CCS and time : a practical and comprehensible approach to a performance evaluation of finite state CCS descriptions

Research Report
ISSN 0167-9708
Coden TEUEDE

Eindhoven
University of Technology
Netherlands

Faculty of Electrical Engineering

CCS and Time: A Practical and Comprehensible Approach to a Performance Evaluation of Finite State CCS Descriptions

by
J.P.M. Voeten
A. van Rangelrooij

EUT Report 95-E-292
ISBN 90-6144-292-3
July 1995

Eindhoven University of Technology Research Reports

EINDHOVEN UNIVERSITY OF TECHNOLOGY

Faculty of Electrical Engineering
Eindhoven, The Netherlands

ISSN 0167-9708 Coden: TEUEDE

CCS and Time: A Practical and Comprehensible Approach
to a Performance Evaluation of Finite State CCS Descriptions

by

J.P.M. Voeten
A. van Rangelrooij

EUT Report 95-E-292
ISBN 90-6144-292-3

Eindhoven
July 1995

CIP-DATA KONINKLIJKE BIBLIOTHEEK, DEN HAAG

Voeten, J.P.M.

CCS and time : a practical and comprehensible approach to a performance evaluation of finite state CCS descriptions / by J.P.M. Voeten, A. van Rangelrooij. - Eindhoven : Eindhoven University of Technology, Faculty of Electrical Engineering. - Fig., tab. - (EUT report, ISSN 0167-9708 ; 95-E-292)
With ref.
ISBN 90-6144-292-3
NUGI 832
Subject headings: formal specification / communication systems ; performance evaluation / CCS.

CCS and Time: A Practical and Comprehensible Approach
to a Performance Evaluation of Finite State CCS Descriptions

J.P.M. Voeten and A. van Rangelrooij

Abstract

Since Milner's CCS (Calculus of Communicating Systems) abstracts from absolute (real) time, it not suitable for the analysis of time properties of real-time systems. A lot of research has already been done (and is still going on) to incorporate notions of time into process algebras and process calculi. However, most approaches are still in a theoretical stage and are not readily comprehensible for non-experts.

In this report we define a simple yet practically applicable extension of CCS which allows for the specification of time behaviour and for a performance evaluation of finite state CCS descriptions. First, we introduce a finite state version of CCS, called fsCCS (finite state CCS). Next, we explain the shortcomings with respect to describing real-time behaviour, and we propose a simple solution resulting in a discrete time version of CCS, called dtCCS (discrete time CCS). Finally, we develop a method to calculate so-called mean-performance figures of systems of communicating processes described in dtCCS, which is based on a stochastic performance-analysis model. As an example we apply all this to a version of the well-known alternating-bit protocol.

Keywords: formal specification, performance evaluation, CCS.

Voeten J.P.M. and A. Van Rangelrooij
CCS and time: A practical and comprehensible approach
to a performance evaluation of finite state CCS descriptions
Eindhoven : Faculty of Electrical Engineering, Eindhoven University of Technology, 1995.
EUT Report 95-E-292

Address of the authors:
Section of Digital Information Systems
Faculty of Electrical Engineering
Eindhoven University of Technology
P.O. Box 513, NL-5600 MB Eindhoven, The Netherlands

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

Currently, there is a growing interest within the Digital Information Systems Group in the use of formal specification, description, and verification techniques for the development of complex digital information systems. One of the techniques we are looking at is the process calculus CCS (Calculus of Communicating Systems) [Mil80, Mil89, Koo91]. CCS is one of the first theories which deals in a formal way with the communication behaviour of parallel systems. Over the past years it has proven its usefulness in the specification and verification of complex parallel systems. A number of currently-used process calculi and process algebras are (partially) based on CCS. Examples include ACP (Algebra of Communicating Processes) [Bae86], LOTOS (Language Of Temporal Ordering Specification) [EVD89], and PSF (Process Specification Formalism) [MV90].

Using CCS we can only specify the *functional* behaviour of a system, and only describe and verify the *functional* behaviour of (potential) implementations, all abstracted from absolute (real) time. In this report we present a practical and comprehensible approach to a performance evaluation of finite state CCS descriptions, which allows us to describe both the *time* and *functional* behaviour and to compare the performance of the implementations.

Section 1.1 introduces the approach. Section 1.2 describes globally an application of this approach and motivates the development of the latter. Section 1.3 gives an overview of the organization of the remainder of this report.

1.1 CCS and Time

The traditional method of analyzing the *functional* behaviour of a system of communicating processes is performed by calculating the combined functional behaviour of this system, abstracted from internal computation (expansion). This combined system has an external *functional* behaviour equivalent to the original system. The advantage is that the external behaviour can be studied using a *single* (sequential) process, which is a lot easier that studying the original description with its possibly huge amount of constituents.

To analyze the *time* behaviour of a system of communicating processes, we have roughly the following two possibilities:

- simulation
- calculation

Simulation can be used to obtain information about the time behaviour and the performance of a system, but has several disadvantages. First, somehow a simulation model of the system must be created. Second, the time behaviour and performance figures must somehow be derived from the obtained simulation results. Third, simulation only provides a "snapshot" of the static behaviour of a system, and to obtain accurate results a large number of "snapshots" must be taken, which takes a lot of time and computation power. Further, simulation fails to present a *total* overview of the combined behaviour of a system.

We have developed a *calculation* method which follows the method to analyze the functional behaviour, which enables us to derive the wanted information (almost) *directly* from the CCS description. The result is a single formal description of both the external *functional* behaviour and the external *time* behaviour of a system of communicating processes. A lot of research has already been done (and is still going on) into similar approaches. Examples can be found in [BB92, MT90, ST92]. However, most, if not all, of these approaches are theoretical and not readily comprehensible for non-experts, probably due to the fact that there is not yet a unified, established, and generally accepted formal notion of time. Further, using another formalism means transforming CCS descriptions into other descriptions, which can induce various problems due to different concepts and notions. Therefore, we have chosen to "extend" CCS with a notion of real time. The resulting process calculus is called dtCCS (discrete time CCS) which allows us to describe both the *time* behaviour and the *functional* behaviour of systems of communicating processes in a *practical* and *comprehensible* way. dtCCS builds upon the notions and definitions of a finite-state version of CCS, called fsCCS (finite state CCS), which has been developed specifically for this purpose.

Since a dtCCS description of a system of communicating processes contains all information about the time behaviour originally present in the system, we are able to derive various performance figures from such a description. A very useful figure for the performance of a process A, given some action a, is the average number of a actions performed per time unit. We will call this number the *mean a-performance* of process A. We have developed a method to calculate such performance figures of systems of communicating processes described in dtCCS, using a stochastic performance-analysis model.

1.2 Application and Motivation

The methods described in the previous section have been developed as part of the first phase of a project for the development of a design methodology for (application-specific)

HMPs (Heap Management Processors) [Van94]. This first phase consists of the following parts:

- a literature study of heap management aspects
- development of a design methodology for system architectures for HMPs
- design and evaluation of a number of system architectures for HMPs using this design methodology

In this project we have used a design approach which is reflected in the basic design cycle described in [Koo91] and shown in figure 1.1. The literature study revealed that heap management consists of several tasks which can be performed in parallel. These tasks can be distributed over several functional blocks, which can be used as *building blocks* to *synthesize* or *construct* a number of system architectures (possibly) implementing the given *specification*. To ensure that these (potential) *implementations* are functionally correct, their functional behaviour must be *verified* against the specification.

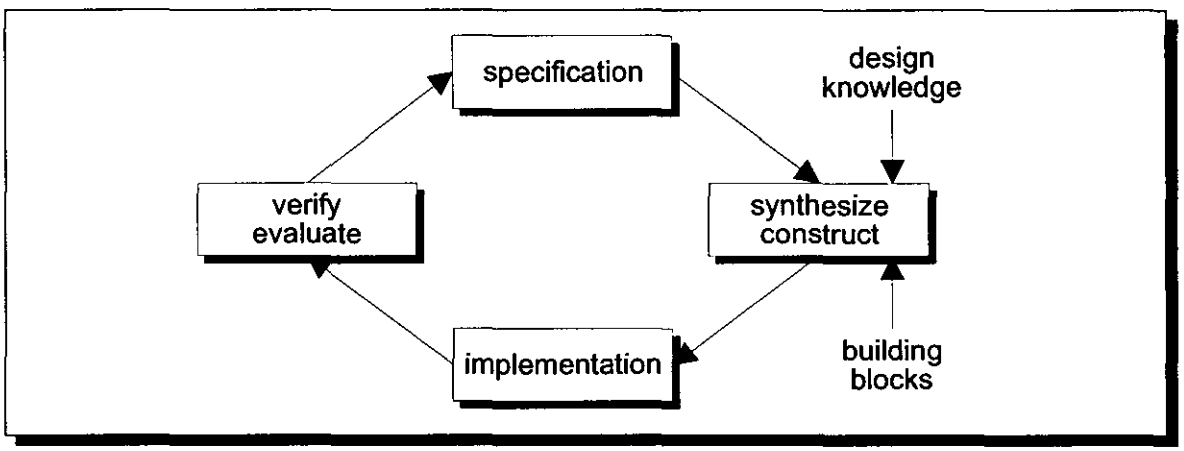

Figure 1.1: Basic design cycle

We would also like to *evaluate* the time behaviour of these (potential) implementations. This evaluation can give us information about the (possible) communication bottlenecks in a system architecture which can be used to improve the various architectures to overcome these bottlenecks. Further, we would like to compare the performance of the various architectures.

If we combine this basic design cycle with the methods described in the previous section and a number of automated tools supporting these methods, we get a design methodology for system architectures for (application-specific) HMPs, which consists of the following components citeran94:

- a basic design cycle, consisting of a specification phase, a synthesize phase, an implementation phase, and a verification phase
- a process calculus called fsCCS to specify, describe, and verify the *functional* behaviour of systems of communicating processes
- an automated tool called CCStool2 to support this
- a process calculus called dtCCS to describe both the *functional* behaviour and the *time* behaviour of such systems
- an automated tool called dtCCStool1 to support this
- a method to calculate the *mean-performance* figures of one or more actions of such systems, using a *stochastic performance-analysis model*
- an automated tool called PAtool1 to transform dtCCS descriptions into such a model
- an automated tool called Mathematica [Wol91] to support the calculation of these mean-performance figures from such a model

This design methodology has been applied as follows citeran94:

1. Specify the functional behaviour of a heap.
2. Repeat the following steps for several HMP system architectures:
   - 2a. Synthesize a system architecture using building blocks, resulting in a description of the functional behaviour of this architecture.
   - 2b. Complement this description with that of the necessary external memories, resulting in a description of the functional behaviour of a heap implementation.
   - 2c. Verify this latter behaviour against that of the heap specification.
   - 2d. Describe the time behaviour of the system architecture and of the external memories.
   - 2e. Calculate the mean-performance figures of this architecture.
3. Compare the mean-performance figures of the various system architectures.

1.3 Report Organization

In this report we only highlight fsCCS, dtCCS, and the method for the calculation of the mean-performance figures. The functionality of CCStool2, dtCCStool1, and PAtool1 is described in [Van94] and also in [VV94b], [VV94c], and [VV94d] respectively. Information about the inner workings of CCStool2 (and, since dtCCStool1 and PAtool1 are partly based

on CCStool2, also partly about the inner workings of these latter two tools) can be found in [VV94]. Information on how to install and use these three tools is given in [VV94b], [VV94c], and [VV94d] respectively.

The remainder of this report is organized as follows:

- Chapter 2 introduces fsCCS.
- Chapter 3 presents dtCCS.
- Chapter 4 describes the method for the calculation of the mean-performance figures.
- Chapter 5 gives an example of the application of dtCCS and the performance-calculation method.
- Chapter 6 presents the conclusions and the recommendations for future work.
- Appendix A contains the proofs of a number of propositions and lemmas of dtCCS.

Chapter 2

Finite State CCS

CCS (Calculus of Communicating Systems) [Mil80, Mil89, Koo91] is one of the first formal description techniques which deals in a formal way with the communication behaviour of processes, called *agents*, and is based upon the following two concepts:

- *Synchronized communication.* A system is built from independent agents which communicate synchronously. *Parallel composition* is used to compose two independent agents, allowing them to communicate.
- *Observation.* Systems are described fully enough to determine the behaviour to be seen or experienced by an external observer. Two systems are indistinguishable if we cannot tell them apart without pulling them apart. *Observational equivalence* is based upon this notion.

In this chapter we introduce a finite-state version of CCS, called fsCCS (finite state CCS), which has primarily been developed to provide a basis for the process calculus dtCCS (discrete time CCS, see Chapter 3).

Section 2.1 presents the basic concepts of fsCCS. Section 2.2 describes the semantics of fsCCS. Section 2.3 introduces notions of equivalence and so-called CCS laws.

2.1 Basics of fsCCS

Agents are externally observable through so-called *labels* they can perform. Observing an agent means communicating with this agent. Agents can communicate by synchronizing on complementary labels. Given a label l, its complement is written as $\overline{l}$. We will denote the set of all labels with $\mathcal{L}$ and will assume this set to be closed under complementation. If $L \subseteq \mathcal{L}$ is a set of labels, $\overline{L}$ denotes $\{\overline{l} \mid l \in L\}$. Next to observable actions, CCS knows a special so-called silent or perfect action, which is denoted as τ. The set of labels $\mathcal{L}$ together with τ forms the set of so-called *actions* $Act = \mathcal{L} \cup \{\tau\}$.

The behaviour of *sequential* finite-state agents is specified in terms of behaviour equations, which are of the form

$$A \stackrel{def}{=} \sum_{i \in I} \alpha_i \cdot A_i$$

A is a sequential *agent constant* and denotes the agent to specify. The set of all sequential agent constants is denoted as $\mathcal{P}$ with typical elements $A, B, \ldots$. We assume that $\mathcal{P}$ is countable infinite. The expression to the right of the $\stackrel{def}{=}$ sign denotes that agent A can choose to perform any action α_i, after which it behaves as agent A_i. We will say that agent A has made a transition from A to A_i, or just that A has made a transition. I denotes some index set of positive integers, so, $I \subseteq \mathbb{N} \setminus \{0\}$. The expression can also be written as $\sum\{\alpha_i \cdot A_i \mid i \in I\}$. If $I = \{1, \ldots, n\}$ we will often write $\alpha_1 \cdot A_1 + \cdots + \alpha_n \cdot A_n$. If $I = \varnothing$ we will abbreviate the expression to $\mathbf{0}$, which denotes the *inactive* agent, capable of performing no action whatsoever.

Further, we define the set of *agent expressions* $\mathcal{E}$. These expressions specify the behaviour of *systems* of sequential communicating agents. We let $E, F, \ldots$ range over $\mathcal{E}$, which is genetically defined as

- $A \in \mathcal{E}$, every *agent constant* A is an agent expression
- $\sum_{i \in I} \alpha_i \cdot A_i \in \mathcal{E}$, $I \subseteq \mathbb{N} \setminus \{0\}$
- $E \mid F \in \mathcal{E}$, the *parallel composition* of E and F
- $E \setminus L \in \mathcal{E}$, the *restriction* of E with respect to a set of labels $L \subseteq \mathcal{L}$
- $E[f] \in \mathcal{E}$, the *action relabeling* of E as dictated by function f

Function f denotes an *action-relabeling function* from Act to Act with the following properties:

- $f(\overline{l}) = \overline{f(l)}$
- $f(\tau) = \tau$

Often we will write $l'_1/l_1, \ldots, l'_n/l_n$ for the relabeling function f for which $f(l_i) = l'_i$, $f(\overline{l_i}) = \overline{l'_i}$ for $i = 1, \ldots, n$ and $f(l) = l$ otherwise.

As an example we consider the parallel composition of two 1-place buffers called $BufA_0$ and $BufB_0$ as shown in figure 2.1.

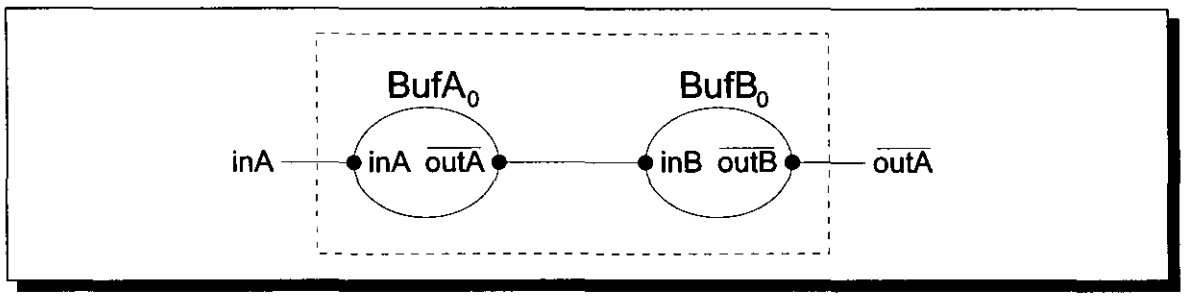

Figure 2.1: Parallel composition of two 1-place buffers

These buffers may be specified by the following fsCCS expression:

$$\Big(\big(BufA_0 \mid (BufB_0\,[outA/inB])\big) \setminus \{outA\}\Big)\,[outA/outB]$$

where

$$\begin{aligned} BufA_0 &\stackrel{def}{=} inA \cdot BufA_1 \\ BufA_1 &\stackrel{def}{=} \overline{outA} \cdot BufA_0 \end{aligned}$$

$$\begin{aligned} BufB_0 &\stackrel{def}{=} inB \cdot BufB_1 \\ BufB_1 &\stackrel{def}{=} \overline{outB} \cdot BufB_0 \end{aligned}$$

Buffer $BufA_0$ can perform actions inA and $\overline{outA}$ alternately. Performing these actions can be interpreted respectively as accepting incoming data at port inA and delivering this data at port $\overline{outA}$. Note that the data is not explicitly represented; only the communication behaviour is taken into account. Buffer $BufB_0$ has a similar behaviour.

As mentioned previously, agents can only communicate by performing complementary actions. Consequently, if we want $BufA_0$ and $BufB_0$ to communicate, we have to relabel the actions on which the buffers have to synchronize. In the example we have renamed action inB to $outA$, which implies that the buffers can synchronize on actions $outA$ and $\overline{outA}$.

Further, because we are only interested in the externally observable actions of the parallel composition of $BufA_0$ and $BufB_0$, we abstract from the (internal) communication of $outA$ and $\overline{outA}$. This is indicated through the action restriction $\setminus\{outA\}$, which means that both $outA$ and $\overline{outA}$ are externally unobservable.

Finally, we relabel action $\overline{outB}$ to $\overline{outA}$, indicated with $[outA/outB]$, to describe an agent which behaves as a 2-place buffer with actions inA and $\overline{outA}$.

2.2 Semantics of fsCCS

The semantics of fsCCS is given as a number of so-called *inference rules*, which are of the form

$$\frac{\text{hypotheses}}{\text{conclusion}}$$

Each rule has zero or more *hypotheses* and a *conclusion.* In a rule associated with a combinator ($\sum$, $|$, $\backslash$, $[f]$), the conclusion will be a transition of an agent expression consisting of the combinator applied to one or more components, and the hypotheses will be transitions of some of the components. The set of rules associated with a combinator can be understood as giving the meaning of that combinator. The rule for agent constants asserts that each constant has the same transitions as its defining expression [Mil89].

