. It has a processor α for each actor name $\alpha \in \mathcal{A}$. Not every processor will be used. For example, consider some channel with a communication named $t \in \mathcal{T}$. Since $\mathcal{A} = \mathcal{T}$, a processor t exists. Due to the uniqueness of names, it will never be initiated in this case.
The number of processors in D' is, of course, infinite. We have chosen the initial states such that D' always starts with a finite actor population and we have already mentioned that this population will always remain finite. The only problem with D' is that it consists of processors with infinitely many output channels. In the remainder of this section, we describe a method to simulate these processors with systems consisting of processors with finitely many output channels, which enables us to actually implement D' in EXSPECT.

5.3 Processors with countable many output channels

The Des model allows to specify processors with countable many output channels. In this subsection, we decompose such a Des into another one where each processor has only finitely many output channels. So this subsection differs from the previous two in the sense that we are not transforming Actor systems into Des'ses, we are only considering Des'ses among themselves.

Let a Des be given with countable many processors, all of them being able to communicate with each other. Such a processor has infinitely many output channels. We assume each processor to have one input channel. This case can easily be generalized for processors with more input channels.
We show that such a Des can be decomposed into another one where each processor has only finitely many output channels. We shall use a two-steps construction. First,

we construct another system having one additional processor and channel, where each originally present processor has left only one output channel. The added processor, however, still has countable many output channels. Step two is to overcome this problem. We prove that the thus arisen Des simulates the original one.

5.3.1 Step one

In a system with processors with countable many output channels, we can built in a new channel in which those processors can place their triggers. We shall call this channel *collect*. Furthermore, we need a processor, called *distributor*, that has to transport these triggers to their destination. Then *distributor* will still have countable many output channels, but in the new system it will be the only one. Step two (see below) shows a way to also replace this one.

We start with a formal specification of the original system (Des D), where we require each processor to have exactly one input channel. Next, we construct the new system (Des D') and we shall prime every symbol referring to it. After that we prove that D' simulates D.

The original system

$D = \langle R, C, I, O \rangle$ with processors $P = \{p_0, p_1, \ldots\}$; channels $K = \{c_0, c_1, \ldots\}$;

$$I = \bigcup_{i \in \mathbb{N}_0} \{(p_i, \{c_i\})\} \; ; \quad \text{(Each processor has a single input channel with multiplicity one)}$$

$$O = \bigcup_{i \in \mathbb{N}_0} \{(p_i, K)\} \; ; \quad \text{(Output to every channel)}$$

and set of initial states L. See Figure 5.3. D implicitly defines a trigger set Q, a state space S, an event function F and a transition relation T.

The new system

$D' = \langle R', C', I', O' \rangle$ with processors $P' = P \cup \{\mathit{distributor}\}$; $K' = K \cup \{\mathit{collect}\}$;

$$C' = C \cup \{(\mathit{collect}, Q)\} \; ;$$

$$I' = I \cup \{(\mathit{distributor}, \{\mathit{collect}\})\} \; ;$$

$$O' = \{(\mathit{distributor}, K)\} \cup \lambda\, p \in P \; : \; \{\mathit{collect}\}$$

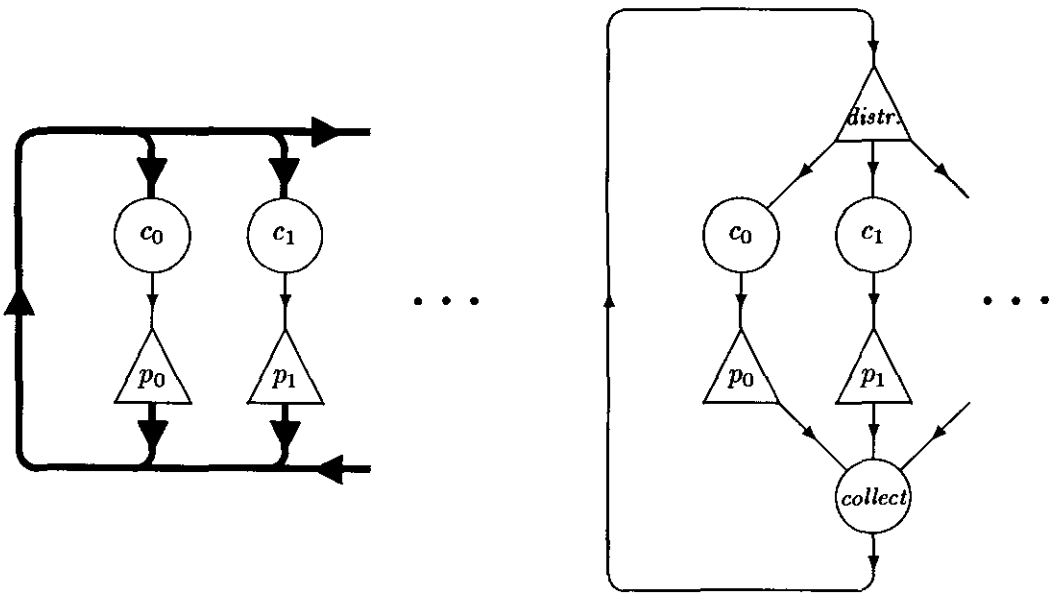
and for $i \in \mathbb{N}_0$ and $w \in C(c_i)$:

$$R'(\mathit{distributor})(\{\langle \mathit{collect}, \langle c_i, w \rangle \rangle\}) = \{\langle c_i, w \rangle\} \; ,$$

$$R'(p_i)(\{\langle c_i, w \rangle\}) = \{\langle \mathit{collect}, q \rangle \mid q \in R(p_i)(\{\langle c_i, w \rangle\})\} \; .$$

See Figure 5.3. The set of initial states will be: $L' := L$. Please note: *Collect* is always

empty at start.
D' implicitly defines a trigger set Q', a state space S', an event function F' and a transition relation T'.



System D System D'

Figure 5.3

Theorem 5.3
D' simulates D.

Proof: See appendix.
□

This result allows us to replace every system D with a system D'. We remark that this transformation can easily be generalized for processors with two input channels instead of one. Applied on the second implementation of the previous section it yields a system with only two processors with infinitely many output channels: A distributor for communications and a distributor for behaviours. The next step concerns these two.

5.3.2 Step two

This step addresses the problem of replacing a single processor with countable many output channels with a chain of simple processors with finitely many output channels. Consequently, this chain will be of infinite length. Such a chain of processors can be implemented. The implementation starts with an empty chain and when the system evolves, this chain grows larger and larger. Whenever we need a new output channel, we

activate a new processor, while we assume the existence of an infinite, but countable chain of relatively simple processors on a conceptual level. Such a simple processor accepts various communications and picks out those destined for a particular output channel. Other communications are simply forwarded.
We could actually do without step one, but that would lead towards an enormous burst of new processors and channels: Every single processor should then be replaced with an infinite amount of other processors, while the original system already consisted of infinitely many processors.
We elaborate this step in the same way as the former one.

The processor with countable many output channels

$D = \langle R, C, I, O\rangle$ with $p_0 \in P$; $\{h_0\} \cup \{c_0, c_1, \ldots\} \subseteq K$;

$$C \supseteq \{(h_0, I\!N_0 \times Y)\} \cup \bigcup_{i \in I\!N_0} \{(c_i, Y)\} ;$$

$$I \supseteq \{(p_0, \{h_0\})\} ; \qquad \text{(The input channel has multiplicity one)}$$

$$O \supseteq \{(p_0, \{c_0, c_1, \ldots\})\}$$

and for $n \in I\!N_0$ and $y \in Y$:

$$R(p_0)(\{\langle h_0, \langle n, y\rangle\rangle\}) = \{\langle c_n, y\rangle\}$$

where Y is an arbitrary type. Des D has a set of initial states L and embodies some unknown part. See Figure 5.4.