The complete set of inference rules is as follows:

fsCon
$$\frac{A \stackrel{def}{=} \sum_{i \in I} \alpha_i \cdot A_i}{A \stackrel{\alpha_j}{\rightarrow} A_j}$$

fsSum
$$\frac{}{\sum_{i \in I} \alpha_i \cdot A_i \stackrel{\alpha_j}{\rightarrow} A_j}$$

fsCom$_1$
$$\frac{E \stackrel{\alpha}{\rightarrow} E'}{E \mid F \stackrel{\alpha}{\rightarrow} E' \mid F}$$

fsCom$_2$
$$\frac{F \stackrel{\alpha}{\rightarrow} F'}{E \mid F \stackrel{\alpha}{\rightarrow} E \mid F'}$$

fsCom$_3$
$$\frac{E \stackrel{\alpha}{\rightarrow} E' \quad F \stackrel{\overline{\alpha}}{\rightarrow} F'}{E \mid F \stackrel{\tau}{\rightarrow} E' \mid F'}$$

fsRes
$$\frac{E \stackrel{\alpha}{\rightarrow} E'}{E \setminus L \stackrel{\alpha}{\rightarrow} E' \setminus L} \quad \text{if } \alpha, \overline{\alpha} \notin L$$

fsRel
$$\frac{E \stackrel{\alpha}{\rightarrow} E'}{E[f] \stackrel{f(\alpha)}{\rightarrow} E'[f]}$$

The names **fsCon**, **fsSum**, **fsCom**, **fsRes**, and **fsRel** indicate that the corresponding rules are associated with respectively finite-state *agent constants*, *summation*, *parallel composition*, *restriction*, and *action relabeling.*

2.3 Equivalences and CCS Laws

CCS defines several equivalences upon agents. The most famous and most used equivalences are *strong equivalence* (denoted as $\sim$), *observational equivalence* ($\approx$), and *observational congruence* (denoted as $=$). The formal definition of these equivalences can be found in [Mil80, Mil89]. In the context of these equivalences a number of so-called *CCS laws* are defined. For a complete overview of these laws, see [Mil89].

A very useful CCS law is the so-called *expansion law.* This law states how a number of independent communicating agents can be combined to form a single agent, which is strong equivalent. The original expansion law can be found in [Mil89]. Here, we will formulate a similar law for fsCCS. Before we state this law we introduce a family Φ of so-called *expression-naming functions.* Such a function is a one-to-one function from agent expressions $\mathcal{E}$ to agent constants $\mathcal{A}$. The idea of such a function is to map each agent expression E uniquely onto an agent constant A.

Proposition 2.1
The fsCCS expansion law. Let ϕ be an expression-naming function. If we define for all $E \in \mathcal{E}$

$$\phi(E) \stackrel{def}{=} \sum\{\alpha \cdot \phi(E') \mid E \stackrel{\alpha}{\rightarrow} E'\}$$

then for all agent expressions E

$$E \sim \phi(E)$$

□

In practice, the expansion law is often applied to systems of the form $(A_1[f_1] \mid \ldots \mid A_n[f_n]) \setminus L$. Then it is convenient to have a law which makes it easy to calculate all transitions the systems can make. Such a law is given in Proposition 2.2.

Proposition 2.2
Let $A_1, \ldots, A_n$, for $n \geq 1$, be sequential agent constants, let $f_1, \ldots, f_n$ be relabeling functions, and let L be a set of labels. Then $(A_1[f_1] \mid \ldots \mid A_n[f_n]) \setminus L \stackrel{\alpha}{\rightarrow} G$ if and only if

$$G \equiv A_1[f_1] \mid \ldots \mid A_i'[f_i] \mid \ldots \mid A_n[f_n] \text{ and } A_i \stackrel{\beta}{\rightarrow} A_i' \text{ and } \alpha = f_i(\beta) \notin L \cup \overline{L}$$

or

$$G \equiv A_1[f_1] \mid \ldots \mid A_i'[f_i] \mid \ldots \mid A_j'[f_j] \mid \ldots \mid A_n[f_n], \alpha = \tau \text{ and } A_i \stackrel{l_1}{\rightarrow} A_i', A_j \stackrel{l_2}{\rightarrow} A_j' \text{ and } f_i(l_1) = \overline{f_j(l_2)}$$

□

Milner's original expansion law combines Proposition 2.1 and Proposition 2.2 into a single law.

If we apply the expansion law (Proposition 2.1) and Proposition 2.2 to the parallel composition of the two 1-place buffers in the example of Section 2.1, we get the following result:

$$\Big(\big(BufA_i \mid (BufB_j\,[outA/inB])\big) \setminus \{outA\}\Big)\,[outA/outB] \sim BufA_i \mid BufB_j$$

for $i, j = 0, 1$, where

$$\begin{aligned}
BufA_0 \mid BufB_0 &\stackrel{def}{=} inA \cdot BufA_1 \mid BufB_0 \\
BufA_1 \mid BufB_0 &\stackrel{def}{=} \tau \cdot BufA_0 \mid BufB_1 \\
BufA_0 \mid BufB_1 &\stackrel{def}{=} inA \cdot BufA_1 \mid BufB_1 + outA \cdot BufA_0 \mid BufB_0 \\
BufA_1 \mid BufB_1 &\stackrel{def}{=} outA \cdot BufA_1 \mid BufB_0
\end{aligned}$$

Here, we have used an expression-naming function ϕ with the following property:

$$\phi\Big(\Big(\big(BufA_i \mid (BufB_j\,[outA/inB])\big) \setminus \{outA\}\Big)\,[outA/outB]\Big) = BufA_i \mid BufB_j$$

for $i, j = 0, 1$.[1]

[1] It can be proved that any partial one-to-one function f from $\mathcal{E}$ to $\mathcal{A}$ for which $\mathcal{A} \setminus Rng(f)$ is (countably) infinite can be extended to an expression-naming function.

Chapter 3

Discrete Time CCS

Using fsCCS (finite-state CCS, see Chapter 2) we can only specify the *functional* behaviour of a system, and only describe and verify the *functional* behaviour of (potential) implementations, all abstracted from absolute (real) time. In this chapter we describe the process calculus dtCCS (discrete time CCS) which allows us to describe both the *time* behaviour and the *functional* behaviour of systems of communicating processes in a *practical* and *comprehensible* way. dtCCS builds upon the notions and definitions of fsCCS.

Section 3.1 gives an informal introduction into dtCCS. Section 3.2 presents the basics and semantics of *basic* dtCCS. Section 3.3 describes a notion of equivalence and an expansion law based on this notion for *basic* dtCCS. Section 3.4 gives an informal introduction into a concept called *maximal progress*. Section 3.5 presents the basics and semantics of *full* dtCCS, which is *basic* dtCCS extended with *maximal progress*. Section 3.7 introduces a notion of equivalence and an expansion law based on this notion for *full* dtCCS. Section 3.9 presents a notation for so-called multi time-step atoms and a way to express such atoms in dtCCS.

3.1 fsCCS and Time

As an example we showed in Section 2.3 how two 1-place buffers (shown in figure 3.1) can be combined into a 2-place buffer using expansion.
These buffers are specified by the following fsCCS expression:

$$\Big(\big(BufA_0 \mid (BufB_0\,[outA/inB])\big) \setminus \{outA\}\Big)\,[outA/outB]$$

where

$$BufA_0 \stackrel{def}{=} inA \cdot BufA_1$$
$$BufA_1 \stackrel{def}{=} \overline{outA} \cdot BufA_0$$

$$BufB_0 \stackrel{def}{=} inB \cdot BufB_1$$
$$BufB_1 \stackrel{def}{=} \overline{outB} \cdot BufB_0$$

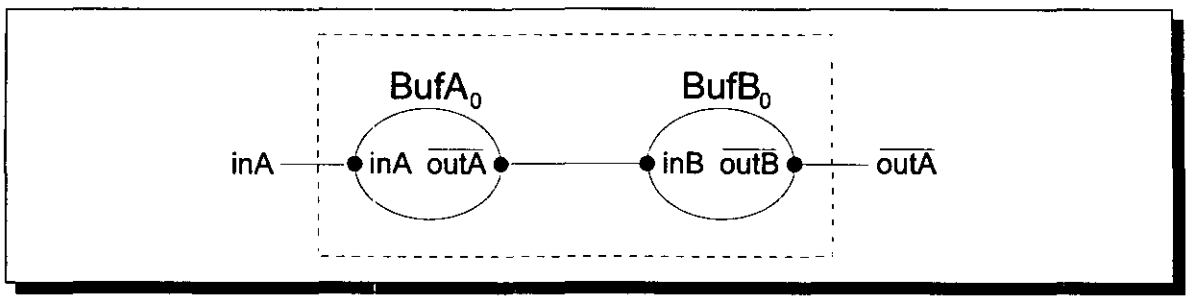

Figure 3.1: Parallel composition of two 1-place buffers

The parallel composition is

$$
\begin{aligned}
BufA_0 \mid BufB_0 &\stackrel{def}{=} inA \cdot BufA_1 \mid BufB_0 \\
BufA_1 \mid BufB_0 &\stackrel{def}{=} \tau \cdot BufA_0 \mid BufB_1 \\
BufA_0 \mid BufB_1 &\stackrel{def}{=} inA \cdot BufA_1 \mid BufB_1 + \overline{outA} \cdot BufA_0 \mid BufB_0 \\
BufA_1 \mid BufB_1 &\stackrel{def}{=} \overline{outA} \cdot BufA_1 \mid BufB_0
\end{aligned}
$$

This example clearly shows how expansion in fsCCS *flattens real parallelism.* If both buffers are actually executing in parallel, actions inA and $\overline{outA}$ can be performed simultaneously if the buffers are in states $BufA_0$ and $BufB_1$ respectively. This cannot be seen by inspecting the result of the expansion. This results from the fact that fsCCS abstracts from real time. The 2-place buffer is certainly *observational equivalent* to the parallel composition of the two 1-place buffers, but not equivalent with respect to *time behaviour.* If we assume that the actions both 1-place buffers can perform take a single time-step and that the "environment" of the buffers is always willing to participate in all actions the buffers want to perform, we would like to represent the behaviour of the 2-place buffer as follows:

$$
\begin{aligned}
BufA_0 \mid BufB_0 &\stackrel{def}{=} [\![inA]\!] \cdot BufA_1 \mid BufB_0 \\
BufA_1 \mid BufB_0 &\stackrel{def}{=} [\![\,]\!] \cdot BufA_0 \mid BufB_1 \\
BufA_0 \mid BufB_1 &\stackrel{def}{=} [\![inA, \overline{outA}]\!] \cdot BufA_1 \mid BufB_0
\end{aligned}
$$

Here, $[\![inA, \overline{outA}]\!]$ denotes that actions inA and $\overline{outA}$ are performed simultaneously. Such a *bag* of labels will be called an *atom.* The empty bag $[\![\,]\!]$ represents the atom which can only perform a time step, and can be compared with the silent τ action.

3.2 Basic dtCCS

Agents in (basic) dtCCS reveal their activities by performing so-called *atoms.* An atom is a *bag* of labels. Such a bag is represented as a function from $\mathcal{L}$ to $\mathbb{N}$, where $\mathbb{N} = \{0, 1, 2, \ldots\}$. The set of all possible atoms $\mathcal{A}$ is defined as

$$\mathcal{A} = \mathbb{N}^{\mathcal{L}} \quad (= \{b \mid b : \mathcal{L} \to \mathbb{N}\})$$

We let $a_1, a_2, \ldots$ range over $\mathcal{A}$. Agents are defined in terms of behaviour equations as those described in Chapter 2, except for the fact that we write atoms instead of actions. We will assume that all atoms take a single time-step to execute.
We will use the binary union operator upon bags ($\uplus$) to calculate unions of atoms. Let a, b be atoms. The union of a and b is defined as

$$a \uplus b = c \text{ iff } \forall l : l \in \mathcal{L} : c(l) = a(l) + b(l)$$

We will say that atoms a and b *perfectly synchronize* if and only if every label l inside a can synchronize precisely with one label $\overline{l}$ inside b, and vice versa:

$$\forall l : l \in \mathcal{L} : a(l) = b(\overline{l})$$

An atom a is called a *partial synchronization* (a p.s.) of atoms b and c if a arises from the union of b and c by striking out zero or more pairs of complementary labels $(l, \overline{l}) \in b \times c$. Formally, this is expressed as

> $a = b' \uplus c'$ for atoms b' and c' for which $b = b' \uplus b''$ and $c = c' \uplus c''$ where b'' and c'' perfectly synchronize

If B is a bag of atoms, we let $PS(B)$ denote the set of all partial synchronizations of all atoms in B.

$$\begin{array}{lcl} PS(\llbracket\,\rrbracket) & = & \{\llbracket\,\rrbracket\} \\ PS(B \uplus \llbracket b \rrbracket) & = & \{a \in \mathcal{A} \mid \exists\, c : c \in PS(B) : a \text{ is a p.s. of } b \text{ and } c\} \end{array}$$

It can be proved that $PS((B \uplus \llbracket a \rrbracket) \uplus \llbracket b \rrbracket) = PS((B \uplus \llbracket b \rrbracket) \uplus \llbracket a \rrbracket)$. This property can be used to show that PS is well-defined.

Let f be a relabeling function. We define the relabeling of an atom a under f as follows:

$$f(a) = a' \text{ iff } \forall l : l \in \mathcal{L} : a'(f(l)) = a(l)$$

Note that $f(\llbracket\,\rrbracket) = \llbracket\,\rrbracket$. This is in accordance with the fsCCS convention that $f(\tau) = \tau$.

Armed with these definitions we are ready to define the semantics of basic dtCCS in terms of the following inference rules (see also Section 2.2):

$$\textbf{dtCon} \quad \frac{A \stackrel{def}{=} \sum_{i \in I} a_i \cdot A_i}{A \xrightarrow{a_i} A_i} \quad i \in I$$

$$\textbf{dtSum} \quad \frac{}{\sum_{i \in I} a_i \cdot A_i \xrightarrow{a_i} A_i}$$

dtCom$_1$ $\dfrac{E \xrightarrow{a} E'}{E \mid F \xrightarrow{a} E' \mid F}$

dtCom$_2$ $\dfrac{F \xrightarrow{a} F'}{E \mid F \xrightarrow{a} E \mid F'}$

dtCom$_3$ $\dfrac{E \xrightarrow{b} E' \quad F \xrightarrow{c} F'}{E \mid F \xrightarrow{a} E' \mid F'}$ if a is a p.s. of b and c

dtRes $\dfrac{E \xrightarrow{a} E'}{E \setminus L \xrightarrow{a} E' \setminus L}$ if $a \cap (L \cup \overline{L}) = \varnothing$

dtRel $\dfrac{E \xrightarrow{a} E'}{E[f] \xrightarrow{f(a)} E'[f]}$

The names **dtCon**, **dtSum**, **dtCom**, **dtRes**, and **dtRel** indicate that the corresponding rules are associated with respectively discrete time *agent constants*, *summation*, *parallel composition*, *restriction*, and *action relabeling*.

3.3 Strong Discrete Time Equivalence

In this section we will define when we consider two agents to be equivalent in basic dtCCS. Further, we will define a basic dtCCS expansion law similar to that for fsCCS defined in Section 2.3. We will start with the definition of what we call *strong discrete time bisimulations*.

Definition 3.1
A binary relation $\mathcal{S} \subseteq \mathcal{E} \times \mathcal{E}$ is a *strong discrete time bisimulation* if $(E, F) \in \mathcal{S}$ implies, for all atoms $a \in \mathcal{A}$,

(i) whenever $E \xrightarrow{a} E'$ then, for some F', $F \xrightarrow{a} F'$ and $(E', F') \in \mathcal{S}$

(ii) whenever $F \xrightarrow{a} F'$ then, for some E', $E \xrightarrow{a} E'$ and $(E', F') \in \mathcal{S}$

□

Definition 3.2
E and F are strong discrete time equivalent, written $E \sim F$, if $(E, F) \in \mathcal{S}$ for some strong discrete time bisimulation $\mathcal{S}$. So, $\sim = \bigcup \{\mathcal{S} \mid \mathcal{S} \text{ is a strong discrete time bisimulation }\}$.

□

3.3.1 Some Properties of Strong Discrete Time Equivalence

In this subsection we will address some useful properties of strong discrete time equivalence. We will not prove them here, but we will prove similar results for *full* dtCCS in Subsection 3.7.1.

Proposition 3.1

(1) $\sim$ is the largest strong discrete time bisimulation.

(2) $\sim$ is an equivalence relation.

□

Proposition 3.2

$E \sim F$ iff $(E, F) \in \mathcal{S}$ implies, for all atoms $a \in \mathcal{A}$,

(i) whenever $E \xrightarrow{a} E'$ then, for some F', $F \xrightarrow{a} F'$ and $E' \sim F'$

(ii) whenever $F \xrightarrow{a} F'$ then, for some E', $E \xrightarrow{a} E'$ and $E' \sim F'$

□

Proposition 3.3

(1) $\sum_{i \in I} a_i \cdot A_i \sim \sum_{j \in J} b_j \cdot B_j$ if there exists a bijective function $\nu : I \to J$ such that $a_i = b_{\nu(i)}$ and $A_i \equiv B_{\nu(i)}$ for all $i \in I$

(2) $\sum_{i \in I \cup \{k\}} a_i \cdot A_i \sim \sum_{i \in I} a_i \cdot A_i$ if $k \notin I$ and if there exists a $j \in I$ such that $a_j = a_k$ and $A_j \equiv A_k$

(3) $E \mid F \sim F \mid E$

(4) $(E \mid F) \mid G \sim (E \mid (F \mid G)$

□

Proposition 3.4

$\sim$ is a congruence relation: it is substitutive under all combinators, i.e., if $E_1 \sim E_2$ then

(1) $E_1 \mid F \sim E_2 \mid F$

(2) $E_1 \setminus L \sim E_2 \setminus L$

(3) $E_1[f] \sim E_2[f]$

□

The following law is called the *basic discrete time expansion law.* It describes how the behaviour of systems of communicating agents can be represented as a single (sequential) agent. The law is defined in terms of expression-naming functions defined in Section 2.3.