The chain

$D' = \langle R', C', I', O'\rangle$ with $P' = P \cup \{p_1, p_2, \ldots\}$; $K' = K \cup \{h_1, h_2, \ldots\}$;

$$C' = C \cup \bigcup_{i \in I\!N_1} \{(h_i, I\!N_i \times Y)\} ;$$

$$I' = I \cup \bigcup_{i \in I\!N_1} \{(p_i, \{h_i\})\} ;$$

$$O' = O \restriction (P \setminus \{p_0\}) \cup \bigcup_{i \in I\!N_0} \{(p_i, \{h_{i+1}, c_i\})\}$$

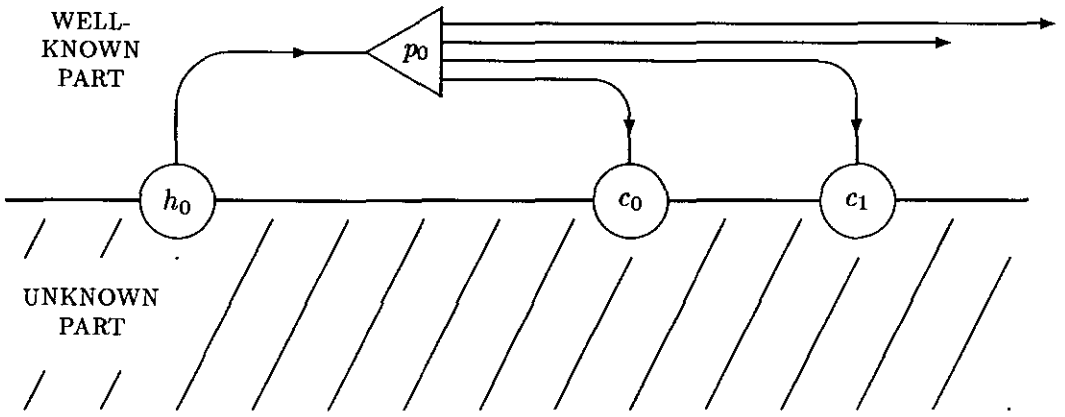
and for $i \in I\!N_0$, $n \in I\!N_i$ and $y \in Y$:

$$R'(p_i)(\{\langle h_i, \langle n, y\rangle\rangle\}) =$$
$$\underline{\text{if}}\ i = n$$
$$\underline{\text{then}}\ \{\langle c_i, y\rangle\}$$
$$\underline{\text{else}}\ \{\langle h_{i+1}, \langle n, y\rangle\rangle\}$$
$$\underline{\text{fi}}\ .$$

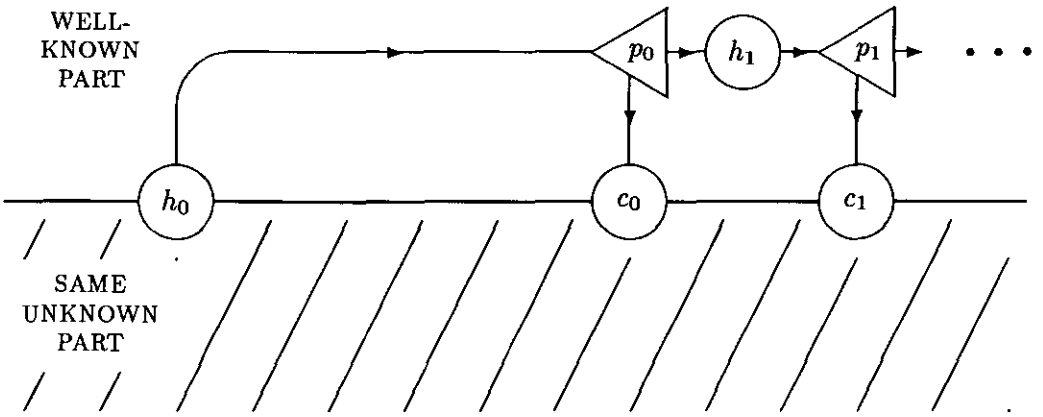
All other processors remain unchanged. For $p \in P \setminus \{p_0\}$ and $b \in \text{dom}(R(p))$:

$$R'(p)(b) \;=\; R(p)(b)\ .$$

See Figure 5.4. The initial states: $L' := L$. Hence, channels h_1, h_2, ... are initially empty.



System D



System D'

Figure 5.4

Theorem 5.4
D' simulates D.

Proof: See appendix.
□

The above result allows us to replace every such processor having countable many output channels by a chain of processors with only finitely many output channels.

If we apply both steps to the second implementation, then we obtain a Des that simulates the original Actor system because of the transitivity of the simulation relation. Moreover, it can be implemented in EXSPECT.

6 Concluding remarks

In this paper, we first introduced transition systems to describe discrete event systems in general and we formalized and analyzed their behaviours. After that we presented two formal models for discrete event systems: The Actor model and the Des model. The main difference is their topology: The former exhibits dynamic process creation, whereas the latter has a fixed interaction structure. The semantics of both models are in terms of transition systems, which gives a powerful tool to compare Actor systems and Des'ses with each other.
In Section 3, we developed the Actor model without using recursive definitions. We proved the serializability of events. Our model is well-suited to prove other properties of Actor systems as well. Furthermore, we proved that every name-generating mechanism is isomorphic to the one introduced by Agha.
Section 4 gives a brief overview of the Des model. Despite of its static topology, we constructed in Section 5 an equivalent Des for each Actor system and we decomposed it into an implementable one.

In further research, we shall mainly be engaged in three topics. At first, we intend to further explore the different classes of transition systems and their similarity relationships. The next topic is about Actor systems. In Section 3 we mentioned that we are interested in sufficient conditions for deleting an actor. We have also in mind to compare our Actor model with other formalisms of Actor systems. For example, Janssens and Rozenberg [9] have proposed a formal Actor model based on graph grammars. We would also like to investigate the modelling power of Actor systems. Are they really useful for many practical cases? Our model might help us to answer this question and the prototypes could also contribute. Lastly, we are working on a formal decomposition/composition theory for Des'ses and we are interested in comparing the Des model with other models coming from literature to describe discrete event systems.

Our experiences with Des'ses are very promising. We have described a lot of systems coming from literature and practice as Des, including Petri nets and Actor systems, and we have built a tool, called EXSPECT, to prototype Des'ses.

References

[1] **Agha G.A.:** *ACTORS: A model of concurrent computation in distributed systems.*
MIT Press, 1986.

[2] **Bos R., C. Hemerik:** *An introduction to the category-theoretic solution of recursive domain equations.*
Computing Science Notes 88/15, Eindhoven University of Technology, 1988.

[3] **Gammelgaard A.:** *Implementation Conditions for Delay Insensitive Circuits.*
In: Lecture Notes in Computer Science 365 pp. 341-355, eds. E. Odijk, M. Rem and J.-C. Syre, Springer-Verlag, 1989.

[4] **Genrich H.J.:** *Predicate/Transition Nets.*
In: Lecture Notes in Computer Science 254 pp. 207-247, eds. W. Brauer, W. Reisig and G. Rozenberg, Springer-Verlag, 1987.

[5] **Hee K.M. van, P.M.P. Rambags:** *Discrete Event Systems: Concepts and basic results.*
Computing Science Notes 88/18, Eindhoven University of Technology, 1988.

[6] **Hee K.M. van, L.J. Somers, M. Voorhoeve:** *Executable Specifications for Distributed Information Systems.*
In: Information System Concepts: An In-depth Analysis pp. 139-156, eds. E.D. Falkenberg and P. Lindgreen, North-Holland, 1989.

[7] **Hee K.M. van, L.J. Somers, M. Voorhoeve:** *EXSPECT, the functional part.*
Computing Science Notes 88/20, Eindhoven University of Technology, 1988.

[8] **Hesselink W.H.:** *Deadlock and Fairness in Morphisms of Transition Systems.*
In: Theoretical computer Science 59 pp. 235-257, North-Holland, 1988.

[9] **Janssens D., G. Rozenberg:** *Actor Grammars.*
In: Mathematical Systems Theory vol. 22 no. 2 pp. 75-107, ed. S.A. Greibach, 1989.

[10] **Jensen K.:** *Coloured Petri Nets.*
In: Lecture Notes in Computer Science 254 pp. 248-299, eds. W. Brauer, W. Reisig and G. Rozenberg, Springer-Verlag, 1987.

[11] **Kaldewaij A.:** *A Formalism for Concurrent Processes.*
Ph.D. Thesis, Eindhoven University of Technology, 1986.

[12] **Milner R.:** *A calculus of communicating systems.*
Lecture Notes in Computer Science 92, Springer, Berlin, 1980.

[13] **Peterson J.L.:** *Petri net theory and the modeling of systems.*
Prentice-Hall, 1981.

[14] **Scott D.:** *Data Types as Lattices.*
In: SIAM Journal on Computing vol. 5 no. 3 pp. 522-587, 1976.

[15] **Smyth M.B., G.D. Plotkin:** *The Category-Theoretic Solution of Recursive Domain Equations.*
In: SIAM Journal on Computing vol. 11 no. 4 pp. 761-783, 1982.

[16] **Tarski A.:** *A Lattice-Theoretical Fixpoint Theorem and its Applications.*
In: Pacific Journal of Mathematics vol. 5 pp. 285-309, 1955.

[17] **Thiagarajan P.S.:** *Elementary Net Systems.*
In: Lecture Notes in Computer Science 254 pp. 26-59, eds. W. Brauer, W. Reisig and G. Rozenberg, Springer-Verlag, 1987.

Appendix

This appendix includes proofs.

Proof Lemma 2.2.
From Eqs. 1 and 2 it follows that $\dot{S}_A \subseteq S$. We first show $\forall s \in \dot{S}_A : f(s) \in \dot{S}_B$. Let $s \in \dot{S}_A$, then $\exists n \in I\!N_0 : \exists s_0, \ldots, s_n \in S_A : s_0 \in L_A \wedge s_n = s \wedge \forall i \in \{1, \ldots, n\} : \langle s_{i-1}, s_i \rangle \in T_A$. Then $f(s_0) \in L_B$, $s_i \in S$ for all $i \in \{0, \ldots, n\}$ and using Eq. 5 we have for all $i \in \{1, \ldots, n\}$, $f(s_{i-1}) = f(s_i)$ or $\langle f(s_{i-1}), f(s_i) \rangle \in T_B$, hence $\exists m \leq n : \exists t_0, \ldots, t_m \in S_B : t_0 \in L_B \wedge t_m = f(s) \wedge \forall i \in \{1, \ldots, m\} : \langle t_{i-1}, t_i \rangle \in T_B$, i.e. $f(s) \in \dot{S}_B$.
Next, let $\langle s, s' \rangle \in \dot{T}_A$. We have already showed $f(s), f(s') \in \dot{S}_B$. What remains to prove is $f(s) = f(s') \vee \langle f(s), f(s') \rangle \in T_B$.
$\langle s, s' \rangle \in \dot{T}_A$ implies $s = s' \vee \langle s, s' \rangle \in T_A$. If $s = s'$ then $f(s) = f(s')$ else with Eq. 5 it follows that $f(s) = f(s')$ or $\langle f(s), f(s') \rangle \in T_B$.
□

Proof Lemma 2.6.
Since $S \subseteq S_A$, we can apply Lemma 2.2. So we already know that A realizes B with f. First, we prove the surjectivity of f for $\dot{S}_A \to \dot{S}_B$. Let $t \in \dot{S}_B$, then $\exists m \in I\!N_0 : \exists t_0, \ldots, t_m \in S_B : t_0 \in L_B \wedge t_m = t \wedge \forall i \in \{1, \ldots, m\} : \langle t_{i-1}, t_i \rangle \in T_B$. We show $t \in f(\dot{S}_A)$ by induction. If $m = 0$ then by Eq. 4, $\exists s \in L_A : f(s) = t$ and $L_A \subseteq \dot{S}_A$. If $m > 0$ then we may assume $\exists s_0 \in \dot{S}_A : f(s_0) = t_{m-1}$. Eqs. 1 and 2 imply $\dot{S}_A \subseteq S$, i.e. $s_0 \in S$. Applying Eq. 6 yields $\exists s_1, \ldots, s_n \in f^{-1}(t_{m-1}) : \exists s_{n+1} \in f^{-1}(t_m) : \forall i \in \{0, \ldots, n\} : \langle s_i, s_{i+1} \rangle \in T_A$, i.e. $s_{n+1} \in \dot{S}_A$ and $f(s_{n+1}) = t_m = t$.
Next, let $\langle t, t' \rangle \in \dot{T}_B$ and $s_0 \in \dot{S}_A$ such that $f(s_0) = t$. If $t = t'$ then $\langle f(s_0), f(s_0) \rangle = \langle t, t' \rangle \in \dot{T}_A$, else with Eq. 6 it follows that $\exists s_1, \ldots, s_n \in f^{-1}(t) : \exists s_{n+1} \in f^{-1}(t') : \forall i \in \{0, \ldots, n\} : \langle s_i, s_{i+1} \rangle \in T_A$. Since $s_0 \in \dot{S}_A$, also $s_i \in \dot{S}_A$, hence $\langle s_{i-1}, s_i \rangle \in \dot{T}_A$ for all $i \in \{1, \ldots, n+1\}$.
□

Proof Lemma 2.9.
Note: $f \in \dot{S}_A \to \dot{S}_B$. First we prove the surjectivity of f for $\dot{S}_A \to \dot{S}_B$. Let $t \in \dot{S}_B$, then $\exists n \in I\!N_0 : \exists t_0, \ldots, t_n \in S_B : t_0 \in L_B \wedge t_n = t \wedge \forall i \in \{1, \ldots, n\} : \langle t_{i-1}, t_i \rangle \in T_B$. We show $t \in f(\dot{S}_A)$. By Eq. 2, $t_0 \in L_B = f(L_A)$ hence $\exists s_0 \in L_A : f(s_0) = t_0$. Applying Eq. 1 n times yields $\forall i \in \{1, \ldots, n\} : \exists s_i \in S_A : t_i = f(s_i)$ and then using Eq. 3 n times leads to $\forall i \in \{1, \ldots, n\} : \langle s_{i-1}, s_i \rangle \in T_A$, i.e. $s_n \in \dot{S}_A$ and $f(s_n) = t_n = t$.
Next, let $s, s' \in \dot{S}_A$. By the bijectivity of f and Eq. 3, $\langle s, s' \rangle \in \dot{T}_A \Leftrightarrow s = s' \vee \langle s, s' \rangle \in T_A \Leftrightarrow f(s) = f(s') \vee \langle f(s), f(s') \rangle \in T_B \Leftrightarrow \langle f(s), f(s') \rangle \in \dot{T}_B$.
□

Proof Lemma 3.4.
$e \in E(s)$ means $\mathrm{dom}(e) \subseteq \mathrm{dom}(s)$ and for all $\alpha \in \mathrm{dom}(e)$, $e(\alpha) \in \mathrm{dom}(s_2(\alpha))$. Since $\mathrm{dom}(e \restriction D_1) \neq \emptyset$ and $\mathrm{dom}(e \restriction D_1) \subseteq \mathrm{dom}(e)$, we have $e \restriction D_1 \in E(s)$. Analogously $e \restriction D_2 \in E(s)$. Using the definition of μ and Lemma 3.3 we derive:

$$\begin{aligned} & \mu(s,e)(\alpha) \\ = \quad & \{ \langle t, \tilde{m} \rangle \mid \exists \alpha' \in \mathrm{dom}(e) \ : \ (t, \langle \alpha, \tilde{m} \rangle) \in \beta_1(s, e, \alpha') \} \end{aligned}$$

$$= \{\langle t, \tilde{m}\rangle \mid \exists \alpha' \in D_1 : (t, \langle \alpha, \tilde{m}\rangle) \in \beta_1(s, e, \alpha')\} \cup \{\langle t, \tilde{m}\rangle \mid \exists \alpha' \in D_2 : (t, \langle \alpha, \tilde{m}\rangle) \in \beta_1(s, e, \alpha')\}$$