Proposition 3.5
The basic discrete time expansion law. Let $\phi \in \Phi$ be an expression-naming function. If we define for all $E \in \mathcal{E}$

$$\phi(E) \stackrel{def}{=} \sum \left\{ a \cdot \phi(E') \mid E \xrightarrow{a} E' \right\}$$

then for all agent expressions E

$$E \sim \phi(E)$$

□

3.3.2 Basic Discrete Time Expansion Example

In this subsection we will apply the basic discrete time expansion law to the example of the two 1-place buffers in Section 3.1. In basic dtCCS these buffers are specified as

$$\Big(\big(BufA_0 \mid (BufB_0\,[outA/inB])\big) \setminus \{outA\}\Big)\,[outA/outB]$$

where

$$\begin{aligned} BufA_0 &\stackrel{def}{=} [\![inA]\!] \cdot BufA_1 \\ BufA_1 &\stackrel{def}{=} [\![\overline{outA}]\!] \cdot BufA_0 \end{aligned}$$

$$\begin{aligned} BufB_0 &\stackrel{def}{=} [\![inB]\!] \cdot BufB_1 \\ BufB_1 &\stackrel{def}{=} [\![\overline{outB}]\!] \cdot BufB_0 \end{aligned}$$

Let ϕ be an expression-naming function which satisfies for $i, j = 0, 1$.

$$\phi\Big(\Big(\big(BufA_i \mid (BufB_j\,[outA/inB])\big) \setminus \{outA\}\Big)\,[outA/outB]\Big) = BufA_i \mid BufB_j$$

By the expansion law we now have

$$\Big(\big(BufA_i \mid (BufB_j\,[outA/inB])\big) \setminus \{outA\}\Big)\,[outA/outB] \sim BufA_i \mid BufB_j$$

for $i, j = 0, 1$ where

$$\begin{aligned} &BufA_i \mid BufB_j \stackrel{def}{=} \\ &\sum \left\{ a \cdot \phi(E') \mid \Big(\big(BufA_i \mid (BufB_j\,[outA/inB])\big) \setminus \{outA\}\Big)\,[outA/outB] \xrightarrow{a} E' \right\} \end{aligned}$$

To make these latter definitions more explicit, we have to calculate all derivations of

$$\Big(\big(BufA_i \mid (BufB_j\,[outA/inB])\big) \setminus \{outA\}\Big)\,[outA/outB]$$

for $i, j = 0, 1$. We shall do this only for the case $i = 0$ and $j = 0$. Using the inference rules of Section 3.2 we get

1. $BufA_0 \stackrel{[inA]}{\rightarrow} BufA_1$ (rule **dtCon**)
2. $BufB_0 \stackrel{[inB]}{\rightarrow} BufB_1$ (rule **dtCon**)
3. $BufB_0[outA/inB] \stackrel{[outA]}{\rightarrow} BufB_1[outA/inB]$ (rule **dtRel**)
4. $BufA_0 \mid (BufB_0[outA/inB])$
   $\stackrel{[inA]}{\rightarrow} BufA_1 \mid (BufB_0[outA/inB])$ (rule $\mathbf{dtCom_1}$)
   $\stackrel{[outA]}{\rightarrow} BufA_0 \mid (BufB_1[outA/inB])$ (rule $\mathbf{dtCom_2}$)
   $\stackrel{[inA,outA]}{\rightarrow} BufA_1 \mid (BufB_1[outA/inB])$ (rule $\mathbf{dtCom_3}$)
5. $\big(BufA_0 \mid (BufB_0[outA/inB])\big) \setminus \{outA\}$
   $\stackrel{[inA]}{\rightarrow} \big(BufA_1 \mid (BufB_0[outA/inB])\big) \setminus \{outA\}$ (rule **dtRes**)
6. $\big(\big(BufA_0 \mid (BufB_0[outA/inB])\big) \setminus \{outA\}\big)[outA/outB]$
   $\stackrel{[inA]}{\rightarrow} \big(\big(BufA_1 \mid (BufB_0[outA/inB])\big) \setminus \{outA\}\big)[outA/outB]$ (rule **dtRel**)

So, we have

$$BufA_0 \mid BufB_0 \stackrel{def}{=} [\![inA]\!] \cdot BufA_1 \mid BufB_0$$

and if we calculate the derivations of the remaining expressions we get

$$\begin{aligned}
BufA_1 \mid BufB_0 &\stackrel{def}{=} [\![\,]\!] \cdot BufA_0 \mid BufB_1 \\
BufA_0 \mid BufB_1 &\stackrel{def}{=} [\![inA]\!] \cdot BufA_1 \mid BufB_1 \\
&\quad + [\![\overline{outA}]\!] \cdot BufA_0 \mid BufB_0 \\
&\quad + [\![inA, \overline{outA}]\!] \cdot BufA_1 \mid BufB_0 \\
BufA_1 \mid BufB_1 &\stackrel{def}{=} [\![\overline{outA}]\!] \cdot BufA_1 \mid BufB_0
\end{aligned}$$

3.4 Maximal Progress: Informal Introduction

The resulting agent in the example above does not yet satisfy our final goal as described in Section 3.1. In state $BufA_0 \mid BufB_1$ it can perform atoms $[\![inA]\!]$ and $[\![\overline{outA}]\!]$ separately as well as simultaneously. Our final goal is to have an expansion in which all agents make "maximal progress" and in which we assume the environment to be willing always to participate in any communication initiated by these agents. This means that in the example the separate atoms $[\![inA]\!]$ and $[\![\overline{outA}]\!]$ have to be discarded.

The reader could wonder why these separate atoms were there in the first place. Since these atoms are a result of the inference rules $\mathbf{dtCom_1}$ and $\mathbf{dtCom_2}$, we might wonder why we need these inference rules. The idea is that initially a system is described without any knowledge or assumption about the environment in which it will be used. This means that a system cannot decide for itself whether certain communications can be performed;

these can only be performed if the environment is willing to participate in them. Assume, for example, that the environment of the 2-place buffer behaves as follows:

$$Env \stackrel{def}{=} [\![\overline{inA}]\!] \cdot Env'$$
$$Env' \stackrel{def}{=} [\![outA]\!] \cdot Env$$

Leaving out the separate atoms results in a deadlock, since the environment is not able to perform atoms $[\![\overline{inA}]\!]$ and $[\![outA]\!]$ simultaneously.

We are now stuck with a number of important questions. How can we construct a "maximal progress" agent from a given agent, and what does "maximal progress" mean in the first place? In general, a system consists of a number of communicating sequential agents, each in a certain state. Every (sequential) agent is able to perform zero or more transitions, possibly simultaneously with transitions of other agents. If each of a number of agents simultaneously performs one of its transitions, and if there is not another agent which could also participate by performing one of its transitions, even not if they choose to perform different transitions, we will say that the system makes maximal progress. If we inspect the expansion above, we see that in state $BufA_0 \mid BufB_1$ the buffer does not make maximal progress if it performs one of the transitions $[\![inA]\!]$ and $[\![\overline{outA}]\!]$ separately.

The idea is to extend relation $\stackrel{a}{\rightarrow}$, used to describe the semantics of basic dtCCS, with information about which sequential agents perform which of their transitions. This information will be represented by lists of natural numbers of the form $\langle n_1, n_2, \ldots, n_k \rangle$, and we will write

$$E \stackrel{a,\langle n_1,n_2,\ldots,n_k\rangle}{\rightarrow} E'$$

to indicate that

- system E can perform atom a after which it behaves as system E'
- E consists of k sequential agents, numbered from 1 to k
- sequential agent i performs transition n_i if $n_i > 0$
- sequential agent i is idle (does not perform any transition) if $n_i = 0$

As an example assume we have some system E which can perform a transition $E \stackrel{a,\langle 1,0,0,4\rangle}{\rightarrow} E'$. Further, assume also that $E \stackrel{b,\langle 1,5,0,4\rangle}{\rightarrow} E''$. The system makes better progress if it performs the latter transition instead of the former. If there exists no list of the form $\langle x, y, k, z \rangle$ with $x, y, z > 0$ for some $k > 0$ such that $E \stackrel{c,\langle x,y,k,z\rangle}{\rightarrow} E'''$, the system makes *maximal progress* by performing transition $E \stackrel{b,\langle 1,5,0,4\rangle}{\rightarrow} E''$. Note that there can exist different "incomparable" transitions which all establish maximal progress.

3.5 dtCCS

We will now extend the syntax of basic dtCCS with a maximal-progress operator $\mathcal{M}$. The new set of agent expressions becomes

- $A \in \mathcal{E}$
- $\sum_{i \in I} a_i \cdot A_i \in \mathcal{E}$ for $I \subseteq \mathbb{N} \setminus \{0\}$
- $E \mid F \in \mathcal{E}$
- $E \setminus L \in \mathcal{E}$
- $E[f] \in \mathcal{E}$
- $\mathcal{M}(E) \in \mathcal{E}$

Here, $\mathcal{M}(E)$ denotes a system E from which all nonmaximal-progress transitions have been discarded.

Definition 3.3
A list over natural numbers is an ordered n-tuple $\langle a_1, \ldots, a_n \rangle$, for $n \geq 0$ such that $a_i \in \mathbb{N}$ for all $i = 1, \ldots, n$. n denotes the number of elements of the list, and is called its arity. The set of all lists will be denoted $\mathbb{N}^*$. The set of all list of arity n will be denoted as $\mathbb{N}^n$. □

Given a list l, we write $\#l$ to indicate the arity of l. l_i denotes the i^{th} element of the list if $i = 1, \ldots, \#l$. For lists l and m we will write $l \cdot m$ for the concatenation of these lists. If $n \in \mathbb{N}$ we will write $\mathbf{0}^n$ to denote the list with n zeros.

We define an orderings relation $\sqsubseteq$ upon lists as follows:

Definition 3.4
Let $l, m \in \mathbb{N}^*$ be lists over $\mathbb{N}$. $l \sqsubseteq m$ if and only if

- $\#l = \#m$
- $\forall i : i = 1, \ldots, \#l : (l_i = 0) \vee (l_i = m_i)$

□

Intuitively, $l \sqsubseteq m$ means that a system E for which $E \xrightarrow{a,l} E'$ and $E \xrightarrow{b,m} E''$ makes more progress if its performs the latter transition. We will write $l \sqsubset m$, if $l \sqsubseteq m$ and $l \neq m$.

Definition 3.5
Let E be a dtCCS agent expression. $\#E$ denotes the amount of *sequential* agents inside E. We inductively define $\#E$ as

- $\#A = 1$
- $\#\sum_{i\in I} a_i \cdot A_i = 1$
- $\#(E \mid F) = \#E + \#F$
- $\#(E \setminus L) = \#E$
- $\#E[f] = \#E$
- $\#\mathcal{M}(E) = \#E$

□

We are now ready to define the semantics of dtCCS, which is defined by the following inference rules (see also Section 2.2):

$$\textbf{dtCon} \quad \frac{A \stackrel{def}{=} \sum_{i\in I} a_i \cdot A_i}{A \stackrel{a_i,\langle i\rangle}{\longrightarrow} A_i} \quad i \in I$$

$$\textbf{dtSum} \quad \frac{}{\sum_{i\in I} a_i \cdot A_i \stackrel{a_i,\langle i\rangle}{\longrightarrow} A_i} \quad i \in I$$

$$\textbf{dtCom}_1 \quad \frac{E \stackrel{a,l}{\longrightarrow} E'}{E \mid F \stackrel{a,l\cdot \mathbf{0}^{\#F}}{\longrightarrow} E' \mid F}$$

$$\textbf{dtCom}_2 \quad \frac{F \stackrel{a,l}{\longrightarrow} F'}{E \mid F \stackrel{a,\mathbf{0}^{\#E}\cdot l}{\longrightarrow} E \mid F'}$$

$$\textbf{dtCom}_3 \quad \frac{E \stackrel{a,l}{\longrightarrow} E' \quad F \stackrel{b,m}{\longrightarrow} F'}{E \mid F \stackrel{c,l\cdot m}{\longrightarrow} E' \mid F'} \quad \text{if } c \text{ is a p.s. of } a \text{ and } b$$

$$\textbf{dtRes} \quad \frac{E \stackrel{a,l}{\longrightarrow} E'}{E \setminus L \stackrel{a,l}{\longrightarrow} E' \setminus L} \quad \text{if } a \notin (L \cup \overline{L}) = \varnothing$$

$$\textbf{dtRel} \quad \frac{E \stackrel{a,l}{\longrightarrow} E'}{E[f] \stackrel{f(a),l}{\longrightarrow} E'[f]}$$

$$\textbf{dtMProg} \quad \frac{E \stackrel{a,l}{\longrightarrow} E'}{\mathcal{M}(E) \stackrel{a,l}{\longrightarrow} \mathcal{M}(E')} \quad \text{if } \neg\, \exists\, b, m, F : E \stackrel{b,m}{\longrightarrow} F : l \sqsubset m$$

The names **dtCon**, **dtSum**, **dtCom**, **dtRes**, **dtRel**, and **dtMProg** indicate that the corresponding rules are associated with respectively discrete time *agent constants*, *summation*, *parallel composition*, *restriction*, *action relabeling*, and *maximal progress*.

3.6 Super Strong Discrete Time Equivalence

In Section 3.3 we have defined the notion of *strong discrete time equivalence* ($\sim$) for basic dtCCS. We would like to adopt this notion for full dtCCS, but unfortunately strong discrete time equivalence is not substitutive under $\mathcal{M}$. So, if $E \sim F$ then it is not necessarily true that $\mathcal{M}(E) \sim \mathcal{M}(F)$. We would like to have an equivalence relation which is substitutive under *all* combinators. Our first equivalence relation enjoying this property is *super strong discrete time equivalence.*

Definition 3.6
A binary relation $\mathcal{S} \subseteq \mathcal{E} \times \mathcal{E}$ is a super strong discrete time bisimulation if $(E,F) \in \mathcal{S}$ implies, for all $a \in \mathcal{A}$,

(i) whenever $E \stackrel{a,l}{\rightarrow} E'$ then, for some F', $F \stackrel{a,l}{\rightarrow} F'$ and $(E',F') \in \mathcal{S}$

(ii) whenever $F \stackrel{a,l}{\rightarrow} F'$ then, for some E', $E \stackrel{a,l}{\rightarrow} E'$ and $(E',F') \in \mathcal{S}$

□

Definition 3.7
E and F are super strong discrete time equivalent, written $E \stackrel{s}{\sim} F$, if $(E,F) \in \mathcal{S}$ for some super strong discrete time bisimulation $\mathcal{S}$. So, $\stackrel{s}{\sim} = \bigcup \{\mathcal{S} \mid \mathcal{S}$ is a super strong discrete time bisimulation $\}$.

□

3.6.1 Some Properties of Super Strong Discrete Time Equivalence

Super strong discrete time equivalence enjoys various properties. This subsection describes a number of them.

Proposition 3.6
(1) $\stackrel{s}{\sim}$ is the largest super strong discrete time bisimulation.

(2) $\stackrel{s}{\sim}$ is an equivalence relation.

□

Proposition 3.7
$E \stackrel{s}{\sim} F$ iff, for all $a \in \mathcal{A}$,

(i) whenever $E \stackrel{a,l}{\rightarrow} E'$ then, for some F', $F \stackrel{a,l}{\rightarrow} F'$ and $E' \stackrel{s}{\sim} F'$

(ii) whenever $F \stackrel{a,l}{\rightarrow} F'$ then, for some E', $E \stackrel{a,l}{\rightarrow} E'$ and $E' \stackrel{s}{\sim} F'$

□

Proposition 3.8
$(E \mid F) \mid G \overset{s}{\sim} (E \mid (F \mid G)$

□

This latter proposition gives us the liberty to write expressions like $E_1 \mid E_2 \mid E_3 \mid \cdots \mid E_n$ instead of, for example, $(\cdots((E_1 \mid E_2) \mid E_3)\cdots \mid E_n)$. Propositions 3.9 and 3.10 given in this subsection make use of this liberty.

Proposition 3.9
$\overset{s}{\sim}$ is a congruence relation: it is substitutive under all combinators, i.e., if $E_1 \overset{s}{\sim} E_2$ then

(1) $E_1 \mid F \overset{s}{\sim} E_2 \mid F$

(2) $E_1 \setminus L \overset{s}{\sim} E_2 \setminus L$

(3) $E_1[f] \overset{s}{\sim} E_2[f]$

(4) $\mathcal{M}(E_1) \overset{s}{\sim} \mathcal{M}(E_2)$

□

Proposition 3.10
Let $A_1, \cdots, A_n$, for $n \geq 1$, be sequential agent constants and let $f_1, \cdots, f_n$ be relabeling functions. Then $A_1[f_1] \mid \cdots \mid A_n[f_n] \xrightarrow{a,l} G$ if and only if $l \in \mathbb{N}^n \setminus \{\mathbf{0}^n\}$ and there exist atoms $a_1, \cdots, a_n$ and agent constants $A'_1, \cdots, A'_n$ such that $a \in PS(\llbracket f_1(a_1), \cdots, f_n(a_n) \rrbracket)$, $G \equiv A'_1[f_1] \mid \cdots \mid A'_n[f_n]$ and

$$\begin{array}{lll}
A_1 & \xrightarrow{a_1,\langle l_1 \rangle} & A'_1 \text{ or}(l_1 = 0 \wedge A'_1 \equiv A_1 \wedge a_1 = \llbracket\rrbracket) \\
A_2 & \xrightarrow{a_2,\langle l_2 \rangle} & A'_2 \text{ or}(l_2 = 0 \wedge A'_2 \equiv A_2 \wedge a_2 = \llbracket\rrbracket) \\
\vdots & \vdots & \vdots \\
A_n & \xrightarrow{a_n,\langle l_n \rangle} & A'_n \text{ or}(l_n = 0 \wedge A'_n \equiv A_n \wedge a_n = \llbracket\rrbracket)
\end{array}$$

□

The proof of this latter proposition can be found in Appendix A.

3.7 Very Strong Discrete Time Equivalence

Super strong discrete time equivalence, defined in the previous section, has some useful properties. Unfortunately, there are a number of important properties which are not enjoyed by this equivalence relation. For example, Properties 3.3(1), 3.3(2) and 3.3(3)! given in Subsection 3.3.1 are not valid under $\overset{s}{\sim}$. Further, behaviour expressions with a different amount of sequential components, are never equivalent under $\overset{s}{\sim}$. Therefore, this relation does not allow for an expansion law similar to that of Subsection 3.3.1! Therefore, we define a second equivalence relation which is substitutive under *all* combinators, and which

enjoys *all* properties given in Subsection 3.3.1. The relation is called *very strong discrete time equivalence*. It is stronger than $\sim$, but weaker than $\overset{s}{\sim}$.

Definition 3.8
A binary relation $\mathcal{S} \subseteq \mathcal{E} \times \mathcal{E}$ is a very strong discrete time bisimulation if $(E, F) \in \mathcal{S}$ implies, for all $a \in \mathcal{A}$,

(i) whenever $E \overset{a,l_1}{\rightarrow} E'$ then, for some m_1, F', $F \overset{a,m_1}{\rightarrow} F'$ and $(E', F') \in \mathcal{S}$ and

- if $F \overset{b,m_2}{\rightarrow} F''$ such that $m_1 = m_2$ then, for some E'' and l_2 with $l_1 = l_2$, $E \overset{b,l_2}{\rightarrow} E''$ and
- if $F \overset{b,m_2}{\rightarrow} F''$ such that $m_1 \sqsubset m_2$ then, for some E'' and l_2 with $l_1 \sqsubset l_2$, $E \overset{b,l_2}{\rightarrow} E''$

(ii) whenever $F \overset{a,m_1}{\rightarrow} F'$ then, for some l_1, E', $E \overset{a,l_1}{\rightarrow} E'$ and $(E', F') \in \mathcal{S}$ and

- if $E \overset{b,l_2}{\rightarrow} E''$ such that $l_1 = l_2$ then, for some F'' and m_2 with $m_1 = m_2$, $F \overset{b,m_2}{\rightarrow} F''$ and
- if $E \overset{b,l_2}{\rightarrow} E''$ such that $l_1 \sqsubset l_2$ then, for some F'' and m_2 with $m_1 \sqsubset m_2$,$F \overset{b,m_2}{\rightarrow} F''$

□

Definition 3.9
E and F are very strong discrete time equivalent, written $E \overset{v}{\sim} F$, if $(E, F) \in \mathcal{S}$ for some strong discrete time bisimulation $\mathcal{S}$. So, $\overset{v}{\sim} = \bigcup \{\mathcal{S} \mid \mathcal{S}$ is a very strong discrete time bisimulation $\}$.

□

3.7.1 Some Properties of Very Strong Discrete Time Equivalence

In this subsection we will give a number of properties of very strong discrete time equivalence. The proofs of these properties can be found in A. Most of these proofs are inspired by proofs given in [Mil89].

Lemma 3.1
If $E \overset{a,l}{\rightarrow} E'$ and $E \overset{b,m}{\rightarrow} E''$ then $\#l = \#m = \#E$.

□

Lemma 3.2
If $E \overset{a,l}{\rightarrow} E'$ then $l \neq 0^{\#E}$.

□

Proposition 3.11
(1) $\overset{v}{\sim}$ is the largest very strong discrete time bisimulation.

(2) $\overset{v}{\sim}$ is an equivalence relation.