$$= \{\langle t, \tilde{m}\rangle \mid \exists \alpha' \in D_1 : (t, \langle \alpha, \tilde{m}\rangle) \in \beta_1(s, e\restriction D_1, \alpha')\} \cup \{\langle t, \tilde{m}\rangle \mid \exists \alpha' \in D_2 : (t, \langle \alpha, \tilde{m}\rangle) \in \beta_1(s, e\restriction D_2, \alpha')\}$$

$$= \mu(s, e\restriction D_1)(\alpha) \cup \mu(s, e\restriction D_2)(\alpha) .$$

□

Proof Lemma 3.5.
Let $s \in S$, $e \in E(s)$, $D \subseteq \mathrm{dom}(e)$, $\alpha \in D$ and $\alpha' \in \mathrm{dom}(\beta_2(s, e, \alpha))$, then $D \neq \emptyset$. If $D = \mathrm{dom}(e)$ then the assertion holds. Assume $D \subset \mathrm{dom}(e)$, then $\mu(s, e)(\alpha') = \mu(s, e\restriction D)(\alpha') \cup \mu(s, e\restriction(\mathrm{dom}(e) \setminus D))(\alpha')$, according to Lemma 3.4.
Let $D' := \mathrm{dom}(e) \setminus D$. It satisfies to show $\mu(s, e\restriction D')(\alpha') = \emptyset$.
$\mu(s, e\restriction D')(\alpha') = \{\langle t, \tilde{m}\rangle \mid \exists \alpha'' \in D' : (t, \langle \alpha', \tilde{m}\rangle) \in \beta_1(s, e\restriction D', \alpha'')\}$. Let $(t, \langle \alpha', \tilde{m}\rangle) \in \beta_1(s, e\restriction D', \alpha'')$ for some $\alpha'' \in D'$ to get a contradiction. The fourth constraint on B (see Definition 3.1) requires α' be either an acquaintance, i.e. $\alpha' \in \mathrm{dom}(s)$, or a newly created actor, i.e. $\exists i \in I : \alpha' = e(\alpha'').i$. We have $\alpha' \notin \mathrm{dom}(s)$ because $\alpha' \in \mathrm{dom}(\beta_2(s, e, \alpha))$. Moreover, $\alpha' \in \mathrm{dom}(\beta_2(s, e, \alpha))$ implies $\exists i' \in I : \alpha' = e(\alpha).i'$. Consequently, $\alpha' = e(\alpha'').i = e(\alpha).i'$. Hence, $e(\alpha'') = e(\alpha)$ and all names in configuration s are different, so $\alpha'' = \alpha$. Contradiction.
□

Proof Theorem 3.2.
By Lemma 3.4, $e\restriction D_1 \in E(s)$ and $e\restriction D_2 \in E(s)$. Let $s' := Q(s, e\restriction D_1)$, then by the definition of Q and Lemma 3.3,

$$\begin{aligned} s' = {} & \lambda\, \alpha \in D_1 : \langle \beta_3(s, e\restriction D_1, \alpha), \\ & \qquad s_2(\alpha) \setminus \{(e\restriction D_1(\alpha), s_2(\alpha)(e\restriction D_1(\alpha)))\} \cup \mu(s, e\restriction D_1)(\alpha)\rangle \\ & \cup \lambda\, \alpha \in \mathrm{dom}(s) \setminus D_1 : \langle s_1(\alpha), s_2(\alpha) \cup \mu(s, e\restriction D_1)(\alpha)\rangle \\ & \cup \bigcup \{\lambda\, \alpha' \in \mathrm{dom}(\beta_2(s, e\restriction D_1, \alpha)) : \langle \beta_2(s, e\restriction D_1, \alpha)(\alpha'), \mu(s, e\restriction D_1)(\alpha')\rangle \\ & \qquad \mid \alpha \in D_1\} \\ = {} & \lambda\, \alpha \in D_1 : \langle \beta_3(s, e, \alpha), s_2(\alpha) \setminus \{(e(\alpha), s_2(\alpha)(e(\alpha)))\} \cup \mu(s, e\restriction D_1)(\alpha)\rangle \\ & \cup \lambda\, \alpha \in \mathrm{dom}(s) \setminus D_1 : \langle s_1(\alpha), s_2(\alpha) \cup \mu(s, e\restriction D_1)(\alpha)\rangle \\ & \cup \bigcup \{\lambda\, \alpha' \in \mathrm{dom}(\beta_2(s, e, \alpha)) : \langle \beta_2(s, e, \alpha)(\alpha'), \mu(s, e\restriction D_1)(\alpha')\rangle \\ & \qquad \mid \alpha \in D_1\} \end{aligned}$$

$D_2 \subseteq \mathrm{dom}(s) \setminus D_1$, hence $\forall \alpha \in D_2 : s_1(\alpha) = s'_1(\alpha)$ and $s_2(\alpha) \subseteq s'_2(\alpha)$. Since $\mathrm{dom}(s) \subseteq \mathrm{dom}(s')$ and $e\restriction D_2 \in E(s)$, the conditions of Lemma 3.2 are satisfied. Consequently, $e\restriction D_2 \in E(s')$. Then,

$$Q(s', e\restriction D_2)$$

$$= \quad \{\!\!\{ \text{ Def. } Q \}\!\!\}$$

$$\begin{aligned}
&\lambda\,\alpha \in D_2 \;:\; \langle \beta_3(s', e\restriction D_2, \alpha),\\
&\qquad\qquad s_2'(\alpha) \setminus \{(e\restriction D_2(\alpha),\, s_2'(\alpha)(e\restriction D_2(\alpha)))\} \cup \mu(s', e\restriction D_2)(\alpha)\rangle\\
&\cup\; \lambda\,\alpha \in \mathrm{dom}(s') \setminus D_2 \;:\; \langle s_1'(\alpha),\, s_2'(\alpha) \cup \mu(s', e\restriction D_2)(\alpha)\rangle\\
&\cup\; \bigcup \{\lambda\,\alpha' \in \mathrm{dom}(\beta_2(s', e\restriction D_2, \alpha)) \;:\; \langle \beta_2(s', e\restriction D_2, \alpha)(\alpha'),\, \mu(s', e\restriction D_2)(\alpha')\rangle\\
&\qquad \mid \alpha \in D_2\}
\end{aligned}$$

$=$ $\{\!\!\{$ Substitution, Lemma 3.2, Lemma 3.3 $\}\!\!\}$

$$\begin{aligned}
&\lambda\,\alpha \in D_2 \;:\; \langle \beta_3(s, e, \alpha),\\
&\qquad\qquad s_2(\alpha) \cup \mu(s, e\restriction D_1)(\alpha) \setminus \{(e(\alpha),\, s_2'(\alpha)(e(\alpha)))\} \cup \mu(s, e\restriction D_2)(\alpha)\rangle\\
&\cup\; \lambda\,\alpha \in D_1 \;:\; \langle \beta_3(s, e, \alpha),\\
&\qquad\qquad\qquad s_2(\alpha) \setminus \{(e(\alpha),\, s_2(\alpha)(e(\alpha)))\} \cup \mu(s, e\restriction D_1)(\alpha) \cup \mu(s, e\restriction D_2)(\alpha)\rangle\\
&\cup\; \lambda\,\alpha \in \mathrm{dom}(s) \setminus (D_1 \cup D_2) \;:\; \langle s_1(\alpha),\, s_2(\alpha) \cup \mu(s, e\restriction D_1)(\alpha) \cup \mu(s, e\restriction D_2)(\alpha)\rangle\\
&\cup\; \bigcup \{\lambda\,\alpha' \in \mathrm{dom}(\beta_2(s, e, \alpha)) \;:\; \langle \beta_2(s, e, \alpha)(\alpha'),\\
&\qquad\qquad\qquad\qquad \mu(s, e\restriction D_1)(\alpha') \cup \mu(s, e\restriction D_2)(\alpha')\rangle\\
&\qquad \mid \alpha \in D_1\}\\
&\cup\; \bigcup \{\lambda\,\alpha' \in \mathrm{dom}(\beta_2(s, e, \alpha)) \;:\; \langle \beta_2(s, e, \alpha)(\alpha'),\, \mu(s, e\restriction D_2)(\alpha')\rangle \mid \alpha \in D_2\}
\end{aligned}$$

$=$ $\{\!\!\{$ For $\alpha \in D_2$ we have $s_2'(\alpha)(e(\alpha)) = s_2(\alpha)(e(\alpha))$, Lemma 3.4, Lemma 3.5 $\}\!\!\}$

$$\begin{aligned}
&\lambda\,\alpha \in D_1 \cup D_2 \;:\; \langle \beta_3(s, e, \alpha),\, s_2(\alpha) \setminus \{(e(\alpha),\, s_2(\alpha)(e(\alpha)))\} \cup \mu(s, e)(\alpha)\rangle\\
&\cup\; \lambda\,\alpha \in \mathrm{dom}(s) \setminus (D_1 \cup D_2) \;:\; \langle s_1(\alpha),\, s_2(\alpha) \cup \mu(s, e)(\alpha)\rangle\\
&\cup\; \bigcup \{\lambda\,\alpha' \in \mathrm{dom}(\beta_2(s, e, \alpha)) \;:\; \langle \beta_2(s, e, \alpha)(\alpha'),\, \mu(s, e)(\alpha')\rangle \mid \alpha \in D_1 \cup D_2\}
\end{aligned}$$

$=$ $\{\!\!\{$ $\mathrm{dom}(e) = D_1 \cup D_2$, def. Q $\}\!\!\}$

$$Q(s, e)\ .$$

□

Proof Theorem 5.1.
We use Lemma 2.2. What remains to prove is $\forall s, s' \in AS \;:\; \langle s, s'\rangle \in TT \;\Rightarrow\; g(s) = g(s') \;\vee\; \langle g(s), g(s')\rangle \in T$.
Let $s, s' = \{(com, b), (act, \{f\})\}, \{(com, b'), (act, \{f'\})\} \in AS$ and assume $\langle s, s'\rangle \in TT$. Then a communication $\langle t, \langle \alpha, \tilde{m}\rangle\rangle \in b$ exists such that $b' = b \setminus \{\langle t, \langle \alpha, \tilde{m}\rangle\rangle\} \cup B_1(f(\alpha), t, \tilde{m})$ and $f' = f[\alpha \;:\; B_3(f(\alpha), t, \tilde{m})] \cup B_2(f(\alpha), t, \tilde{m})$. Moreover, $\{(\alpha, t)\} \in E(g(s))$ and $\beta(g(s), \{(\alpha, t)\}, \alpha) = B(f(\alpha), t, \tilde{m})$ and $\mu(g(s), \{(\alpha, t)\}) = \lambda\,\alpha' \in \mathcal{A} \;:\; \{\langle t', \tilde{m}'\rangle \mid (t', \langle \alpha', \tilde{m}'\rangle) \in B_1(f(\alpha), t, \tilde{m})\}$. By def. Q (the transition function of the Actor system),

$$Q(g(s), \{(\alpha, t)\})$$

$$
\begin{aligned}
= \quad & \{(\alpha,\ \langle B_3(f(\alpha),t,\tilde{m}),\ \{(t',\tilde{m}') \mid \langle t',\langle\alpha,\tilde{m}'\rangle\rangle \in b\} \setminus \{(t,\tilde{m})\} \cup \\
& \qquad\qquad \{(t',\tilde{m}') \mid (t',\langle\alpha,\tilde{m}'\rangle) \in B_1(f(\alpha),t,\tilde{m})\}\rangle)\} \\
& \cup\ \lambda\ \alpha' \in \mathrm{dom}(f) \setminus \{\alpha\}\ : \\
& \qquad \langle f(\alpha'),\ \{(t',\tilde{m}') \mid \langle t',\langle\alpha',\tilde{m}'\rangle\rangle \in b\} \cup \\
& \qquad\qquad \{(t',\tilde{m}') \mid (t',\langle\alpha',\tilde{m}'\rangle) \in B_1(f(\alpha),t,\tilde{m})\}\rangle \\
& \cup\ \lambda\ \alpha' \in \mathrm{dom}(B_2(f(\alpha),t,\tilde{m}))\ : \\
& \qquad \langle B_2(f(\alpha),t,\tilde{m})(\alpha'),\ \{(t',\tilde{m}') \mid (t',\langle\alpha',\tilde{m}'\rangle) \in B_1(f(\alpha),t,\tilde{m})\}\rangle
\end{aligned}
$$

$$
= \quad \lambda\ \alpha' \in \mathrm{dom}(f')\ : \\
\qquad \langle f'(\alpha'),\ \{(t',\tilde{m}') \mid \langle t',\langle\alpha',\tilde{m}'\rangle\rangle \in b \setminus \{\langle t,\langle\alpha,\tilde{m}\rangle\rangle\} \cup B_1(f(\alpha),t,\tilde{m})\}\rangle
$$

$$
= \quad g(\{(com,\ b'),\ (act,\ \{f'\})\})
$$

$$
= \quad g(s')\ .
$$

Hence, $\langle g(s), g(s')\rangle \in T$.
□

Proof Lemma 5.3.
First we give function h. Let $s \in S$, then

$$
h\ :=\ \lambda\ e \in E(s)\ :\ \lambda\ \alpha \in \mathrm{dom}(e)\ : \\
\qquad \{\langle b_\alpha, s_1(\alpha)\rangle,\ \langle c_\alpha, \langle e(\alpha), s_2(\alpha)(e(\alpha))\rangle\rangle\}\ .
$$

Clearly, h is injective. We prove $h(E(s)) = F'(g(s))$.

$$
\begin{aligned}
h(E(s)) \ &= \ h(\{e \in \mathcal{A} \nrightarrow \mathcal{T} \mid \mathrm{dom}(e) \neq \emptyset \wedge \mathrm{dom}(e) \subseteq \mathrm{dom}(s) \wedge \\
&\qquad\qquad \mathrm{dom}(e) \text{ is finite} \wedge \\
&\qquad\qquad \forall\alpha \in \mathrm{dom}(e)\ :\ e(\alpha) \in \mathrm{dom}(s_2(\alpha))\}) \\
&= \ \{e' \in \mathcal{A} \nrightarrow S' \mid \mathrm{dom}(e') \neq \emptyset \wedge \mathrm{dom}(e') \subseteq \mathrm{dom}(s) \wedge \\
&\qquad\qquad \mathrm{dom}(e') \text{ is finite} \wedge \\
&\qquad\qquad \forall\alpha \in \mathrm{dom}(e')\ :\ e'(\alpha) \subseteq g(s) \wedge \\
&\qquad\qquad\qquad\qquad e'(\alpha) \in \mathrm{dom}(R'(\alpha))\} \\
&= \ \{\!\!\{\ \text{no two different processors share an input channel}\ \}\!\!\} \\
&\qquad \{e' \in \mathcal{A} \nrightarrow S' \mid \mathrm{dom}(e') \neq \emptyset \wedge \mathrm{mrng}(e') \subseteq g(s) \wedge \\
&\qquad\qquad \mathrm{dom}(e') \text{ is finite} \wedge \\
&\qquad\qquad \forall\alpha \in \mathrm{dom}(e')\ :\ e'(\alpha) \in \mathrm{dom}(R'(\alpha))\} \\
&= \ F'(g(s))\ .
\end{aligned}
$$

□

Proof Theorem 5.2.
According to Lemma 2.9 we only need to prove:

1. $\forall \langle t, t' \rangle \in T' \ : \ t \in \mathrm{rng}(g) \Rightarrow t' \in \mathrm{rng}(g)$;
2. $\forall s, s' \in S \ : \ \langle s, s' \rangle \in T \Leftrightarrow \langle g(s), g(s') \rangle \in T'$.

We start with 2. Let $s, s' \in S$.