□

Proposition 3.12
$E \overset{v}{\sim} F$ iff, for all $a \in \mathcal{A}$,

(i) whenever $E \overset{a,l_1}{\rightarrow} E'$ then, for some m_1, F', $F \overset{a,m_1}{\rightarrow} F'$ and $E' \overset{v}{\sim} F'$ and

- If $F \overset{b,m_2}{\rightarrow} F''$ such that $m_1 = m_2$ then, for some E'' and l_2 with $l_1 = l_2$, $E \overset{b,l_2}{\rightarrow} E''$ and
- If $F \overset{b,m_2}{\rightarrow} F''$ such that $m_1 \sqsubset m_2$ then, for some E'' and l_2 with $l_1 \sqsubset l_2$, $E \overset{b,l_2}{\rightarrow} E''$

(ii) whenever $F \overset{a,m_1}{\rightarrow} F'$ then, for some l_1, E', $E \overset{a,l_1}{\rightarrow} E'$ and $E' \overset{v}{\sim} F'$ and

- If $E \overset{b,l_2}{\rightarrow} E''$ such that $l_1 = l_2$ then, for some F'' and m_2 with $m_1 = m_2$, $F \overset{b,m_2}{\rightarrow} F''$ and
- If $E \overset{b,l_2}{\rightarrow} E''$ such that $l_1 \sqsubset l_2$ then, for some F'' and m_2 with $m_1 \sqsubset m_2$, $F \overset{b,m_2}{\rightarrow} F''$

□

Proposition 3.13
(1) $\sum_{i \in I} a_i \cdot A_i \overset{v}{\sim} \sum_{j \in J} b_j \cdot B_j$ if there exists a bijective function $\nu : I \rightarrow J$ such that $a_i = b_{\nu(i)}$ and $A_i \equiv B_{\nu(i)}$ for all $i \in I$

(2) $\sum_{i \in I \cup \{k\}} a_i \cdot A_i \overset{v}{\sim} \sum_{i \in I} a_i \cdot A_i$ if $k \notin I$ and if there exists a $j \in I$ such that $a_j = a_k$ and $A_j \equiv A_k$

(3) $E \mid F \overset{v}{\sim} F \mid E$

(4) $(E \mid F) \mid G \overset{v}{\sim} (E \mid (F \mid G)$

□

Proposition 3.14
$\overset{v}{\sim}$ is a congruence relation: it is substitutive under all combinators, i.e., if $E_1 \overset{v}{\sim} E_2$ then

(1) $E_1 \mid F \overset{v}{\sim} E_2 \mid F$

(2) $E_1 \setminus L \overset{v}{\sim} E_2 \setminus L$

(3) $E_1[f] \overset{v}{\sim} E_2[f]$

(4) $\mathcal{M}(E_1) \overset{v}{\sim} \mathcal{M}(E_2)$

□

Proposition 3.15
The discrete time expansion law. Let $\phi \in \Phi$ be an expression-naming function. If we define for all $E \in \mathcal{E}$

$$\phi(\mathcal{M}(E)) \stackrel{def}{=} \sum \left\{ a \cdot \phi(\mathcal{M}(E')) \mid E \stackrel{a,l}{\rightarrow} E' \wedge (\neg\, \exists\, b, m, F : E \stackrel{b,m}{\rightarrow} F : l \sqsubset m) \right\}$$

then for all agent expressions E

$$\mathcal{M}(E) \stackrel{v}{\sim} \phi(\mathcal{M}(E))$$

□

In practice, the expansion law is often applied to systems of the form $(A_1[f_1] \mid \ldots \mid A_n[f_n]) \setminus L$. Then it is convenient to have a law which makes it easy to calculate all transitions the systems can make. Such a law is given in Proposition 3.16.

Proposition 3.16
Let $A_1, \ldots, A_n$, for $n \geq 1$, be sequential agent constants, let $f_1, \ldots, f_n$ be relabeling functions, and let L be a set of actions. Then $(A_1[f_1] \mid \ldots \mid A_n[f_n]) \setminus L \stackrel{a,l}{\rightarrow} G$ if and only if $l \in \mathbb{N}^n \setminus \{\mathbf{0}^n\}$ and there exist atoms $a_1, \ldots, a_n$ and agent constants $A'_1, \ldots, A'_n$ such that $a \in PS([\![f_1(a_1), \ldots, f_n(a_n)]\!])$ with $a \cap (L \cup \overline{L}) = \varnothing$, $G \equiv (A'_1[f_1] \mid \ldots \mid A'_n[f_n]) \setminus L$ and

$$\begin{array}{lcl}
A_1 & \stackrel{a_1,(l_1)}{\rightarrow} & A'_1 \text{ or} (l_1 = 0 \wedge A'_1 \equiv A_1 \wedge a_1 = [\![]\!]) \\
A_2 & \stackrel{a_2,(l_2)}{\rightarrow} & A'_2 \text{ or} (l_2 = 0 \wedge A'_2 \equiv A_2 \wedge a_2 = [\![]\!]) \\
\vdots & \vdots & \vdots \\
A_n & \stackrel{a_n,(l_n)}{\rightarrow} & A'_n \text{ or} (l_n = 0 \wedge A'_n \equiv A_n \wedge a_n = [\![]\!])
\end{array}$$

□

3.7.2 Discrete Time Expansion Example

In this subsection we will apply the discrete time expansion law to the example of the two 1-place buffers in Section 3.1. In dtCCS these buffers are specified as

$$\mathcal{M}\left(\left(\left(\dot{BufA}_0 \mid (BufB_0\,[outA/inB])\right) \setminus \{outA\}\right)[outA/outB]\right)$$

where

$$\begin{array}{lcl}
BufA_0 & \stackrel{def}{=} & [\![inA]\!] \cdot BufA_1 \\
BufA_1 & \stackrel{def}{=} & [\![\overline{outA}]\!] \cdot BufA_0
\end{array}$$

$$\begin{array}{lcl}
BufB_0 & \stackrel{def}{=} & [\![inB]\!] \cdot BufB_1 \\
BufB_1 & \stackrel{def}{=} & [\![\overline{outB}]\!] \cdot BufB_0
\end{array}$$

Let ϕ be an expression-naming function which satisfies for $i,j = 0,1$.

$$\phi\left(\mathcal{M}\left(\left(\left(BufA_i \mid (BufB_j\,[outA/inB])\right) \setminus \{outA\}\right)[outA/outB]\right)\right) = BufA_i \mid BufB_j$$

By the expansion law we now have

$$\mathcal{M}\left(\left(\left(BufA_i \mid (BufB_j\,[outA/inB])\right) \setminus \{outA\}\right)[outA/outB]\right) \overset{v}{\sim} BufA_i \mid BufB_j$$

for $i,j = 0,1$ where

$$\begin{aligned} &BufA_i \mid BufB_j \overset{def}{=} \\ &\sum\Big\{a \cdot \phi(\mathcal{M}(E')) \mid \left(\left(BufA_i \mid (BufB_j\,[outA/inB])\right) \setminus \{outA\}\right)[outA/outB] \overset{a,l}{\rightarrow} E' \\ &\qquad\qquad \wedge\,(\neg\ \exists\, b,m,F : E \overset{b,m}{\rightarrow} F : l \sqsubset m)\Big\} \end{aligned}$$

To make these latter definitions more explicit, we have to calculate all derivations of

$$\left(\left(BufA_i \mid (BufB_j\,[outA/inB])\right) \setminus \{outA\}\right)[outA/outB]$$

for $i,j = 0,1$. We shall do this only for the case $i = 0$ and $j = 1$. Using Proposition 3.16 and inference rule **dtRel** we get

$$\begin{aligned} &\left(\left(BufA_0 \mid (BufB_1\,[outA/inB])\right) \setminus \{outA\}\right)[outA/outB] \\ &\quad \overset{[\![inA]\!],\langle 1,0\rangle}{\rightarrow} \left(\left(BufA_1 \mid (BufB_1\,[outA/inB])\right) \setminus \{outA\}\right)[outA/outB] \\ &\quad \overset{[\![\overline{outA}]\!],\langle 0,2\rangle}{\rightarrow} \left(\left(BufA_0 \mid (BufB_0\,[outA/inB])\right) \setminus \{outA\}\right)[outA/outB] \\ &\quad \overset{[\![inA,\overline{outA}]\!],\langle 1,2\rangle}{\rightarrow} \left(\left(BufA_1 \mid (BufB_0\,[outA/inB])\right) \setminus \{outA\}\right)[outA/outB] \end{aligned}$$

Since the first two transitions are "covered" by the third, because $\langle 1,0\rangle \sqsubset \langle 1,2\rangle$ and $\langle 0,2\rangle \sqsubset \langle 1,2\rangle$, these two transitions have to be discarded under maximal progress. So, we get

$$BufA_0 \mid BufB_1 \overset{def}{=} [\![inA, \overline{outA}]\!] \cdot BufA_1 \mid BufB_0$$

and if we calculate the derivations of the remaining expressions we get

$$\begin{aligned} BufA_0 \mid BufB_0 &\overset{def}{=} [\![inA]\!] \cdot BufA_1 \mid BufB_0 \\ BufA_1 \mid BufB_0 &\overset{def}{=} [\![\,]\!] \cdot BufA_0 \mid BufB_1 \end{aligned}$$

This is the result we were aiming at (see Section 3.1).

3.8 Refining the Notion of Maximal Progress

In many cases specifications of real-time systems make use of timers. A typical specification of a timer in dtCCS is the following:

$$
\begin{array}{lcl}
Timer & \stackrel{def}{=} & [\![trigger]\!] \cdot TimerSet_1 \\
TimerSet_1 & \stackrel{def}{=} & [\![trigger]\!] \cdot TimerSet_1 + [\![\,]\!] \cdot TimerSet_2 \\
TimerSet_2 & \stackrel{def}{=} & [\![trigger]\!] \cdot TimerSet_1 + [\![\,]\!] \cdot TimerSet_3 \\
\vdots & \vdots & \vdots \\
TimerSet_n & \stackrel{def}{=} & [\![trigger]\!] \cdot TimerSet_1 + \overline{timeout} \cdot Timer
\end{array}
$$

After the timer is 'triggered' it can produce a time-out after precisely n time units. If the timer is in some state $TimerSet_i$, with $i = 1, \cdots, n-1$, it can be reset, after which it returns to state $TimerSet_1$, or it can proceed to state $TimerSet_{i+1}$. In state $TimerSet_n$ the timer can either be reset or produce a time-out.

Now assume the timer is used by some agent A defined as

$$A \stackrel{def}{=} [\![\overline{trigger}]\!] \cdot B$$

and suppose the timer is in one of its states $TimerSet_i$ for $i = 1, \cdots, n-1$. Further, assume that the agents proceed in maximal progress. One would expect that during the next timeslot the timer would receive the trigger atom, after which it returned to state $TimerSet_1$. However, the expansion law stated in Proposition 3.15 allowes the timer to take the $[\![\,]\!]$-transition to proceed after which it proceeds to state $TimerSet_n$! It is not possible to 'force' the timer into receiving the offered *trigger* atom, and therefore it is not really possible to specify the true behaviour of systems which use timers.

This problem can be overcome by slightly modifying our previous notion of maximal progress. Our modified notion of maximal progress is as follows: If a number of agents simultaneously perform one of their transitions, and if there is not another agent which could also participate by performing one of its transitions, even not if they choose to perform different transitions, the system has maximal progress.

Formally, this modification boils down to adapting the ordering relation $\sqsubseteq$ on lists defined in Definition 3.4 of Section 3.5. We will denote the adapted relation by $\sqsubseteq'$.

Definition 3.10
Let $l, m \in \mathbb{N}^*$ be lists over $\mathbb{N}$. $l \sqsubseteq' m$ if and only if

- $\#l = \#m$
- $\forall i : i = 1, \cdots, \#l : l_i \not= 0 \Rightarrow m_i \neq 0$

□

We will write $l \sqsubset' m$ if $l \sqsubseteq' m$ and $l \neq' m$. Here, $='$ is different from the traditional list equality $=$. The definition is as follows:

Definition 3.11
Let $l, m \in \mathbb{N}^*$ be lists over $\mathbb{N}$. $l =' m$ if and only if

- $\#l = \#m$
- $\forall i : i = 1, \cdots, \#l : l_i = 0 \Leftrightarrow m_i = 0$

□

It can be proved that all propositions of Chapter 3 remain valid if we replace $\sqsubseteq$, $\sqsubset$ and $=$ by $\sqsubseteq'$, $\sqsubset'$ and $='$ respectively in Definition 3.8 and Proposition 3.12. From now on we will explicitly state which notion of list equivalence we use and thus which notion of maximal progress we apply.

Note that the new notion of maximal progress solves the problem stated at the beginning of this section. If agent A wants to *trigger* the timer, this timer cannot choose to perform the $[\![]$-transition, for if it chooses the other transition, better overall progress is established.

3.9 Multi Time-Step Atoms

As mentioned in Section 3.2, we will assume that all atoms take a *single* time-step to execute. In this section we introduce a way to denote *multi* time-step atoms. Since this notation is not supported by the semantics of dtCCS, we will also present a way to express such atoms in dtCCS.

We will write

$$a < n >$$

to denote that atom a takes n time steps to execute.

We can express these multi time-step atoms is dtCCS by mapping them onto sequences of single time-step atoms. There are several ways to do this, among others the following:

- $a_1 \cdot a_2 \cdot \cdots \cdot a_{n-1} \cdot a_n$, i.e., indicate each partial single time-step atom explicitly
- $a_1 \cdot [\![]\!] \cdot \cdots \cdot [\![]\!] \cdot a_n$, i.e., indicate the first and last partial single time-step atom explicitly, and represent the intermediate partial single time-step atoms as empty atoms
- $a \cdot [\![]\!] \cdot \cdots \cdot [\![]\!]$, i.e., indicate only the first partial single time-step atom explicitly, and represent the other partial single time-step atoms as empty atoms

Although either of these three mapping methods could have been used, we have chosen to use the third method.

As an example, the agent expression

$$A \stackrel{def}{=} [\![in]\!] < 3 > \cdot B$$

will be mapped onto the following sequence of agent expression:

$$\begin{aligned} A &\stackrel{def}{=} [\![in]\!] \cdot A[1] \\ A[1] &\stackrel{def}{=} [\![]\!] \cdot A[2] \\ A[2] &\stackrel{def}{=} [\![]\!] \cdot B \end{aligned}$$

where $A[i]$ for $i = 1, 2$ are fresh agent constants. Of course, the resulting behaviour is independent of the choice of these fresh agent constants.

Chapter 4

Performance-Analysis Model

In Chapter 3 we introduced the process calculus dtCCS (discrete time CCS). Using dtCCS we can describe both the *time* behaviour and the *functional* behaviour of systems of communicating agents. Agents in dtCCS reveal their activities by performing so-called atoms. Every atom consists of zero or more actions which are performed simultaneously. The performance of an atom takes a single time-step. A very useful figure for the performance analysis of an agent A, given some action a, is the average number of a actions performed per time step. We will call this number the *mean a-performance* of agent A.

As an example, consider the 1-place buffer $BufA_0$ of the example in Subsection 3.7.2 specified as

$$BufA_0 \stackrel{def}{=} [\![inA]\!] \cdot BufA_1$$
$$BufA_1 \stackrel{def}{=} [\![\overline{outA}]\!] \cdot BufA_0$$

The performance of this buffer can be given in terms of the average number of input actions inA which can be performed per time step. In this case it is easy to see that the mean inA-performance equals a half; every two time steps one input action is performed. In general, however, for some arbitrary agent A and some arbitrary action a, it can be quite complicated to calculate the mean a-performance. In this chapter we present a method to calculate such performance figures of systems of communicating agents described in dtCCS, using a *stochastic performance-analysis model.*

Section 4.1 introduces a method to calculate so-called equilibrium probabilities of processes with single time-step states. Section 4.2 generalizes this calculation method to processes with multi time-step states. Section 4.3 gives an example of the application of this method to calculate the mean-performance figures of a system of communicating agents described in dtCCS. Section 4.4 generalizes this application. Section 4.5 describes two transformations of behaviour equations of dtCCS agents to reduce the computational complexity of the calculation method.

4.1 A Result from the Markov Theory

Assume we have some process with a certain state space $S = \{1, 2, \ldots, n\}$. At any time the process is in exactly one of its states, and it will stay there for a fixed time, say one time step. Further, at certain moments in time, it can move from one state to another. If, at a certain moment, the process is in state i, we let p_{ij} denote the probability that its next state is j.

The most convenient way to describe such a process is by a so-called *transition-probability matrix* P corresponding to a so-called *discrete-time stationary Markov chain* [IM76]. Associate the i^{th} row and column of P with the i^{th} state of S and the matrix takes the form

$$P = \begin{pmatrix} p_{11} & p_{12} & \cdots & p_{1n} \\ p_{21} & p_{22} & \cdots & p_{2n} \\ \vdots & \vdots & & \vdots \\ p_{n1} & p_{n2} & \cdots & p_{nn} \end{pmatrix}$$

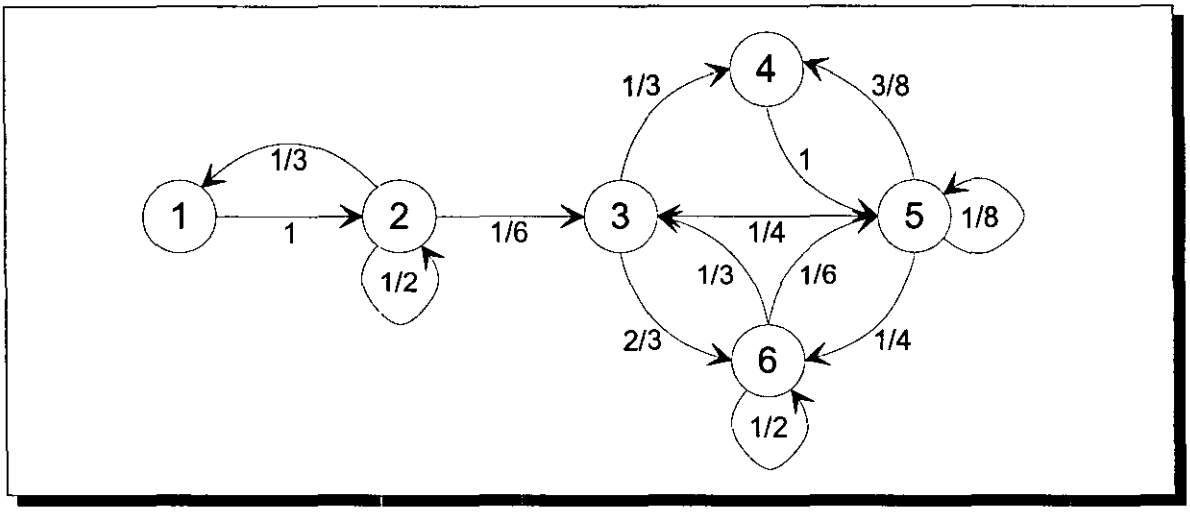

Figure 4.1: Directed graph visualizing a transition-probability matrix

A process described by some transition-probability matrix P can be visualized by a directed graph. For example, figure 4.1 shows a directed graph visualizing a process with six states ($S = \{1, 2, 3, 4, 5, 6\}$) with a matrix P be given by

$$P = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ \frac{1}{3} & \frac{1}{2} & \frac{1}{6} & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{1}{3} & 0 & \frac{2}{3} \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & \frac{1}{4} & \frac{3}{8} & \frac{1}{8} & \frac{1}{4} \\ 0 & 0 & \frac{1}{3} & 0 & \frac{1}{6} & \frac{1}{2} \end{pmatrix}$$

At any time a process described by P is in one of its six states. Given that the process is in equilibrium, let π_i denote the probability that the process is in state i at a certain moment. Using some elementary Markov theory we are able to calculate these so-called *equilibrium probabilities.*

Let P be a transition-probability matrix of some process with state space $S = \{1, 2, \ldots, n\}$. A subset C of S is called *closed* if $p_{ik} = 0$ for all $i \in C$ and $k \notin C$. That is, if once the process enters C, it can never leave C. A *path* in S is a sequence of states $(s_1, s_2, \ldots, s_m)$ for some $m \geq 1$ such that $p_{s_i s_{i+1}} > 0$ for all $i = 1, 2, \ldots, m-1$. A subset C of S is called *irreducible* if for every two states s and t in C there exists a path from s to t as well as a path from t to s.