$\langle s, s' \rangle \in T$

$\Leftrightarrow$ $\quad \{\!\!\{ \text{ def. } T \}\!\!\}$

$\exists e \in E(s) \ : \ s' = Q(s, e)$

$\Leftrightarrow$ $\quad \{\!\!\{ \text{ def. } Q \}\!\!\}$

$\exists e \in E(s) \ :$
$\quad s' = \lambda\, \alpha \in \mathrm{dom}(e) \ : \ \langle \beta_3(s,e,\alpha),$
$\qquad\qquad s_2(\alpha) \setminus \{(e(\alpha), s_2(\alpha)(e(\alpha)))\} \cup \mu(s,e)(\alpha) \rangle$
$\qquad \cup \ \lambda\, \alpha \in \mathrm{dom}(s) \setminus \mathrm{dom}(e) \ : \ \langle s_1(\alpha), s_2(\alpha) \cup \mu(s,e)(\alpha) \rangle$
$\qquad \cup \ \bigcup \{ \lambda\, \alpha' \in \mathrm{dom}(\beta_2(s,e,\alpha)) \ : \ \langle \beta_2(s,e,\alpha)(\alpha'), \mu(s,e)(\alpha') \rangle$
$\qquad\quad \mid \alpha \in \mathrm{dom}(e) \}$

$\Leftrightarrow$ $\quad \{\!\!\{ \text{ def. } g, \, g \text{ is injective (Lemma 5.2) } \}\!\!\}$

$\exists e \in E(s) \ :$
$\quad g(s') = \{ \langle c_\alpha, \langle t, \tilde{m} \rangle \rangle \mid \alpha \in \mathcal{A} \wedge (t, \tilde{m}) \in \mu(s,e)(\alpha) \}$
$\qquad \cup \{ \langle b_\alpha, \beta_3(s,e,\alpha) \rangle \mid \alpha \in \mathrm{dom}(e) \}$
$\qquad \cup \{ \langle c_\alpha, \langle t, \tilde{m} \rangle \rangle \mid \alpha \in \mathrm{dom}(e) \wedge (t, \tilde{m}) \in s_2(\alpha) \}$
$\qquad \setminus \{ \langle c_\alpha, \langle e(\alpha), s_2(\alpha)(e(\alpha)) \rangle \rangle \mid \alpha \in \mathrm{dom}(e) \}$
$\qquad \cup \{ \langle b_\alpha, s_1(\alpha) \rangle \mid \alpha \in \mathrm{dom}(s) \setminus \mathrm{dom}(e) \}$
$\qquad \cup \{ \langle c_\alpha, \langle t, \tilde{m} \rangle \rangle \mid \alpha \in \mathrm{dom}(s) \setminus \mathrm{dom}(e) \wedge (t, \tilde{m}) \in s_2(\alpha) \}$
$\qquad \cup \{ \langle b_{\alpha'}, \beta_2(s,e,\alpha)(\alpha') \rangle \mid \alpha \in \mathrm{dom}(e) \wedge \alpha' \in \mathrm{dom}(\beta_2(s,e,\alpha)) \}$

$\Leftrightarrow$ $\quad \{\!\!\{ \text{ rearrange } \}\!\!\}$

$\exists e \in E(s) \ :$
$\quad g(s') = \{ \langle b_\alpha, s_1(\alpha) \rangle \mid \alpha \in \mathrm{dom}(s) \}$
$\qquad \cup \{ \langle c_\alpha, \langle t, \tilde{m} \rangle \rangle \mid \alpha \in \mathrm{dom}(s) \wedge (t, \tilde{m}) \in s_2(\alpha) \}$
$\qquad \setminus \{ \langle b_\alpha, s_1(\alpha) \rangle \mid \alpha \in \mathrm{dom}(e) \}$
$\qquad \setminus \{ \langle c_\alpha, \langle e(\alpha), s_2(\alpha)(e(\alpha)) \rangle \rangle \mid \alpha \in \mathrm{dom}(e) \}$
$\qquad \cup \{ \langle c_\alpha, \langle t, \tilde{m} \rangle \rangle \mid \alpha \in \mathcal{A} \wedge (t, \tilde{m}) \in \mu(s,e)(\alpha) \}$
$\qquad \cup \{ \langle b_{\alpha'}, \beta_2(s,e,\alpha)(\alpha') \rangle \mid \alpha \in \mathrm{dom}(e) \wedge \alpha' \in \mathrm{dom}(\beta_2(s,e,\alpha)) \}$
$\qquad \cup \{ \langle b_\alpha, \beta_3(s,e,\alpha) \rangle \mid \alpha \in \mathrm{dom}(e) \}$

$\Leftrightarrow$ $\quad \{\!\!\{$ def. g, def. h, def. μ, def. β $\}\!\!\}$

$$\begin{aligned}
&\exists e \in E(s) \ : \\
&\quad g(s') \ = \ g(s) \ \setminus \ \bigcup \{h(e)(\alpha) \mid \alpha \in \mathrm{dom}(e)\} \\
&\qquad\qquad \cup \{\langle c_\alpha, \langle t, \tilde{m}\rangle\rangle \\
&\qquad\qquad\quad \mid \alpha \in \mathcal{A} \wedge \exists \alpha' \in \mathrm{dom}(e) \ : \\
&\qquad\qquad\qquad (t, \langle \alpha, \tilde{m}\rangle) \in B_1(s_1(\alpha'), e(\alpha'), s_2(\alpha')(e(\alpha')))\} \\
&\qquad\qquad \cup \{\langle b_{\alpha'}, \tilde{a}\rangle \mid \exists \alpha \in \mathrm{dom}(e) \ : \\
&\qquad\qquad\qquad\qquad (\alpha', \tilde{a}) \in B_2(s_1(\alpha), e(\alpha), s_2(\alpha)(e(\alpha)))\} \\
&\qquad\qquad \cup \{\langle b_\alpha, B_3(s_1(\alpha), e(\alpha), s_2(\alpha)(e(\alpha)))\rangle \mid \alpha \in \mathrm{dom}(e)\}
\end{aligned}$$

$\Leftrightarrow$ $\quad \{\!\!\{$ def. h, def. R' $\}\!\!\}$

$$\exists e \in E(s) \ : \ g(s') \ = \ g(s) \setminus \mathrm{mrng}(h(e)) \cup \bigcup_{\alpha \in \mathrm{dom}(h(e))} R'(\alpha)(h(e)(\alpha))$$

$\Leftrightarrow$ $\quad \{\!\!\{$ Lemma 5.3, $e' = h(e)$ $\}\!\!\}$

$$\exists e' \in F'(g(s)) \ : \ g(s') \ = \ g(s) \setminus \mathrm{mrng}(e') \cup \bigcup_{\alpha \in \mathrm{dom}(e')} R'(\alpha)(e'(\alpha))$$

$\Leftrightarrow$ $\quad \{\!\!\{$ def. T' $\}\!\!\}$

$$\langle g(s), g(s')\rangle \in T' \ .$$

Next, we turn to 1.
Let $\langle t, t'\rangle \in T'$, then $\exists e' \in F'(t) \ : \ t' = t \setminus \mathrm{mrng}(e') \cup \bigcup_{\alpha \in \mathrm{dom}(e')} R'(\alpha)(e'(\alpha))$. Suppose $t \in \mathrm{rng}(g)$. The injectivity of g and the bijectivity of h allow us to define $s' := Q(g^{-1}(t), h^{-1}(e'))$. Hence, $s' \in S$ and because of the totalness of g, $s' \in \mathrm{dom}(g)$. Similar to the above it follows that $g(s') = t'$, so $t' \in \mathrm{rng}(g)$.
□

Proof Theorem 5.3.
We define $V' := \{s \in S' \mid \sum_{q \in Q} s(\langle \mathit{collect}, q\rangle) \neq \omega\}$, i.e. V' consists of the states in S' with finitely many triggers in channel *collect*. Obviously, $L' \subseteq V'$. We also define a function $f \in S' \to S$:

$$f \ := \ \lambda\, s \in S' \ : \ (s \restriction Q) \ \cup \ \{q \in Q \mid \langle \mathit{collect}, q\rangle \in s\} \ .$$

Clearly, f is total and surjective, $f(L') = L$ and for all $s, s' \in S' \ : \ f(s \cup s') = f(s) \cup f(s')$ and if $s' \subseteq s$ then $f(s \setminus s') = f(s) \setminus f(s')$. We claim: D' simulates D with f. Using Lemma 2.6 we have to prove:

$$\forall s \in V' \ : \ \forall s' \in S' \ : \ \langle s, s'\rangle \in T' \ \Rightarrow \ s' \in V' \tag{1}$$

$$\forall s, s' \in V' \ : \ \langle s, s'\rangle \in T' \ \Rightarrow \ f(s) = f(s') \ \vee \ \langle f(s), f(s')\rangle \in T \tag{2}$$

$$\begin{aligned}
\forall \langle t, t'\rangle \in T \ : \ &\forall s_0 \in f^{-1}(t) \cap V' \ : \ \exists s_1, \ldots, s_n \in f^{-1}(t) \ : \ \exists s_{n+1} \in f^{-1}(t') \ : \\
&\forall i \in \{0, \ldots, n\} \ : \ \langle s_i, s_{i+1}\rangle \in T'
\end{aligned} \tag{3}$$

1. Let $s \in V'$, then *collect* has finitely many triggers. Let $s' \in S'$ and suppose $\langle s, s' \rangle \in T'$. As only finitely many processors can perform an action and each processor produces finitely many triggers, s' can have only finitely many additional triggers in channel *collect*. Hence, $s' \in V'$.

2. Let $\langle s, s' \rangle \in T'$, then $\exists e \in F'(s) \ : \ s' = s \setminus \mathrm{mrng}(e) \cup \bigcup_{p \in \mathrm{dom}(e)} R'(p)(e(p))$. Note: For $p \in \mathrm{dom}(e) \setminus \{\mathit{distributor}\}$, $f(e(p)) = e(p)$ and $f(R'(p)(e(p))) = R(p)(e(p))$ and if $\mathit{distributor} \in \mathrm{dom}(e)$, then $f(R'(\mathit{distributor})(e(\mathit{distributor})) = f(e(\mathit{distributor}))$. We distinguish two cases. If $\mathrm{dom}(e) \cap P = \emptyset$ then $\mathrm{dom}(e) = \{\mathit{distributor}\}$, hence $f(s) = f(s')$. Case $\mathrm{dom}(e) \cap P \neq \emptyset$, we define $e' := e \restriction P$. Then $\mathrm{dom}(e') \neq \emptyset$ and $f(s') = f(s) \setminus f(\mathrm{mrng}(e)) \cup \bigcup_{p \in \mathrm{dom}(e)} f(R'(p)(e(p))) = f(s) \setminus \mathrm{mrng}(e') \cup \bigcup_{p \in \mathrm{dom}(e')} R(p)(e'(p))$, i.e. $\langle f(s), f(s') \rangle \in T$.

3. Let $\langle t, t' \rangle \in T$ and let $s_0 \in f^{-1}(t) \cap V'$. We define $b := s_0 \restriction (Q' \setminus Q)$, then $\#b \neq \omega$, so $\exists n \in I\!N_0 \ : \ n = \#b$.
Let $q_1, \ldots, q_n \in Q$ such that $b = \bigcup_{i \in \{1, \ldots, n\}} \{\langle \mathit{collect}, q_i \rangle\}$. We define for $i \in \{1, \ldots n\}$, $s_i := (s_0 \restriction Q) \cup \{q_j \mid 1 \leq j \leq i\} \cup \{\langle \mathit{collect}, q_j \rangle \mid i < j \leq n\}$. Then $f(s_i) = t$ and because of def. $R'(\mathit{distributor})$, $\langle s_{i-1}, s_i \rangle \in T'$. Furthermore, $s_n = t$. $\langle t, t' \rangle \in T$ implies $\exists e \in F(t) \ : \ t' = t \setminus \mathrm{mrng}(e) \cup \bigcup_{p \in \mathrm{dom}(e)} R(p)(e(p))$. Notice that $e \in F'(s)$. We define $s_{n+1} := t \setminus \mathrm{mrng}(e) \cup \bigcup_{p \in \mathrm{dom}(e)} R'(p)(e(p))$, then $\langle s_n, s_{n+1} \rangle \in T'$ and $f(s_{n+1}) = t'$.

□

Proof Theorem 5.4.
Please note: $Q \subset Q'$. We define $V' := \{s \in S' \mid s \restriction (Q' \setminus Q) \text{ is finite}\}$, i.e. V' consists of the states in S' with finitely many triggers on the way to their destination. Obviously, $L' \subseteq V'$. We also define a function $g : Q' \to Q$ that maps triggers of D' to triggers of D. Triggers in $\{h_1, h_2, \ldots\}$ are mapped onto their destination channel. For all $q \in Q$:

$$g(q) \ := \ q$$

and for all $i \in I\!N_1$, $n \in I\!N_i$ and $y \in Y$:

$$g(\langle h_i, \langle n, y \rangle \rangle) \ := \ \langle c_n, y \rangle \ .$$

Next, we define a function $f \in S' \to S$:

$$f \ := \ \lambda \, s \in S' \ : \ \{g(q) \mid q \in s\} \ .$$

Clearly, f is total and surjective, $f(L') = L$ and for all $s, s' \in S' \ : \ f(s \cup s') = f(s) \cup f(s')$ and if $s' \subseteq s$ then $f(s \setminus s') = f(s) \setminus f(s')$. We use Lemma 2.6 again to prove the simulation property. We have to prove the same three items as in step one:

$$\forall s \in V' \ : \ \forall s' \in S' \ : \ \langle s, s' \rangle \in T' \ \Rightarrow \ s' \in V' \tag{1}$$

$$\forall s, s' \in V' \ : \ \langle s, s' \rangle \in T' \ \Rightarrow \ f(s) = f(s') \ \vee \ \langle f(s), f(s') \rangle \in T \tag{2}$$

$$\forall \langle t, t' \rangle \in T \ : \ \forall s_0 \in f^{-1}(t) \cap V' \ : \ \exists s_1, \ldots, s_n \in f^{-1}(t) \ : \ \exists s_{n+1} \in f^{-1}(t') \ : \\ \forall i \in \{0, \ldots, n\} \ : \ \langle s_i, s_{i+1} \rangle \in T' \qquad (3)$$

1. The same argument can be used here, but now for channels $h_1, h_2, \ldots$ instead of channel *collect*.

2. Let $\langle s, s' \rangle \in T'$, then $\exists e \in F'(s) \ : \ s' = s \setminus \mathrm{mrng}(e) \cup \bigcup_{p \in \mathrm{dom}(e)} R'(p)(e(p))$. $\mathrm{Dom}(e)$ can have three kinds of processors:
   - From $\{p_1, p_2, \ldots\}$;
   - p_0;
   - From the 'UNKNOWN PART'.

   For $p \in \{p_1, p_2, \ldots\}$, $f(e(p)) = f(R'(p)(e(p))$. If $p_0 \in \mathrm{dom}(e)$, then for some $n \in I\!N_0$ and $y \in Y$, $e(p_0) = \{\langle h_0, \langle n, y \rangle\rangle\}$. $f(e(p_0)) = e(p_0)$ and $f(R'(p_0)(e(p_0))) = \{\langle c_n, y \rangle\} = R(p_0)(e(p_0))$. For $p \in$ 'UNKNOWN PART', $f(e(p)) = e(p)$ and $f(R'(p)(e(p))) = R(p)(e(p))$.
   Let $X := \mathrm{dom}(e) \setminus \{p_1, p_2, \ldots\}$. If $X = \emptyset$ then $f(\mathrm{mrng}(e)) = f(\bigcup_{p \in \mathrm{dom}(e)} R'(p)(e(p)))$, hence $f(s) = f(s')$. Case $X \neq \emptyset$ then $f(s') = f(s) \setminus (\bigcup_{p \in X} e(p)) \cup \bigcup_{p \in X} R(p)(e(p))$, i.e. $\langle f(s), f(s') \rangle \in T$.