Proposition 4.1
Let P be a transition-probability matrix of some process with state space $S = \{1, 2, \ldots, n\}$, such that S has precisely one closed irreducible subset C and that for every state s outside C there exists a path from s to some state t in C. Then we can uniquely calculate the equilibrium probabilities π_i for $i = 1, 2, \ldots, n$ by solving the following system of equations:

$$\begin{cases} \pi_j = \sum_{i=1}^{n} \pi_i p_{ij} \text{ for } j = 1, 2, \ldots, n \\ \sum_{j=1}^{n} \pi_j = 1 \end{cases}$$

□

The state space of the example given in figure 4.1 can be split into two disjunct sets $\{1, 2\}$ and $\{3, 4, 5, 6\}$. It is easy to see that for all states of the former set there exists a path to the latter set which is closed and irreducible. Therefore, we can apply Proposition 4.1 and determine the probabilities π_i by solving

$$\begin{cases} & \frac{1}{3}\pi_2 & & & & & & & & = & \pi_1 \\ \pi_1 + & \frac{1}{2}\pi_2 & & & & & & & & = & \pi_2 \\ & \frac{1}{6}\pi_2 & & & & + & \frac{1}{4}\pi_5 + & \frac{1}{3}\pi_6 & = & \pi_3 \\ & & \frac{1}{3}\pi_3 & & & + & \frac{3}{8}\pi_5 & & = & \pi_4 \\ & & & \pi_4 & + & \frac{1}{8}\pi_5 + & & \frac{1}{6}\pi_6 & = & \pi_5 \\ & & \frac{2}{3}\pi_3 & & & + & \frac{1}{4}\pi_5 + & \frac{1}{2}\pi_6 & = & \pi_6 \\ \pi_1 + \pi_2 + & \pi_3 + & \pi_4 & + & \pi_5 & + & \pi_6 & & = & 1 \end{cases}$$

The solution of this system is given by $(\pi_1, \pi_2, \pi_3, \pi_4, \pi_5, \pi_6) = (0, 0, \frac{6}{31}, \frac{5}{31}, \frac{8}{31}, \frac{12}{31})$. So, if the process is in equilibrium, the probability that the process is, for example, in state 4

equals $\frac{5}{31}$. Further, the probability that it is in state 1 or 2 equals zero. In general, this latter is true for those states for which there exists a path to the closed irreducible set of states and which themselves do not belong to this set.

4.2 Generalizing the Result

Using Proposition 4.1 given in the previous section we can calculate the equilibrium probabilities of a restricted class of processes. One of the assumptions was that a process remains in some state for exactly one time step. We will now drop this assumption and look at processes which remain for exactly τ_i time steps in state i. Such a process is given by a triple $\langle S, P, \tau \rangle$ consisting of a state space S, a transition-probability matrix P, and a time vector $\tau = (\tau_1, \tau_2, \ldots, \tau_n)$ with $\tau_i = 1, 2, \ldots$ for all $i = 1, 2, \ldots, n$.

Before we can give a method to calculate the equilibrium probabilities, we first have to explain what we mean by such a process. We will interpret a process defined by $\langle S, P, \tau \rangle$ as a process with a state space $S' = \{1, 2, \ldots, \tau_1 + \tau_2 + \cdots + \tau_n\}$ which we will denote as $\{s_{11}, s_{12}, \ldots, s_{1t_1}, s_{21}, s_{22}, \ldots, s_{2t_2}, \ldots, s_{n1}, s_{n2}, \ldots, s_{nt_n}\}$, and with a transition-probability matrix P' defined as

$$\begin{cases} p'_{s_{it_i}s_{j1}} = p_{ij} & \text{for all } i, j \in S \\ p'_{s_{ik}s_{ik+1}} = 1 & \text{for all } i \in S \text{ and } 1 \leq k < \tau_i \\ p'_{s_{ik}s_{jl}} = 0 & \text{otherwise} \end{cases}$$

This means that we consider a state i to be decomposed into τ_i consecutive states as indicated in figure 4.2.

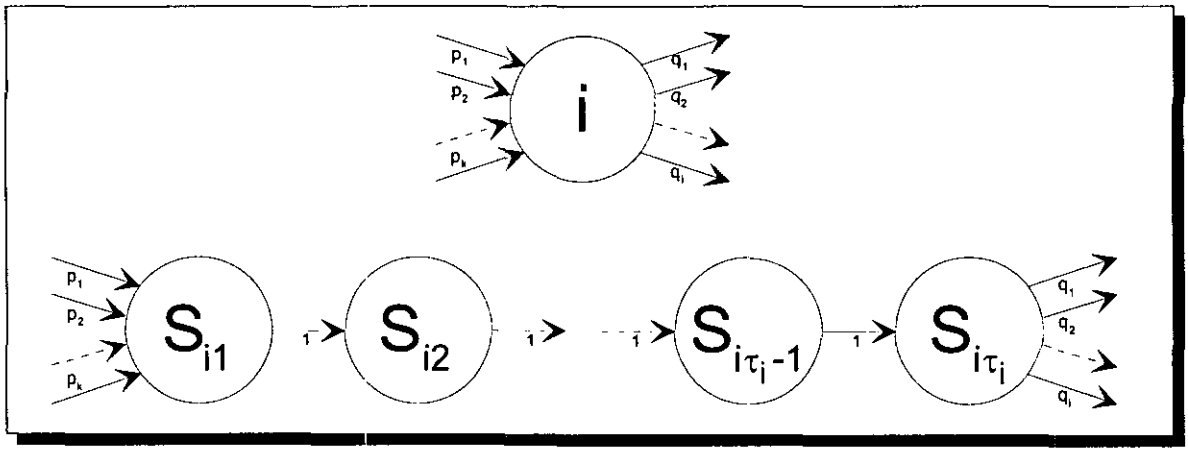

Figure 4.2: State decomposition

The probability that the process represented by matrix P is in state i, written Π_i, equals the probability that process represented by matrix P' is in one of its τ_i corresponding states, i.e.,

$$\Pi_i = \pi_{s_{i1}} + \pi_{s_{i2}} + \cdots + \pi_{s_{i\tau_i}}$$

The system of equations corresponding to matrix P' is

$$\begin{cases} \pi_{s_{j1}} = \sum_{i=1}^{n} \pi_{s_{i\tau_i}} p'_{s_{i\tau_i} s_{j1}} & \text{for } j = 1, 2, \ldots, n \\ \pi_{s_{ik+1}} = \pi_{s_{ik}} & \text{for } i \in S, 1 \leq k \leq \tau_i \\ s_{11} + s_{12} + \cdots + s_{1t_1} + s_{21} + s_{22} + \cdots + s_{2t_2} + \cdots + s_{n1} + s_{n2} + \cdots + s_{nt_n} = 1 \end{cases}$$

Since $\Pi_i = \pi_{s_{i1}} + \pi_{s_{i2}} + \cdots + \pi_{s_{i\tau_i}}$ and because $\pi_{s_{i\tau_i}} = \pi_{s_{i\tau_i - 1}} = \cdots = \pi_{s_{i1}}$, we have $\pi_{s_{j1}} = \pi_{s_{j\tau_j}} = \frac{\Pi_j}{\tau_j}$ for all $j \in S$. Further, $p'_{s_{i\tau_i} s_{j1}} = p_{ij}$ for all $i, j \in S$. So, the system of equations for P' can be transformed into

$$\begin{cases} \frac{\Pi_j}{\tau_j} = \sum_{i=1}^{n} \frac{\Pi_i}{\tau_i} p_{ij} & \text{for } j = 1, 2, \ldots, n \\ \sum_{j=1}^{n} \Pi_j = 1 \end{cases}$$

This result leads us to

Proposition 4.2
Let P be a transition-probability matrix of a process with state space $S = \{1, 2, \ldots, n\}$ and time vector τ, such that S has precisely one closed irreducible subset C and that for every state s outside C there exists a path to a state t inside C. Then we can uniquely calculate the probabilities Π_i for $i = 1, 2, \ldots, n$ by solving the following system of equations:

$$\begin{cases} \Pi_j = \tau_j \sum_{i=1}^{n} \Pi_i \frac{p_{ij}}{\tau_i} & \text{for } j = 1, 2, \ldots, n \\ \sum_{j=1}^{n} \Pi_j = 1 \end{cases}$$

□

4.3 Applying the Result to dtCCS: An Example

Proposition 4.2 stated in the previous section gives us a method to calculate the equilibrium probabilities of a restricted class of processes which remain for exactly τ_i time steps in state i. In this section we will show how we can apply this proposition to calculate the mean-performance figures of a system of communicating agents described in dtCCS.

Consider a dtCCS agent A defined as

$$
\begin{aligned}
A &\stackrel{def}{=} [\![a]\!] \cdot A + [\![b,c]\!] \cdot B \\
B &\stackrel{def}{=} [\![a]\!] \cdot A + [\![b]\!] \cdot C \\
C &\stackrel{def}{=} [\![c]\!] \cdot D \\
D &\stackrel{def}{=} [\![b]\!] \cdot B
\end{aligned}
$$

and suppose we want to calculate the mean-performance figures of actions a and c of A.

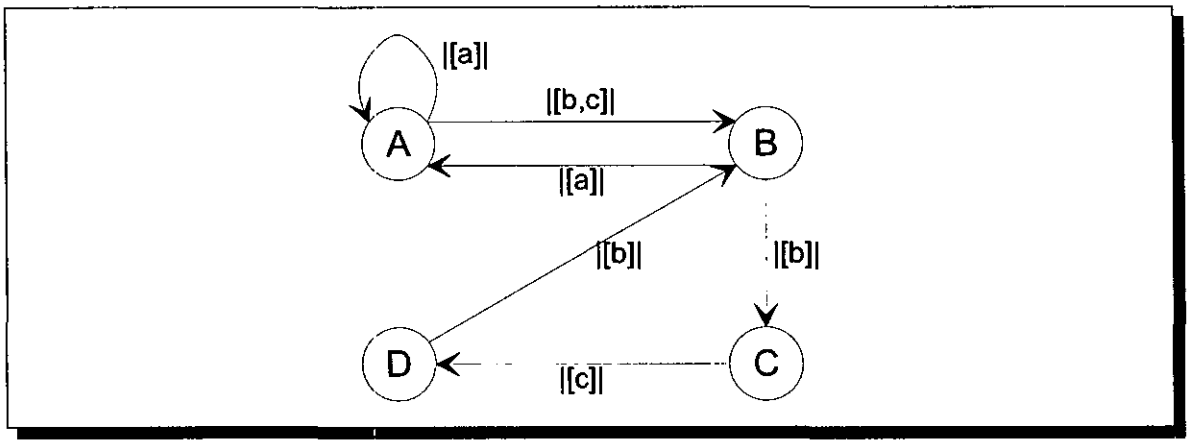

Figure 4.3: Transition graph representing a dtCCS agent

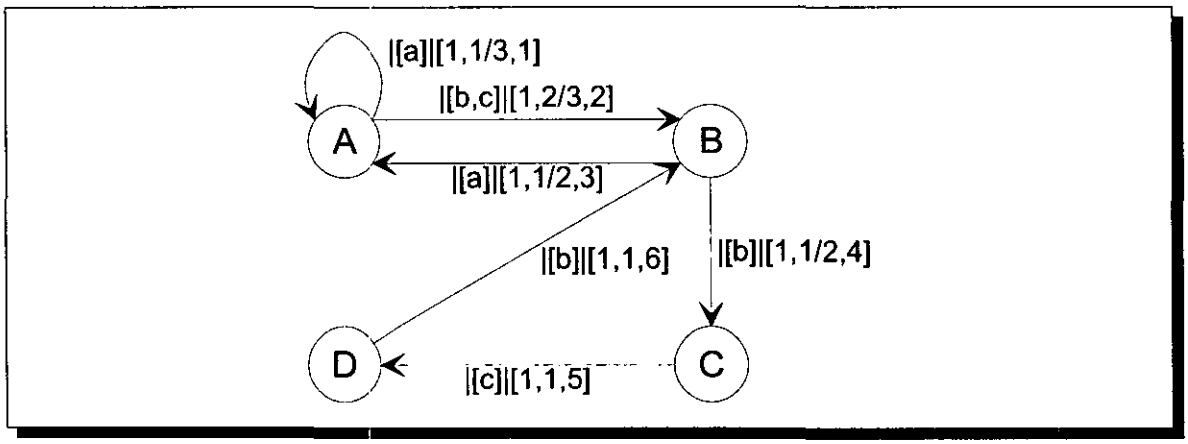

Figure 4.4: Extended transition graph representing a dtCCS agent

First, we represent A by a so-called *transition graph* as the one shown in figure 4.3. The sequential agent constants are interpreted as the *states* of the graph. The agent transitions

are represented by the labeled arrows. Next, we add time durations, probabilities and sequence numbers to every transition of the graph. Since the performance of an atom takes a single time-step, the time durations all equal 1. The transition probabilities are not derivable from the agent definition and hence they represent additional information. The used transition sequence numbers are allocated in such a way that different transitions get different numbers. A possible resulting graph is shown in figure 4.4.

Since we are only interested in transition-time durations, transition probabilities, and transition numbers, we can also represent this graph as indicated in figure 4.5 (we will show this later). Here, for example, expression $[\![b]\!] \cdot [\![c]\!] \cdot [\![b]\!][3, \frac{1}{2}, 4]$ indicates that if the process is in state B, it has a probability of $\frac{1}{2}$ that it will perform consecutively atoms $[\![b]\!]$, $[\![c]\!]$, and $[\![b]\!]$ of transition 4, which takes 3 time steps.

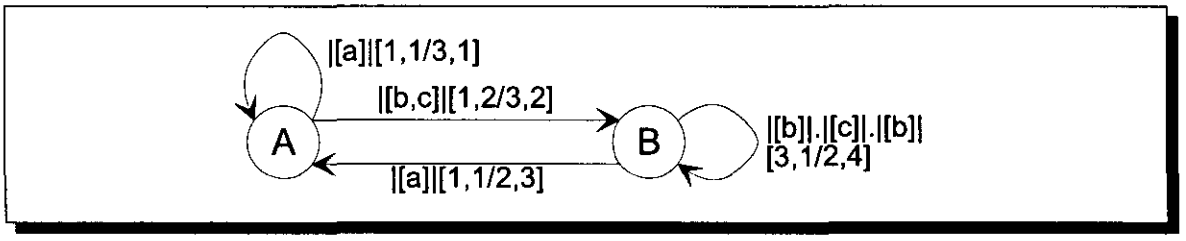

Figure 4.5: Denormalized extended transition graph representing a dtCCS agent

The graph of figure 4.5 can also be represented textually as follows:

$$A \stackrel{def}{=} [\![a]\!][1,1,1] \cdot A \; + \; [\![b,c]\!][1,2,2] \cdot B$$
$$B \stackrel{def}{=} [\![a]\!][1,1,3] \cdot A \; + \; [\![b]\!] \cdot [\![c]\!] \cdot [\![b]\!][3,1,4] \cdot B$$

Here, we have written transition *weights* instead of transition *probabilities*. Of course, given the textual representation, it is straightforward to reconstruct the graph of figure 4.5.

To calculate the mean-performance figures of agent A, we transform the graph of figure 4.5 into a Markov process $\langle S, P, \tau \rangle$ by representing transitions as states. We define $S = \{1,2,3,4\}$. The time duration of state i equals the time duration of transition i, so $\tau = (1,1,1,3)$. The transition-probability matrix P is determined in the following way: Assume that agent A is performing transition 2. Then the probability that the next performed transition is 4 equals $\frac{1}{2}$. Because transitions are uniquely identified with states of the Markov process, we have $p_{24} = \frac{1}{2}$. In a similar way we determine the other probabilities and get

$$P = \begin{pmatrix} \frac{1}{3} & \frac{2}{3} & 0 & 0 \\ 0 & 0 & \frac{1}{2} & \frac{1}{2} \\ \frac{1}{3} & \frac{2}{3} & 0 & 0 \\ 0 & 0 & \frac{1}{2} & \frac{1}{2} \end{pmatrix}$$

The resulting Markov process is visualized in figure 4.6.

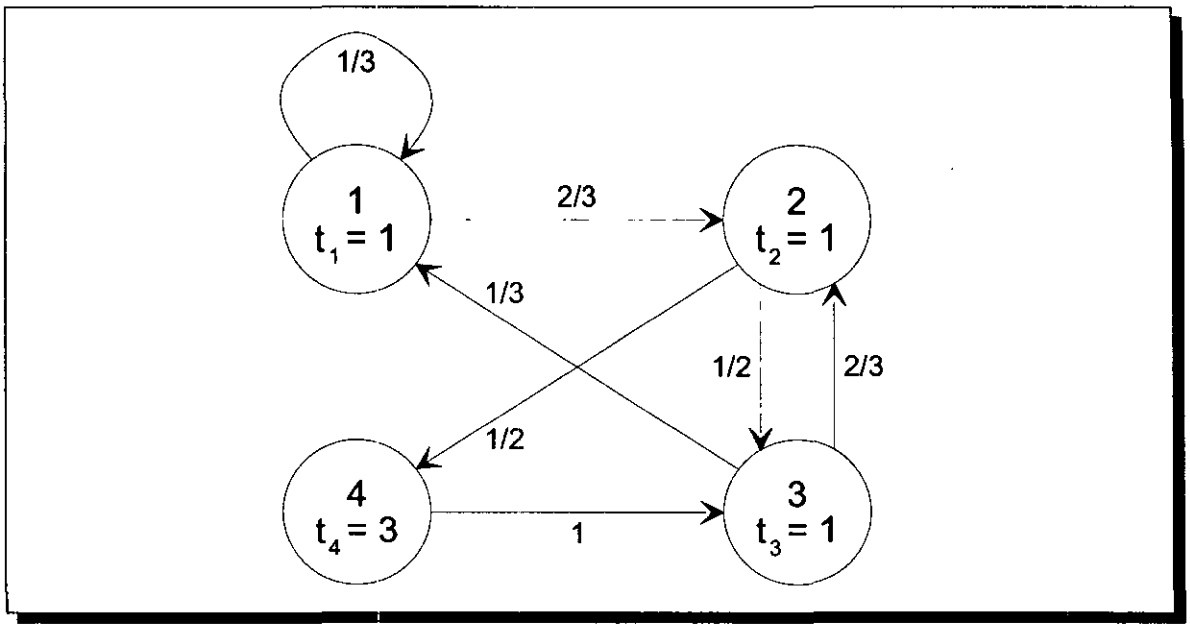

Figure 4.6: Markov process representing a dtCCS agent

Note that the Markov process satisfies the conditions of Proposition 4.2. Therefore, we can calculate the equilibrium probabilities by solving

$$\begin{cases} \Pi_1 = 1\frac{1}{3}(\Pi_1\frac{1}{1} + \Pi_3\frac{1}{1}) \\ \Pi_2 = 1\frac{2}{3}(\Pi_1\frac{1}{1} + \Pi_3\frac{1}{1}) \\ \Pi_3 = 1\frac{1}{2}(\Pi_2\frac{1}{1} + \Pi_4\frac{1}{3}) \\ \Pi_4 = 3\frac{1}{2}(\Pi_2\frac{1}{1} + \Pi_4\frac{1}{3}) \\ 1 = \Pi_1 + \Pi_2 + \Pi_3 + \Pi_4 \end{cases}$$

If we now look at figure 4.5 again, we see that the equation for probability Π_i equals the time duration τ_i of transition i times the probability p_i of transition i times the sum of all $\frac{\Pi_j}{\tau_j}$ of transitions j for which the destination state equals the source state of transition i. The solution is $(\Pi_1, \Pi_2, \Pi_3, \Pi_4) = (\frac{1}{11}, \frac{2}{11}, \frac{2}{11}, \frac{6}{11})$.

So, agent A is in average $\frac{1}{11}$ of the time in transition 1, and $\frac{2}{11}$ of the time in transition 3. This means that A spends $\frac{2}{11} + \frac{1}{11} = \frac{3}{11}$ of its time performing a actions. So, A performs $\frac{3}{11}$ a action during a single time-step, and hence the mean a-performance of A is $\frac{3}{11}$. Further, A spends $\frac{2}{11}$ of the time in transition 2 performing the corresponding c action. A also spends $\frac{6}{11}$ of the time in transition 4, of which $\frac{1}{3}$ is used to perform a c action. This means that the mean c-performance of A equals $\frac{2}{11} + \frac{1}{3}\frac{6}{11} = \frac{4}{11}$.

4.4 Applying the Result to dtCCS: The General Case

Let us now consider some general dtCCS agent A_1 to which time information, probability (or weight) information, and transition-sequence numbers have been added. Following the textual representation of figure 4.5, we can represent A_1 by the following system of equations:

$$\begin{array}{lcl} A_1 & \stackrel{def}{=} & x_{11}[\tau_{11}, w_{11}, s_{11}] \cdot A_{11} + \cdots + x_{1n_1}[\tau_{1n_1}, w_{1n_1}, s_{1n_1}] \cdot A_{1n_1} \\ A_2 & \stackrel{def}{=} & x_{21}[\tau_{21}, w_{21}, s_{21}] \cdot A_{21} + \cdots + x_{2n_2}[\tau_{2n_2}, w_{2n_2}, s_{2n_2}] \cdot A_{2n_2} \\ \vdots & \vdots & \vdots \\ A_m & \stackrel{def}{=} & x_{m1}[\tau_{m1}, w_{m1}, s_{m1}] \cdot A_{m1} + \cdots + x_{mn_m}[\tau_{mn_m}, w_{mn_m}, s_{mn_m}] \cdot A_{mn_m} \end{array}$$

Here, x_{ij} denotes a sequence of atoms with at least one element. It can be written as $a_1 \cdot a_2 \cdot \cdots \cdot a_k$ for some atoms $a_1, \ldots, a_k$ and for some $k \geq 1$. τ_{ij} is the time it takes to perform the transition, w_{ij} is the (probability) weight, and s_{ij} is the sequence number of the transition. We require that the specification satisfies the following conditions:

(1) $m \geq 1$ and $n_i \geq 1$ for all $i = 1, \ldots, m$. That is, the system contains at least one equation which contains at least one transition.

(2) For all $i, j = 1, \ldots, m$ if $i \neq j$ then $A_i \not\equiv A_j$. So, the m agent constants A_i are different.

(3) $w_{ij} = 0, 1, 2, \ldots$, $s_{ij} = 1, 2, \ldots$, and $\tau_{ij} = \#x_{ij}$ for all $i = 1, \ldots, m$ and $j = 1, \ldots, n_i$. Thus, the time duration equals the amount of atoms of the transition.

(4) Different transitions have different sequence numbers.

(5) For every $i = 1, \ldots, m$ there exists at least one $j = 1, \ldots, n_i$ such that $w_{ij} > 0$. So, for every state there exists at least one transition which can be performed.

(6) $A_{ij} \in \{A_1, \ldots, A_m\}$ for all $i = 1, \ldots, m$ and $j = 1, \ldots, n_i$. This means that the system of equations is "closed".

(7) If the agent is mapped onto a Markov process by interpreting transition numbers as states, the state space S of the resulting Markov process should have precisely one closed irreducible subset C and for all states outside C there should be a path to a state inside C.

Condition (5) allows us to define transition probabilities

$$p_{ij} = \frac{w_{ij}}{\sum_{k=1}^{n_i} w_{ik}}$$

for all $i = 1, \ldots, m$ and $j = 1, \ldots, n_i$. Further, given the agent is in equilibrium, we let $\Pi_{s_{ij}}$ denote the probability that the transition with number s_{ij} is being performed.

In the example of Section 4.3 we calculated the equilibrium probabilities by first transforming the agent into a Markov process $\langle S, P, \tau \rangle$ which we used to derive a system of equations using Proposition 4.2. In this case we will not make this intermediate step, and directly give the following equations:

$$\begin{cases} \Pi_{s_{ij}} &= \tau_{ij} p_{ij} \sum \left\{ \frac{\Pi_{s_{kl}}}{\tau_{kl}} \mid k = 1, \ldots, m,\ l = 1, \ldots, n_k,\ s_{kl} \rightsquigarrow s_{ij} \right\} \\ 1 &= \sum \left\{ \Pi_{s_{ij}} \mid i = 1, \ldots, m,\ j = 1, \ldots, n_i \right\} \end{cases}$$

Here, $s_{kl} \rightsquigarrow s_{ij}$ means that the destination state of transition s_{kl} equals the source state of transition s_{ij}.

Note that Proposition 4.2 can only be applied, if the the state space S of the Markov process has precisely one closed irreducible subset C, and if for all states outside C there exists a path to a state inside C. Condition (7) guarantees that this is indeed true.

Let us now fix some action a and calculate the mean a-performance of agent A_1. If s_{ij} is the number of some transition, x_{ij} denotes the sequence of atoms performed when the transition is made. The performance of a single atom involves the performance of possibly several actions. We define $N(a, x)$ as the amount of a actions included in x. Then the execution of transition s_{ij} involves the performance of $N(a, x_{ij})$ a actions. Agent A_1 spends $\Pi_{s_{ij}}$ of the time in transition s_{ij} and this transition takes τ_{ij} time steps. So, in average $\Pi_{s_{ij}} \frac{N(a,x_{ij})}{\tau_{ij}}$ a actions of transition s_{ij} are performed, and the mean a-performance equals

$$\sum \left\{ \Pi_{s_{ij}} \frac{N(a,x_{ij})}{\tau_{ij}} \mid i = 1, \ldots, m,\ j = 1, \ldots, n_i \right\}$$

4.5 Reducing the Computational Complexity

In many practical cases the calculation of some mean-performance figure involves solving systems with a huge amount of equations and variables. We will now describe two transformations on behaviour equations (extended with time information, weight information, and sequence numbers), which reduce this complexity.

Transformation 4.1
Consider the specification of agent A_1

$$\begin{array}{lcl} A_1 & \stackrel{def}{=} & x_{11}[\tau_{11}, w_{11}, s_{11}] \cdot A_{11} + \cdots + x_{1n_1}[\tau_{1n_1}, w_{1n_1}, s_{1n_1}] \cdot A_{1n_1} \\ A_2 & \stackrel{def}{=} & x_{21}[\tau_{21}, w_{21}, s_{21}] \cdot A_{21} + \cdots + x_{2n_2}[\tau_{2n_2}, w_{2n_2}, s_{2n_2}] \cdot A_{2n_2} \\ \vdots & \vdots & \vdots \\ A_m & \stackrel{def}{=} & x_{m1}[\tau_{m1}, w_{m1}, s_{m1}] \cdot A_{m1} + \cdots + x_{mn_m}[\tau_{mn_m}, w_{mn_m}, s_{mn_m}] \cdot A_{mn_m} \end{array}$$

and assume that this specification satisfies conditions (1)–(7) stated in the previous section. Further, suppose that the equation of some state A_k is

$$A_k \stackrel{def}{=} x_{k1}[\tau_{k1}, w_{k1}, s_{k1}] \cdot A_{k1}$$

where $A_{k1} \not\equiv A_k$.

If we remove the equation of state A_k and if we replace every transition of the form $x_{jk}[\tau_{jk}, w_{jk}, s_{jk}] \cdot A_i$ by $x_{jk} \cdot x_{i1}[\tau_{i1} + \tau_{jk}, w_{jk}, s_{jk}] \cdot A_{i1}$, the mean a-performance remains the same for any a.

□

Proof

A sketch of the proof is as follows: Let $\Pi_{s_{ij}}$ denote the equilibrium probabilities of the original agent. Then it is not hard to verify that the equilibrium probabilities $\Pi'_{s_{ij}}$ of the transformed system of equations are given by

$$\Pi_{s_{ij}} = \begin{cases} \frac{\Pi_{s_{ij}}}{\tau_{ij}}(\tau_{ij} + \tau_{k1}) & \text{if } A_i \stackrel{def}{=} \cdots + x_{ij}[\tau_{ij}, w_{ij}, s_{ij}] \cdot A_k + \cdots \\ \Pi_{s_{ij}} & \text{otherwise} \end{cases}$$

for all $i = 1, \ldots, m$, $j = 1, \ldots, n_i$ with $i, j \neq k, 1$.

Let x'_{ij} and τ'_{ij} respectively denote atom sequences and time durations of transitions s_{ij} of the transformed agent, and let a be some action. Then the mean a-performance of the transformed agent is

$$\sum \left\{ \Pi'_{s_{ij}} \frac{N(a, x'_{ij})}{\tau'_{ij}} \;\middle|\; i = 1, \ldots, m,\ j = 1, \ldots, n_i,\ i, j \neq k, 1 \right\}$$

and satisfies

$$\sum \left\{ \Pi'_{s_{ij}} \frac{N(a, x'_{ij})}{\tau'_{ij}} \;\middle|\; i = 1, \ldots, m,\ j = 1, \ldots, n_i,\ i, j \neq k, 1 \right\} =$$

$$\sum \left\{ \Pi'_{s_{ij}} \frac{N(a, x'_{ij})}{\tau'_{ij}} \;\middle|\; i = 1, \ldots, m,\ j = 1, \ldots, n_i,\ i, j \neq k, 1,\ A_i \stackrel{def}{=} \cdots + x_{ij}[\tau_{ij}, w_{ij}, s_{ij}] \cdot A_k + \cdots \right\} +$$

$$\sum \left\{ \Pi'_{s_{ij}} \frac{N(a, x'_{ij})}{\tau'_{ij}} \;\middle|\; i = 1, \ldots, m,\ j = 1, \ldots, n_i,\ i, j \neq k, 1,\ A_i \stackrel{def}{\neq} \cdots + x_{ij}[\tau_{ij}, w_{ij}, s_{ij}] \cdot A_k + \cdots \right\} =$$

$$\sum \left\{ \frac{\Pi_{s_{ij}}}{\tau_{ij}}(\tau_{ij} + \tau_{k1}) \frac{N(a, x_{ij}) + N(a, x_{k1})}{\tau_{ij} + \tau_{k1}} \;\middle|\; i = 1, \ldots, m,\ j = 1, \ldots, n_i,\ i, j \neq k, 1,\ A_i \stackrel{def}{=} \cdots + x_{ij}[\tau_{ij}, w_{ij}, s_{ij}] \cdot A_k + \cdots \right\} +$$

$$\sum \left\{ \Pi_{s_{ij}} \frac{N(a, x_{ij})}{\tau_{ij}} \;\middle|\; i = 1, \ldots, m,\ j = 1, \ldots, n_i,\ i, j \neq k, 1,\ A_i \stackrel{def}{\neq} \cdots + x_{ij}[\tau_{ij}, w_{ij}, s_{ij}] \cdot A_k + \cdots \right\} =$$

$$\sum\left\{\prod_{s_{ij}}\frac{N(a,x_{ij})}{\tau_{ij}} \mid i=1,\ldots,m,\ j=1,\ldots,n_i,\ i,j\neq k,1,\right.$$
$$\left. A_i \stackrel{def}{=} \cdots + x_{ij}[\tau_{ij},w_{ij},s_{ij}]\cdot A_k + \cdots\right\} +$$
$$\sum\left\{\prod_{s_{ij}}\frac{N(a,x_{k1})}{\tau_{ij}} \mid i=1,\ldots,m,\ j=1,\ldots,n_i,\ i,j\neq k,1,\right.$$
$$\left. A_i \stackrel{def}{=} \cdots + x_{ij}[\tau_{ij},w_{ij},s_{ij}]\cdot A_k + \cdots\right\} +$$
$$\sum\left\{\prod_{s_{ij}}\frac{N(a,x_{ij})}{\tau_{ij}} \mid i=1,\ldots,m,\ j=1,\ldots,n_i,\ i,j\neq k,1,\right.$$
$$\left. A_i \stackrel{def}{=} \cdots + x_{ij}[\tau_{ij},w_{ij},s_{ij}]\cdot A_k + \cdots\right\} =$$

$$\sum\left\{\prod_{s_{ij}}\frac{N(a,x_{ij})}{\tau_{ij}} \mid i=1,\ldots,m,\ j=1,\ldots,n_i,\ i,j\neq k,1,\right.$$
$$\left. A_i \stackrel{def}{=} \cdots + x_{ij}[\tau_{ij},w_{ij},s_{ij}]\cdot A_k + \cdots\right\} +$$
$$\prod_{s_{k1}}\frac{N(a,x_{k1})}{\tau_{k1}} +$$
$$\sum\left\{\prod_{s_{ij}}\frac{N(a,x_{ij})}{\tau_{ij}} \mid i=1,\ldots,m,\ j=1,\ldots,n_i,\ i,j\neq k,1,\right.$$
$$\left. A_i \stackrel{def}{=} \cdots + x_{ij}[\tau_{ij},w_{ij},s_{ij}]\cdot A_k + \cdots\right\} =$$

$$\sum\left\{\prod_{s_{ij}}\frac{N(a,x_{ij})}{\tau_{ij}} \mid i=1,\ldots,m,\ j=1,\ldots,n_i\right\}$$

which is the mean a-performance of the agent before transformation.

□

Note that the application of Transformation 4.1 reduces the complexity of the system of equations by 1. Of course, the transformation can be applied repeatedly.

Transformation 4.1 allows us to remove certain states together with their defining equations from a system of behaviour equations. Transformation 4.2 involves the deletion of transitions from such a system.

Transformation 4.2
Consider again the specification of agent A_1, and assume that there exists a state A_k with equation

$$A_k \stackrel{def}{=} \cdots + x_{kv}[\tau_{kv},w_{kv},s_{kv}]\cdot A_{kv} + \cdots + x_{kw}[\tau_{kw},w_{kw},s_{kw}]\cdot A_{kw} + \cdots$$

such that $x_{kv}=x_{kw}$, $\tau_{kv}=\tau_{kw}$, $A_{kv}\equiv A_{kw}$, and $s_{kv}\neq s_{kw}$.

If we replace transitions s_{kv} and s_{kw} by $x_{kv}[\tau_{kv},w_{kv}+w_{kw},s_{kv}]\cdot A_{kv}$, the mean a-performance before and after transformation remain the same for all a.

□

Proof

Since the proof is as easy as the proof of Transformation 4.1, we will therefore not consider it.

□

The application of Transformation 4.2 also reduces the complexity of the system of equations by 1. Of course, this transformation can also be applied repeatedly.

Chapter 5

Example

As an example we will use the well-known alternating-bit protocol. A simple specification of this protocol is given in [Mil89]. We will model a slightly different version. The specification consists of the following six communicating processes: a Sender, a Receiver, two Timers, a Transmission Channel, and an Acknowledgement Channel, which are connected as is shown in figure 5.1.

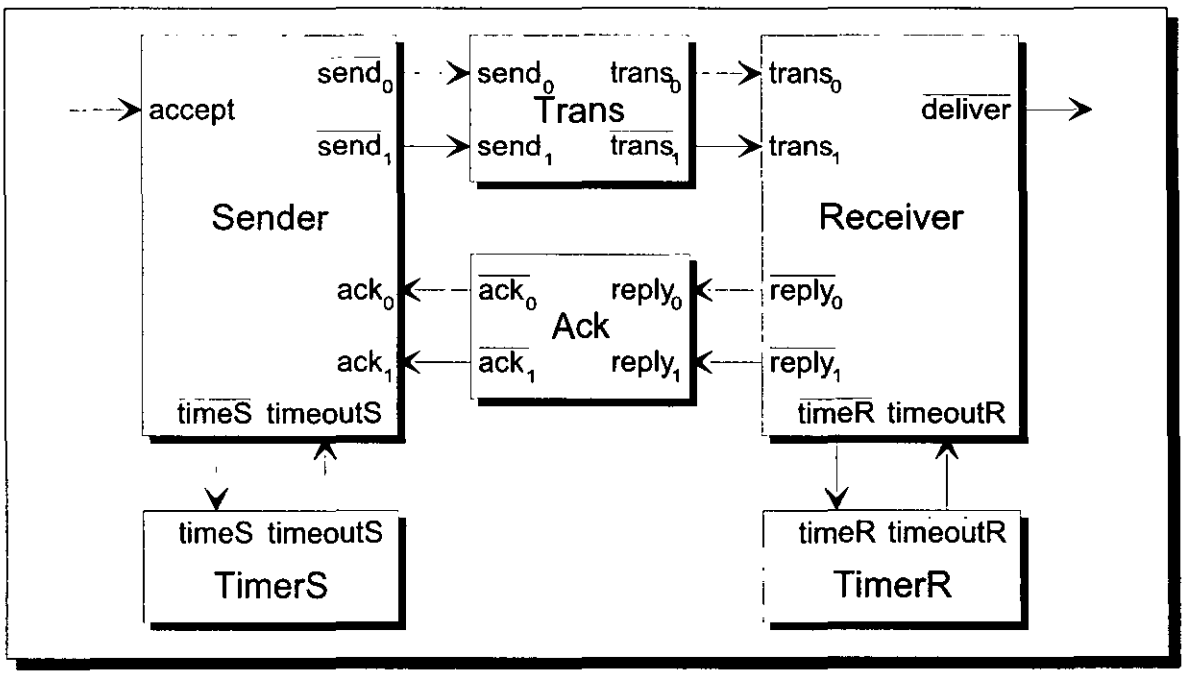

Figure 5.1: Structure of the alternating-bit protocol specification

If the Sender receives a message (together with some data) from its input port, it sets

its Timer, adds a bit to the received message, and sends the tagged message via the Transmission Channel to the Receiver. The value of the bit is the complement of the value added to the previous message. After the tagged message has been sent, the Sender waits for a message together with an acknowledge bit to arrive. The value of this latter bit should be equal to the value of the bit added to the previously sent message. If this is the case, a new message can be offered by the environment. If the bits are not equal, the received message (with the acknowledge bit) is simply ignored. If the Timer expires and gives a time-out, the tagged message is retransmitted. The Receiver operates in a similar manner. If the Receiver is offered a tagged message it checks whether the tag bit equals the expected bit. If this is the case, the message (together with the data) is offered to the environment, an acknowledge message is offered to the Acknowledgement Channel, and the Timer is set. If the bit differs from the expected bit, the message is just ignored. When the Receiver gets a time-out, the previously sent acknowledge message is retransmitted. Both channels are unreliable: they can occasionally lose messages. Further, if a message is offered to the channel before the previous message is delivered, the old message is lost.