3. Let $\langle t, t' \rangle \in T$ and $s_0 \in f^{-1}(t) \cap V'$, then $\exists e \in F(t) \ : \ t' = t \setminus \mathrm{mrng}(e) \cup \bigcup_{p \in \mathrm{dom}(e)} R(p)(e(p))$ and s_0 has only finitely many triggers in $h_1, h_2, \ldots$. These triggers can be forwarded to their destination in $n := \sum_{\langle h_i, \langle m, y \rangle\rangle \in s_0 \restriction (Q' \setminus Q)} (m - i + 1)$ single-processor events. Thus we have a sequence $s_1, \ldots, s_n \in f^{-1}(t)$ such that $\langle s_{i-1}, s_i \rangle \in T'$ for all $i \in \{1, \ldots n\}$ and $s_n \restriction (Q' \setminus Q) = \emptyset$, i.e. $s_n = t$. Consequently, $e \in F'(s_n)$. If $p_0 \in \mathrm{dom}(e)$ then $e(p_0) = \{\langle h_0, \langle m, y \rangle\rangle\}$ for some $m \in I\!N_0, y \in Y$ and $f(R'(p_0)(e(p_0))) = \{\langle c_m, y \rangle\} = R(p_0)(e(p_0))$. For $p \in \mathrm{dom}(e) \setminus \{p_0\}$, i.e. $p \in$ 'UNKNOWN PART', we already have $f(R'(p)(e(p))) = R(p)(e(p))$. We define $s_{n+1} := s_n \setminus \mathrm{mrng}(e) \cup \bigcup_{p \in \mathrm{dom}(e)} R'(p)(e(p))$, then $\langle s_n, s_{n+1} \rangle \in T'$ and $f(s_{n+1}) = f(s_n) \setminus f(\mathrm{mrng}(e)) \cup \bigcup_{p \in \mathrm{dom}(e)} f(R'(p)(e(p))) = t \setminus \mathrm{mrng}(e) \cup \bigcup_{p \in \mathrm{dom}(e)} R(p)(e(p)) = t'$.

□

In this series appeared :

No.	Author(s)	Title
85/01	R.H. Mak	The formal specification and derivation of CMOS-circuits
85/02	W.M.C.J. van Overveld	On arithmetic operations with M-out-of-N-codes
85/03	W.J.M. Lemmens	Use of a computer for evaluation of flow films
85/04	T. Verhoeff H.M.L.J.Schols	Delay insensitive directed trace structures satisfy the foam rubber wrapper postulate
86/01	R. Koymans	Specifying message passing and real-time systems
86/02	G.A. Bussing K.M. van Hee M. Voorhoeve	ELISA, A language for formal specifications of information systems
86/03	Rob Hoogerwoord	Some reflections on the implementation of trace structures
86/04	G.J. Houben J. Paredaens K.M. van Hee	The partition of an information system in several parallel systems
86/05	Jan L.G. Dietz Kees M. van Hee	A framework for the conceptual modeling of discrete dynamic systems
86/06	Tom Verhoeff	Nondeterminism and divergence created by concealment in CSP
86/07	R. Gerth L. Shira	On proving communication closedness of distributed layers
86/08	R. Koymans R.K. Shyamasundar W.P. de Roever R. Gerth S. Arun Kumar	Compositional semantics for real-time distributed computing (Inf.&Control 1987)
86/09	C. Huizing R. Gerth W.P. de Roever	Full abstraction of a real-time denotational semantics for an OCCAM-like language
86/10	J. Hooman	A compositional proof theory for real-time distributed message passing
86/11	W.P. de Roever	Questions to Robin Milner - A responder's commentary (IFIP86)
86/12	A. Boucher R. Gerth	A timed failures model for extended communicating processes

86/13	R. Gerth W.P. de Roever	Proving monitors revisited: a first step towards verifying object oriented systems (Fund. Informatica IX-4)
86/14	R. Koymans	Specifying passing systems requires extending temporal logic
87/01	R. Gerth	On the existence of sound and complete axiomatizations of the monitor concept
87/02	Simon J. Klaver Chris F.M. Verberne	Federatieve Databases
87/03	G.J. Houben J.Paredaens	A formal approach to distributed information systems
87/04	T.Verhoeff	Delay-insensitive codes - An overview
87/05	R.Kuiper	Enforcing non-determinism via linear time temporal logic specification.
87/06	R.Koymans	Temporele logica specificatie van message passing en real-time systemen (in Dutch)
87/07	R.Koymans	Specifying message passing and real-time systems with real-time temporal logic
87/08	H.M.J.L. Schols	The maximum number of states after projection
87/09	J. Kalisvaart L.R.A. Kessener W.J.M. Lemmens M.L.P. van Lierop F.J. Peters H.M.M. van de Wetering	Language extensions to study structures for raster graphics
87/10	T.Verhoeff	Three families of maximally nondeterministic automata
87/11	P.Lemmens	Eldorado ins and outs Specifications of a data base management toolkit according to the functional model
87/12	K.M. van Hee and A.Lapinski	OR and AI approaches to decision support systems
87/13	J.C.S.P. van der Woude	Playing with patterns searching for strings
87/14	J. Hooman	A compositional proof system for an occam-like real-time language
87/15	C. Huizing R. Gerth	A compositional semantics for statecharts

	W.P. de Roever	
87/16	H.M.M. ten Eikelder J.C.F. Wilmont	Normal forms for a class of formulas
87/17	K.M. van Hee G.-J.Houben J.L.G. Dietz	Modelling of discrete dynamic systems framework and examples
87/18	C.W.A.M. van Overveld	An integer algorithm for rendering curved surfaces
87/19	A.J.Seebregts	Optimalisering van file allocatie in gedistribueerde database systemen
87/20	G.J. Houben J. Paredaens	The R^2 -Algebra: An extension of an algebra for nested relations
87/21	R. Gerth M. Codish Y. Lichtenstein E. Shapiro	Fully abstract denotational semantics for concurrent PROLOG
88/01	T. Verhoeff	A Parallel Program That Generates the Möbius Sequence
88/02	K.M. van Hee G.J. Houben L.J. Somers M. Voorhoeve	Executable Specification for Information Systems
88/03	T. Verhoeff	Settling a Question about Pythagorean Triples
88/04	G.J. Houben J.Paredaens D.Tahon	The Nested Relational Algebra: A Tool to Handle Structured Information
88/05	K.M. van Hee G.J. Houben L.J. Somers M. Voorhoeve	Executable Specifications for Information Systems
88/06	H.M.J.L. Schols	Notes on Delay-Insensitive Communication
88/07	C. Huizing R. Gerth W.P. de Roever	Modelling Statecharts behaviour in a fully abstract way
88/08	K.M. van Hee G.J. Houben L.J. Somers M. Voorhoeve	A Formal model for System Specification
88/09	A.T.M. Aerts K.M. van Hee	A Tutorial for Data Modelling
88/10	J.C. Ebergen	A Formal Approach to Designing Delay Insensitive Circuits
88/11	G.J. Houben J.Paredaens	A graphical interface formalism: specifying nested relational databases

88/12	A.E. Eiben	Abstract theory of planning
88/13	A. Bijlsma	A unified approach to sequences, bags, and trees
88/14	H.M.M. ten Eikelder R.H. Mak	Language theory of a lambda-calculus with recursive types
88/15	R. Bos C. Hemerik	An introduction to the category theoretic solution of recursive domain equations
88/16	C.Hemerik J.P.Katoen	Bottom-up tree acceptors
88/17	K.M. van Hee G.J. Houben L.J. Somers M. Voorhoeve	Executable specifications for discrete event systems
88/18	K.M. van Hee P.M.P. Rambags	Discrete event systems: concepts and basic results
88/19	D.K. Hammer K.M. van Hee	Fasering en documentatie in software engineering.
88/20	K.M. van Hee L. Somers M.Voorhoeve	EXSPECT, the functional part
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 paralell 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