First, we will give a "pure" functional specification in dtCCS of the alternating-bit protocol and its components. The protocol itself is specified as

$$\big(Accept_1 \mid Reply_0 \mid TimerS \mid TimerR \mid Trans \mid Ack\big) \setminus \{send_1, send_0, trans_1, trans_0, reply_1, reply_0, ack_1, ack_0, timeS, timeoutS, timeR, timeoutR\}$$

The Sender, with start state $Accept_1$, is specified as

$$\begin{aligned}
Accept_1 &\stackrel{def}{=} [\![accept]\!] \cdot Send_1 \\
Send_1 &\stackrel{def}{=} [\![\overline{send_1}]\!] \cdot SetTimerS_1 \\
SetTimerS_1 &\stackrel{def}{=} [\![\overline{timeS}]\!] \cdot Sending_1 \\
Sending_1 &\stackrel{def}{=} [\![timeoutS]\!] \cdot Send_1 \;+\; [\![ack_1]\!] \cdot Accept_0 \;+\; [\![ack_0]\!] \cdot Sending_1 \\
Accept_0 &\stackrel{def}{=} [\![accept]\!] \cdot Send_0 \\
Send_0 &\stackrel{def}{=} [\![\overline{send_0}]\!] \cdot SetTimerS_0 \\
SetTimerS_0 &\stackrel{def}{=} [\![\overline{timeS}]\!] \cdot Sending_0 \\
Sending_0 &\stackrel{def}{=} [\![timeoutS]\!] \cdot Send_0 \;+\; [\![ack_0]\!] \cdot Accept_1 \;+\; [\![ack_1]\!] \cdot Sending_0
\end{aligned}$$

The Receiver starts in state $Reply_0$ and is specified as

$$\begin{aligned}
Reply_0 &\stackrel{def}{=} [\![\overline{reply_0}]\!] \cdot SetTimerR_0 \\
SetTimerR_0 &\stackrel{def}{=} [\![\overline{timeR}]\!] \cdot Replying_0 \\
Replying_0 &\stackrel{def}{=} [\![timeoutR]\!] \cdot Reply_0 \;+\; [\![trans_0]\!] \cdot Replying_0 \;+\; [\![trans_1]\!] \cdot Deliver_1 \\
Deliver_1 &\stackrel{def}{=} [\![\overline{deliver}]\!] \cdot Reply_1 \\
Reply_1 &\stackrel{def}{=} [\![\overline{reply_1}]\!] \cdot SetTimerR_1
\end{aligned}$$

$$
\begin{aligned}
SetTimerR_1 &\stackrel{def}{=} [\![\overline{timeR}]\!] \cdot Replying_1 \\
Replying_1 &\stackrel{def}{=} [\![timeoutR]\!] \cdot Reply_1 + [\![trans_1]\!] \cdot Replying_1 + [\![trans_0]\!] \cdot Deliver_0 \\
Deliver_0 &\stackrel{def}{=} [\![\overline{deliver}]\!] \cdot Reply_0
\end{aligned}
$$

The timer at the Sender side has start state $TimerS$ and the timer at the Receiver side starts in state $TimerR$. The timers are defined as

$$
\begin{aligned}
TimerS &\stackrel{def}{=} [\![timeS]\!] \cdot TimerSSet \\
TimerSSet &\stackrel{def}{=} [\![timeS]\!] \cdot TimerSSet + [\![\overline{timeoutS}]\!] \cdot TimerS
\end{aligned}
$$

and

$$
\begin{aligned}
TimerR &\stackrel{def}{=} [\![timeR]\!] \cdot TimerRSet \\
TimerRSet &\stackrel{def}{=} [\![timeR]\!] \cdot TimerRSet + [\![\overline{timeoutR}]\!] \cdot TimerR
\end{aligned}
$$

The Transmission channel $Trans$ and the Acknowledgement channel Ack are given respectively by

$$
\begin{aligned}
Trans &\stackrel{def}{=} [\![send_0]\!] \cdot TransFull_0 + [\![send_0]\!] \cdot Trans \\
&\quad + [\![send_1]\!] \cdot TransFull_1 + [\![send_1]\!] \cdot Trans \\
TransFull_0 &\stackrel{def}{=} [\![send_0]\!] \cdot TransFull_0 + [\![send_0]\!] \cdot Trans \\
&\quad + [\![send_1]\!] \cdot TransFull_1 + [\![send_1]\!] \cdot Trans + [\![\overline{trans_0}]\!] \cdot Trans \\
TransFull_1 &\stackrel{def}{=} [\![send_0]\!] \cdot TransFull_0 + [\![send_0]\!] \cdot Trans \\
&\quad + [\![send_1]\!] \cdot TransFull_1 + [\![send_1]\!] \cdot Trans + [\![\overline{trans_1}]\!] \cdot Trans
\end{aligned}
$$

and

$$
\begin{aligned}
Ack &\stackrel{def}{=} [\![reply_0]\!] \cdot AckFull_0 + [\![reply_0]\!] \cdot Ack \\
&\quad + [\![reply_1]\!] \cdot AckFull_1 + [\![reply_1]\!] \cdot Ack \\
AckFull_0 &\stackrel{def}{=} [\![reply_0]\!] \cdot AckFull_0 + [\![reply_0]\!] \cdot Ack \\
&\quad + [\![reply_1]\!] \cdot AckFull_1 + [\![reply_1]\!] \cdot Ack + [\![\overline{ack_0}]\!] \cdot Ack \\
AckFull_1 &\stackrel{def}{=} [\![reply_0]\!] \cdot AckFull_0 + [\![reply_0]\!] \cdot Ack \\
&\quad + [\![reply_1]\!] \cdot AckFull_1 + [\![reply_1]\!] \cdot Ack + [\![\overline{ack_1}]\!] \cdot Ack
\end{aligned}
$$

Note that if any of the channels receives some message, it can either lose it by returning to its initial state, or proceed to a state where the message can be delivered.

If we interpret the specification as a normal fsCCS specification, we can prove that it is observational equivalent to ABP_0 specified as

$$
\begin{aligned}
ABP_0 &\stackrel{def}{=} accept \cdot ABP_1 \\
ABP_1 &\stackrel{def}{=} \overline{deliver} \cdot ABP_0
\end{aligned}
$$

which means that externally the protocol behaves as a 1-place buffer.

An interesting question is: How many *accept* actions can be performed per time step? Of course, the answer to this question depends on the following parameters:

- the time-out time of *TimerR*
- the time-out time of *TimerS*
- the transmission time of *Trans*
- the transmission time of *Ack*
- the transmission-error probability of *Trans*
- the transmission-error probability of *Ack*

Assume we would like *TimerR* to time-out after n time steps. Then we could adapt its specification and write

$$\begin{array}{lcl}
TimerR & \stackrel{def}{=} & [\![timeR]\!] \cdot TimerRSet_1 \\
TimerRSet_1 & \stackrel{def}{=} & [\![timeR]\!] \cdot TimerRSet \; + \; [\![]\!] \cdot TimerRSet_2 \\
TimerRSet_2 & \stackrel{def}{=} & [\![timeR]\!] \cdot TimerRSet \; + \; [\![]\!] \cdot TimerRSet_3 \\
\vdots & \vdots & \vdots \\
TimerRSet_n & \stackrel{def}{=} & [\![timeR]\!] \cdot TimerRSet \; + \; [\![\overline{timeoutR}]\!] \cdot TimerR
\end{array}$$

If we interpret this specification as a normal fsCCS specification, it is not hard to show that it is observational equivalent to the original specification. This means that we may replace the old timer with the new one without changing the abstract meaning of the protocol. In a similar manner we can modify *TimerS*, *Trans*, and *Ack*. We will assume that both timers time-out in exactly n time steps, which means that if a timer is set at, say, time k, a possible time-out is generated at time $k + n$. Next, we assume that it takes m time steps to transmit a message over *Trans* or *Ack*, which means that if a message is offered to a channel at time k, if no transmission error occurs, and if in the mean time no other message is offered, it is delivered at time $k + m$. Further, we suppose that the transmission-error probability of *Trans* and *Ack* both equal p. This means that if we send a message over the channels, and if we wait long enough for the message to be transmitted, the probability that the message reaches the other side equals p. In the context of the channel specifications, p is interpreted as indicated by the following example: Assume the transmission channel is in state *Trans*. Then it is able to receive, for example, message $send_0$, after which it can return to its initial state or proceed to state $Transfull_0$. The probability that the former choice is taken equals to $1 - p$. The probability of the latter choice is p.

Before we can calculate the mean *accept*-performance, we have to fix the parameters n, m, and p. We have chosen $m = 2$, $n \in \{1,2,3,4,5,6,7,8\}$, and $p \in \{0, \frac{1}{10}, \frac{1}{2}, \frac{9}{10}, 1\}$. Next, we have to apply the discrete time expansion law given in Proposition 3.15. This yields a single agent which has an equivalent time behaviour as the alternating-bit protocol. Then we have to add transition probabilities (weights), transition-time durations, and transition numbers as shown in the example of Section 4.3. Finally, the resulting behaviour equations can be reduced by applying Transformations 4.1 and 4.2 of Section 4.5, and the mean *accept*-performance can be calculated. The (rounded) results are shown in table 5.1.

$n \setminus p$	0	$\frac{1}{10}$	$\frac{1}{2}$	$\frac{9}{10}$	1
1	0.0000	0.0000	0.0000	0.0000	0.0000
2	0.0000	0.0125	0.0625	0.1125	0.1250
3	0.0000	0.0101	0.0526	0.0989	0.1111
4	0.0000	0.0085	0.0455	0.0882	0.1000
5	0.0000	0.0074	0.0417	0.0865	0.1000
6	0.0000	0.0066	0.0417	0.1023	0.1250
7	0.0000	0.0059	0.0385	0.1000	0.1250
8	0.0000	0.0053	0.0357	0.0978	0.1250

Table 5.1: Mean *accept*-performance figures of the alternating-bit protocol

A time table of the protocol can be found in table 5.2. This table indicates which actions are performed per time step, except for time-outs.

From these results the following conclusions can be drawn:

- For $p = 0$ not a single message can be delivered by the protocol, since the error rate of both channels is 100%, and the mean *accept*-performance equals 0 as expected.
- For $n = 1$ also no message can be delivered by the protocol, since both timers give a time-out the next time step after being set. This results in a continuous retransmission of messages by both Sender and Receiver, which are ignored by both Receiver and Sender, and again the mean *accept*-performance equals 0 as expected.
- For $n = 2$, although both timers give a time-out which results in a retransmission of the message, these retransmitted messages are simply ignored by both Sender and Receiver, since both receive the message they expect before the timers can give another time-out. Hence, the protocol is independent of these time-outs. Since the protocol needs 8 time steps to deliver a message, the mean-performance figures are as shown in table 5.1.

time step	TimeS	Sender	Trans	Ack	Receiver	TimerR
1		$accept$				
2		$\overline{send_1}$	$send_1$			
3	$timeS$	$\overline{timeS}$	τ			
4			$\overline{trans_1}$		$trans_1$	
5					$deliver$	
6				$reply_1$	$\overline{reply_1}$	
7				τ	$\overline{timeR}$	$timeR$
8		ack_1		$\overline{ack_1}$		
9		$accept$				
10		$\overline{send_0}$	$send_0$			
11	$timeS$	$\overline{timeS}$	τ			
12			$\overline{trans_0}$		$trans_0$	
13					$deliver$	
14				$reply_0$	$\overline{reply_0}$	
15				τ	$\overline{timeR}$	$timeR$
16		ack_0		$\overline{ack_0}$		

Table 5.2: Time table of the alternating-bit protocol

- For $n = 3, 4$ and for $p = 1$ the same happens as for $n = 2$, but now the protocol needs respectively 1 and 2 time steps extra to deliver a message, due to the retransmission, and the mean *accept*-performance is $\frac{1}{9}$ and $\frac{1}{10}$ respectively.

- For $n = 3, 4$ and $p \in \{\frac{1}{10}, \frac{1}{2}, \frac{9}{10}\}$ the protocol also needs extra time steps to deliver a message and the performance figures are also lower than for $n = 2$.

- For $n = 5$ both timers give a time-out at a time step at which both Sender and Receiver can also receive a message. This *slows down* the performance drop set into motion for $n = 3, 4$. Without further analysis nothing can be said about this.

- For $n = 6, 7, 8$ and for $p = 1$ the timers give a time-out just after both Sender and Receiver have received the message they expect and the protocol is again independent of any time-out and the mean *accept*-performance is again $\frac{1}{8}$.
- For $n = 6, 7, 8$ and for $p \in \{\frac{1}{10}, \frac{1}{2}, \frac{9}{10}\}$ the trend described for $n = 3, 4$ is continued.

Finally, we can conclude that for $m = 2$ the best performance is obtained for $n = 2$, irrespective of the value of p. If error-free communication channels are used, i.e., $p = 1$, the best performance is also obtained for $n \geq 6$.

Chapter 6

Conclusions and Future Work

In this report we have developed a timed variant of Milner's CCS, called dtCCS (discrete time CCS). The resulting calculus resembles the *synchronous* calculus described in Paragraph 9.3 of [Mil89]. The only two differences are that we replaced the operator ∂ implicitly by two inference rules and that this latter calculus does not incorporate maximal progress. dtCCS builds upon the notions and definitions of a finite-state version of CCS, called fsCCS (finite state CCS), which we have also developed in this report.

Further, we have developed a method for the calculation of so-called mean-performance figures of systems of communicating processes described in dtCCS. This calculation method is based on a stochastic performance-analysis model.

As an example we have applied the methods described in this report to a version of the well-known alternating-bit protocol. First, we have described the protocol in dtCCS. Next, we have verified, by interpreting the dtCCS descriptions as fsCCS descriptions, that the protocol is a 1-place buffer. Finally, we have calculated the mean-performance figures of this protocol for a number of values for the time-out time, the transmission time, and the transmission-error probability.

The presented tools have been developed as components of a design methodology for system architectures for HMPs (Heap Management Processors). The other components of this design methodology are a number of supporting automated tools, called CCStool2, dtCCStool1, PAtool1, and Mathematica, and a design cycle consisting of a specification phase, a synthesize phase, an implementation phase, and a verification phase. This design methodology has been developed as a part of the first phase of a project for the development of a design methodology for (application-specific) HMPs.

The calculi and tools have already proven to be very useful. However, there still is room for several improvements and extensions. The most urgent ones are the following:

- Extend dtCCS (and dtCCStool1) to support compositionality, thereby allowing the analysis of more complex systems. Equivalence relations in dtCCS are not substitutive under $\mathcal{M}$. This means that to calculate combined maximal-progress behaviour of a system of communicating processes, we first have to calculate the discrete-time expansion of the system as a whole and only then we may apply $\mathcal{M}$ to this expanded system. The problem is that the intermediate result can become *huge*, even if the maximal-progress variant has an average complexity. A possible solution is to incorporate maximal progress *directly* into parallel composition. This necessarily implies that the notion of maximal progress is weakened. This on its turn results in a larger maximal-progress expansion than before. We do not know yet how this should be established or what the practical consequences will be.

- Extend dtCCS to allow for the description of parallel actions of a single process.

- Incorporate a linear-equation solver in PAtool1, which opens the way to calculate the mean-performance figures automatically. This removes the necessity to use Mathematica which proved to be of limited use with respect to solving large sets of linear equations. Further, it increases the portability of the whole design methodology to other computing platforms.

- Extend the calculation of performance figures to include, for example, best-case and worst-case figures.

Finally, we could extend the dtCCS description language to allow for the description of the performance probabilities of a single process. Currently, one has to work its way through the expanded dtCCS description and set the performance probabilities of the appropriate actions manually, which is rather laborious, time consuming, and prone to error. Of course, the functions of dtCCStool1 have to respect these extensions of a dtCCS description. If we combine this with the possibility to describe the parallel behaviour of actions of a single agent, we could get a complete description of the *functional*, *time*, and *performance probability* behaviour of a single process.

Appendix A

Proofs of dtCCS Properties and Lemmas

Proof of Proposition 3.10

The proof proceeds by induction on n.

For $n = 1$ we will first proof the 'only if' part. Assume $A_1[f_1] \xrightarrow{a,l} G$. Then by rules **dtRel** and **dtCon**, $A_1 \stackrel{def}{=} \sum_{i \in I} a_i \cdot A_i$, $G \equiv A_i[f_1]$, $A_1 \xrightarrow{a_i,\langle i \rangle} A_i$, $a = f_1(a_i)$ and $l = \langle i \rangle$ for some $i \in I$. The result follows by choosing $a_1 = a_i$, $A'_1 \equiv A_i$, since $\langle l_1 \rangle = l \in \mathbb{N}^1 \setminus \{\mathbf{0}^1\}$ and since $a \in \{a\} = PS([\![a]\!]) = PS([\![f_1(a_i)]\!])$.

For the 'if' part let a_1, A'_1 and $l \in \mathbb{N}^1 \setminus \{\mathbf{0}^1\}$ be such that $A_1 \xrightarrow{a_1,\langle l_1 \rangle} A'_1$ or $(l_1 = 0 \ \wedge \ A'_1 \equiv A_1 \ \wedge \ a_1 = [\![]\!])$, and let $a \in PS([\![f_1(a_1)]\!])$. Then because $l_1 \neq 0$ we have $A_1 \xrightarrow{a_1,\langle l_1 \rangle} A'_1$. Further, if $a \in PS([\![f_1(a_1)]\!])$, then $a = f_1(a_1)$. Also $\langle l_1 \rangle = l$, so, $A_1[f_1] \xrightarrow{a,l} A'_1$.

Suppose the result holds for some $n \geq 1$. We will prove it for $n+1$. We will start with the 'only if' part. Assume $A_1[f_1] \mid \ldots \mid A_n[f_n] \mid A_{n+1}[f_{n+1}] \xrightarrow{a,l} G$. We distinguish the following 3 cases:

(i) $A_{n+1} \stackrel{def}{=} \sum_{i \in I} a_i \cdot A_i$, $A_{n+1} \xrightarrow{a_i,\langle i \rangle} A_i$, $G \equiv (A_1[f_1] \mid \cdots \mid A_n[f_n]) \mid A_i[f_{n+1}]$, $a = f_{n+1}(a_i)$, $l = \mathbf{0}^n \cdot \langle i \rangle$ for some $i \in I$. We choose $a_1 = [\![]\!]$, ..., $a_n = [\![]\!]$, $A'_1 \equiv A_1$, ..., $A'_n \equiv A_n$, $a_{n+1} = a_i$, $A'_{n+1} \equiv A_i$. Then the result follows because $l_1 = 0$, ..., $l_n = 0$, $A_{n+1} \xrightarrow{a_i,\langle i \rangle} A'_{n+1}$, $l \in \mathbb{N}^{n+1} \setminus \{\mathbf{0}^{n+1}\}$, $a \in \{a\} = PS([\![a]\!]) = PS([\![[\![]\!], \ldots, [\![]\!], a]\!]) = PS([\![f_1([\![]\!]), \ldots, f_n([\![]\!]), f_{n+1}(a_{n+1})]\!])$.

(ii) $A_1[f_1] \mid \ldots \mid A_n[f_n] \xrightarrow{a,m} H$, $G \equiv H \mid A_{n+1}[f_{n+1}]$, $l = m \cdot \mathbf{0}^1$. Then by induction $m \in \mathbb{N}^n \setminus \{\mathbf{0}^n\}$ and we can find atoms $a_1, \ldots, a_n$ and agent constants $A'_1, \ldots, A'_n$ such that $H \equiv A'_1[f_1] \mid \ldots \mid A'_n[f_n]$, $a \in PS([\![f_1(a_1), \ldots, f_n(a_n)]\!])$ and

$$\begin{array}{lll} A_1 & \stackrel{a_1,\langle l_1\rangle}{\longrightarrow} & A_1' \,\mathrm{or}(l_1 = 0 \wedge A_1' \equiv A_1 \wedge a_1 = [\![]\!]) \\ A_2 & \stackrel{a_2,\langle l_2\rangle}{\longrightarrow} & A_2' \,\mathrm{or}(l_2 = 0 \wedge A_2' \equiv A_2 \wedge a_2 = [\![]\!]) \\ \vdots & \vdots & \vdots \\ A_n & \stackrel{a_n,\langle l_n\rangle}{\longrightarrow} & A_n' \,\mathrm{or}(l_n = 0 \wedge A_n' \equiv A_n \wedge a_n = [\![]\!]) \end{array}$$

For the $(n+1)^{th}$ agent we choose $l_{n+1} = 0$, $A'_{n+1} \equiv A_{n+1}$ and $a_{n+1} = [\![]\!]$. Then $a \in PS([\![f_1(a_1), \ldots, f_n(a_n), f_{n+1}(a_{n+1})]\!])$ because $PS([\![f_1(a_1), \ldots, f_n(a_n), f_{n+1}(a_{n+1})]\!])$ $= PS([\![f_1(a_1), \ldots, f_n(a_n), f_{n+1}([\![]\!])]\!]) = PS([\![f_1(a_1), \ldots, f_n(a_n)]\!])$. Further, because $m \in \mathbb{N}^n \setminus \{\mathbf{0}^n\}$ we have $l \in \mathbb{N}^{n+1} \setminus \{\mathbf{0}^{n+1}\}$.

(iii) $A_1[f_1] \mid \ldots \mid A_n[f_n] \stackrel{b,m}{\longrightarrow} H$, $A_{n+1} \stackrel{def}{=} \sum_{i \in I} a_i \cdot A_i$, $A_{n+1} \stackrel{a_i,\langle i\rangle}{\longrightarrow} A_i$, $G \equiv H \mid A_i[f_{n+1}]$, $l = m \cdot \langle i \rangle$, and a is a p.s. of b and $f_{n+1}(a_i)$ for some $i \in I$. Then by induction $m \in \mathbb{N}^n \setminus \{\mathbf{0}^n\}$ and we can find atoms $a_1, \ldots, a_n$ and agent constants $A'_1, \ldots, A'_n$ such that $H \equiv A'_1[f_1] \mid \ldots \mid A'_n[f_n]$, $b \in PS([\![f_1(a_1), \ldots, f_n(a_n)]\!])$ and

$$\begin{array}{lll} A_1 & \stackrel{a_1,\langle l_1\rangle}{\longrightarrow} & A_1' \,\mathrm{or}(l_1 = 0 \wedge A_1' \equiv A_1 \wedge a_1 = [\![]\!]) \\ A_2 & \stackrel{a_2,\langle l_2\rangle}{\longrightarrow} & A_2' \,\mathrm{or}(l_2 = 0 \wedge A_2' \equiv A_2 \wedge a_2 = [\![]\!]) \\ \vdots & \vdots & \vdots \\ A_n & \stackrel{a_n,\langle l_n\rangle}{\longrightarrow} & A_n' \,\mathrm{or}(l_n = 0 \wedge A_n' \equiv A_n \wedge a_n = [\![]\!]) \end{array}$$

For the $(n+1)^{th}$ agent we choose $a_{n+1} = a_i$, $\langle l_{n+1}\rangle = \langle i \rangle$ and $A'_{n+1} \equiv A_i$. Then $A_{n+1} \stackrel{a_{n+1},\langle l_{n+1}\rangle}{\longrightarrow} A'_{n+1}$ and $l \in \mathbb{N}^{n+1} \setminus \{\mathbf{0}^{n+1}\}$. Further, we have a is a p.s. of b and $f_{n+1}(a_{n+1})$ and $b \in PS([\![f_1(a_1), \ldots, f_n(a_n)]\!])$, so, $(\exists\, b : b \in PS([\![f_1(a_1), \ldots, f_n(a_n)]\!]) :$ a is a p.s. of b and $f_{n+1}(a_{n+1}))$, which means that $a \in PS([\![f_1(a_1), \ldots, f_n(a_n)]\!]$ $\uplus [\![f_{n+1}(a_{n+1})]\!]) = PS([\![f_1(a_1), \ldots, f_{n+1}(a_{n+1})]\!])$.

For the 'if' part of the induction step, let $a_1, \ldots, a_{n+1}$, $A'_1, \ldots, A'_{n+1}$, $l \in \mathbb{N}^{n+1} \setminus \{\mathbf{0}^{n+1}\}$, G and a be such that $G \equiv A'_1[f_1] \mid \ldots \mid A'_{n+1}[f_{n+1}]$, $a \in PS([\![f_1(a_1), \ldots, f_{n+1}(a_{n+1})]\!])$ and

$$\begin{array}{lll} A_1 & \stackrel{a_1,\langle l_1\rangle}{\longrightarrow} & A_1' \,\mathrm{or}(l_1 = 0 \wedge A_1' \equiv A_1 \wedge a_1 = [\![]\!]) \\ \vdots & \vdots & \vdots \\ A_n & \stackrel{a_n,\langle l_n\rangle}{\longrightarrow} & A_n' \,\mathrm{or}(l_n = 0 \wedge A_n' \equiv A_n \wedge a_n = [\![]\!]) \\ A_{n+1} & \stackrel{a_{n+1},\langle l_{n+1}\rangle}{\longrightarrow} & A_{n+1}' \,\mathrm{or}(l_{n+1} = 0 \wedge A_{n+1}' \equiv A_{n+1} \wedge a_{n+1} = [\![]\!]) \end{array}$$

The following 3 cases can be distinguished:

(i) $l_{n+1} = 0 \wedge A'_{n+1} \equiv A_{n+1} \wedge a_{n+1} = [\![]\!]$. We define m, H such that $l = m \cdot \mathbf{0}^1$ and $H \equiv A'_1[f_1] \mid \ldots \mid A'_n[f_n]$. Then $a \in PS([\![f_1(a_1), \ldots, f_n(a_n)]\!])$ because $a_n = [\![]\!]$. By induction we then have $A_1[f_1] \mid \ldots \mid A_n[f_n] \stackrel{a,m}{\longrightarrow} H$. But then $A_1[f_1] \mid \ldots \mid A_n[f_n] \mid A_{n+1}[f_{n+1}]$ $\stackrel{a,m\cdot\mathbf{0}^1}{\longrightarrow} H \mid A_{n+1}[f_{n+1}]$. And thus $A_1[f_1] \mid \ldots \mid A_{n+1}[f_{n+1}] \stackrel{a,l}{\longrightarrow} G$.

(ii) $A_{n+1} \stackrel{a_{n+1},\langle l_{n+1}\rangle}{\rightarrow} A'_{n+1}$ and $(l_1 = 0 \wedge A'_1 \equiv A_1 \wedge a_1 = [\![]\!]), \ldots, (l_n = 0 \wedge A'_n \equiv A_n \wedge a_n = [\![]\!])$. Then because $a \in PS([\![f_1(a_1), \ldots, f_{n+1}(a_{n+1})]\!]) = PS([\![f_{n+1}(a_{n+1})]\!])$ $= \{f_{n+1}(a_{n+1}),\ a = f_{n+1}(a_{n+1})$. But then $A_1[f_1] \mid \ldots \mid A_n[f_n] \mid A_{n+1}[f_{n+1}] \stackrel{a,\mathbf{0}^n\cdot\langle l_{n+1}\rangle}{\rightarrow}$ $A_1[f_1] \mid \ldots \mid A_n[f_n] \mid A'_{n+1}[f_{n+1}]$ and thus $A_1[f_1] \mid \ldots \mid A_n[f_n] \mid A_{n+1}[f_{n+1}] \stackrel{a,l}{\rightarrow} G$.

(iii) $A_{n+1} \stackrel{a_{n+1},\langle l_{n+1}\rangle}{\rightarrow} A'_{n+1}$ and for some $k : 1 \leq k \leq n$, $A_k \stackrel{a_k,\langle l_k\rangle}{\rightarrow} A'_k$. We define m, H such that $l = m \cdot \langle l_{n+1}\rangle$ and $H \equiv A'_1[f_1] \mid \ldots \mid A'_n[f_n]$. Further, we pick a $b \in PS([\![f_1(a_1), \ldots, f_n(a_n)]\!])$ with the property that a is a p.s. of b and $f_{n+1}(a_{n+1})$ (note that this can always be done). Then by induction $A_1[f_1] \mid \ldots \mid A_n[f_n] \stackrel{b,m}{\rightarrow} H$. But then $A_1[f_1] \mid \ldots \mid A_n[f_n] \mid A_{n+1}[f_{n+1}] \stackrel{a,m\cdot\langle l_{n+1}\rangle}{\rightarrow} H \mid A'_{n+1}$. So, $A_1[f_1] \mid \ldots \mid A_{n+1}[f_{n+1}]$ $\stackrel{a,l}{\rightarrow} G$.

This ends the induction step and thereby the proof of Proposition 3.10.

□

Proof of Lemma 3.1

The proof proceeds by an easy transition induction on the complexity of the derivation tree of $E \stackrel{a,l}{\rightarrow} E'$ using Definition 3.5.

□

Proof of Lemma 3.2

The proof is an easy transition induction.

□

Proof of Proposition 3.11

(1) Let each $\mathcal{S}_i$, with $i \in I$ for some index set I, be a v.s.d.t. bisimulation. Then it is easy to verify that $\bigcup_{i\in I} \mathcal{S}_i$ is a v.s.d.t. bisimulation. This means that $\stackrel{v}{\sim}$ is a v.s.d.t. bisimulation. Further, $\stackrel{v}{\sim}$ contains any other bisimulation, and hence it is the largest one.

(2) We have to proof that $\stackrel{v}{\sim}$ is reflexive (r), transitive (t), and symmetric (s).

(r) It is easy to proof that $I_{\mathcal{E}} = \{(E, E) \mid E \in \mathcal{E}\}$ is a v.s.d.t. bisimulation. Further, $(E, E) \in I_{\mathcal{E}}$ for any E and thus $E \stackrel{v}{\sim} E$ for any E.

(t) Let $\mathcal{S}_1$ and $\mathcal{S}_2$ be v.s.d.t. bisimulations and define $\mathcal{S}_1\mathcal{S}_2 = \{(E, G) \mid$ for some F $(E, F) \in \mathcal{S}_1$ and $(F, G) \in \mathcal{S}_2\}$. It it is easy to show that $\mathcal{S}_1\mathcal{S}_2$ is a v.s.d.t. bisimulation. Now suppose $E \stackrel{v}{\sim} F$ and $F \stackrel{v}{\sim} G$. Then $(E, F) \in \mathcal{S}_1$ and $(F, G) \in \mathcal{S}_2$ for some v.s.d.t. bisimulations $\mathcal{S}_1$ and $\mathcal{S}_2$. But then by definition $(E, G) \in \mathcal{S}_1\mathcal{S}_2$ and thus $E \stackrel{v}{\sim} G$.

(s) If S is a v.s.d.t. bisimulation, then $\mathcal{S}^{-1} = \{(F, E) \mid (E, F) \in \mathcal{S}\}$ is a v.s.d.t. bisimulation, which is easily shown. Suppose $E \overset{v}{\sim} F$. Then $(E, F) \in \mathcal{S}$ for some $\mathcal{S}$ and thus $(F, E) \in \mathcal{S}^{-1}$. But then $F \overset{v}{\sim} E$.

□

Proof of Proposition 3.12

We first define a relation $\overset{v'}{\sim}$ as follows:

$E \overset{v'}{\sim} F$ iff, for all $a \in \mathcal{A}$,

(i) whenever $E \overset{a,l_1}{\rightarrow} E'$ then, for some $m_1, F', F \overset{a,m_1}{\rightarrow} F'$ and $E' \overset{v}{\sim} F'$ and

- If $F \overset{b,m_2}{\rightarrow} F''$ such that $m_1 = m_2$ then, for some E'' and l_2 with $l_1 = l_2$, $E \overset{b,l_2}{\rightarrow} E''$ and
- If $F \overset{b,m_2}{\rightarrow} F''$ such that $m_1 \sqsubset m_2$ then, for some E'' and l_2 with $l_1 \sqsubset l_2$, $E \overset{b,l_2}{\rightarrow} E''$

(ii) whenever $F \overset{a,m_1}{\rightarrow} F'$ then, for some $l_1, E', E \overset{a,l_1}{\rightarrow} E'$ and $E' \overset{v}{\sim} F'$ and

- If $E \overset{b,l_2}{\rightarrow} E''$ such that $l_1 = l_2$ then, for some F'' and m_2 with $m_1 = m_2$, $F \overset{b,m_2}{\rightarrow} F''$ and
- If $E \overset{b,l_2}{\rightarrow} E''$ such that $l_1 \sqsubset l_2$ then, for some F'' and m_2 with $m_1 \sqsubset m_2$, $F \overset{b,m_2}{\rightarrow} F''$

Now let E and F be such that $E \overset{v}{\sim} F$. Then, because $\overset{v}{\sim}$ is a v.s.d.t. bisimulation, we have $E \overset{v'}{\sim} F'$. So $E \overset{v}{\sim} F$ implies $E \overset{v'}{\sim} F'$.

We also have that $\overset{v'}{\sim}$ is a v.s.d.t. bisimulation. To see this, let $E \overset{v'}{\sim} F$ and assume $E \overset{a,l_1}{\rightarrow} E'$. Then by the definition of $\overset{v'}{\sim}$, $F \overset{a,m_1}{\rightarrow} F'$ and $E' \overset{v}{\sim} F'$ and if $F \overset{b,m_1}{\rightarrow} F''$ then $E \overset{b,l_1}{\rightarrow} E''$ and if $F \overset{b,m_2}{\rightarrow} F''$ for $m_2 \sqsupset m_1$, then $E \overset{b,l_2}{\rightarrow} E''$ for some $l_2 \sqsupset l_1$. But then, because $E' \overset{v}{\sim} F'$ implies $E' \overset{v'}{\sim} F'$, we also have $F \overset{a,m_1}{\rightarrow} F'$ and $E' \overset{v'}{\sim} F'$ and if $F \overset{b,m_1}{\rightarrow} F''$ then $E \overset{b,l_1}{\rightarrow} E''$ and if $F \overset{b,m_2}{\rightarrow} F''$ for $m_2 \sqsupset m_1$, then $E \overset{b,l_2}{\rightarrow} E''$ for some $l_2 \sqsupset l_1$. So, part (i) of Definition 3.8 is satisfied. In a similar fashion we can show part (ii).

Now suppose $E \overset{v'}{\sim} F$ then, because $\overset{v'}{\sim}$ is a v.s.d.t. bisimulation, we also have $E \overset{v}{\sim} F$. So, we now have $E \overset{v}{\sim} F$ iff $E \overset{v'}{\sim} F$ and thus $\overset{v}{\sim} = \overset{v'}{\sim}$. We are then allowed to substitute relation $\overset{v'}{\sim}$ in the definition of $\overset{v'}{\sim}$ by relation $\overset{v}{\sim}$, which gives us Proposition 3.12.

□

Proof of Proposition 3.13

All these laws may be proved by applying Proposition 3.12 or by exhibiting appropriate very strong discrete time bisimulations.

For part (1), we will use Proposition 3.12. Let $\sum_{i\in I} a_i \cdot A_i \stackrel{a,l_1}{\rightarrow} F$ and let $\nu : I \rightarrow J$ be a bijective function such that $a_i = b_{\nu(i)}$ and $A_i \equiv B_{\nu(i)}$ for all $i \in I$. Then $a = a_i$, $F \equiv A_i$, and $l_1 = \langle i \rangle$ for some $i \in I$. But then clearly $\sum_{j\in J} b_j \cdot B_j \stackrel{a,\langle\nu(i)\rangle}{\rightarrow} A_i$ and by Proposition 3.11(2) $A_i \stackrel{v}{\sim} A_i$. Now suppose $\sum_{j\in J} b_j \cdot B_j \stackrel{b,m}{\rightarrow} H$ for some H and for some $m \sqsupseteq \langle\nu(i)\rangle$. Then necessarily $m = \langle\nu(i)\rangle$, $b = b_{\nu(i)}$, and $H \equiv B_{\nu(i)}$. And, of course, $\sum_{i\in I} a_i \cdot A_i \stackrel{b,\langle i\rangle}{\rightarrow} A_i$.

So, part (i) of Proposition 3.12 is satisfied. In a similar manner part (ii) is shown, and thus Proposition 3.13(1) holds.

The proof of part (2) of Proposition 3.13 is as simple as the proof of (1). We will not write it out.

For part (3) we will proof that $\mathcal{S} = \{(E \mid F, F \mid E) \mid E, F \in \mathcal{E}\}$ is a v.s.d.t. bisimulation. We will only consider condition (i) of Definition 3.8. Condition (ii) is handled in a similar manner. Let $(E \mid F, F \mid E) \in \mathcal{S}$ and let $E \mid F \stackrel{a,l}{\rightarrow} G$. Now it is not hard to verify that $l = l_e \cdot l_f$ with $l_e \in \mathbb{N}^{\#E}$ and $l_f \in \mathbb{N}^{\#F}$, that $G \equiv E' \mid F'$ for some E',F', and that $F \mid E \stackrel{a,l_f\cdot l_e}{\rightarrow} F' \mid E'$. It is clear that $(E' \mid F', F' \mid E') \in \mathcal{S}$. Now suppose $F \mid E \stackrel{b,m}{\rightarrow} H$ for some $m \sqsupseteq l_f \cdot l_e$. Then $m = m_f \cdot m_e$ with $m_f \in \mathbb{N}^{\#F}$ and $m_e \in \mathbb{N}^{\#E}$, $H \equiv F'' \mid E''$ for some F'',E'', and $E \mid F \stackrel{b,m_e\cdot m_f}{\rightarrow} E'' \mid F''$. Further, if $m = l_f \cdot l_e$, then $m_f = l_f$ and $m_e = l_e$ and thus $m_e \cdot m_f = l_e \cdot l_f = l$. This means that $E \mid F \stackrel{b,l}{\rightarrow} E'' \mid F''$. If $m \sqsupset l_f \cdot l_e$, then $m_f \cdot m_e \sqsupset l_f \cdot l_e$ and thus $m_e \cdot m_f \sqsubset l_e \cdot l_f$.

Part (4) of Proposition 3.13 is proved in an analogous way as part (3).

This ends our proof of Proposition 3.13.

□

Proof of Proposition 3.14

For part (1) we will show that $\mathcal{S} = \{(E_1 \mid F, E_2 \mid F) \mid E_1 \stackrel{v}{\sim} E_2\}$ is a v.s.d.t. bisimulation. Let $(E_1 \mid F, E_2 \mid F) \in \mathcal{S}$ and assume $E_1 \mid F \stackrel{a,l_1}{\rightarrow} G$. Then the following cases apply:

(i) $E_1 \stackrel{a,k_1}{\rightarrow} E_1'$, $G \equiv E_1' \mid F$ and $l_1 = k_1 \cdot \mathbf{0}^{\#F}$. Then, because $E_1 \stackrel{v}{\sim} E_2$, we have by Proposition 3.12, $E_2 \stackrel{a,n_1}{\rightarrow} E_2'$ with $E_1' \stackrel{v}{\sim} E_2'$. Then $E_2 \mid F \stackrel{a,m_1}{\rightarrow} E_2'$, with $m_1 = n_1 \cdot \mathbf{0}^{\#F}$,

and clearly $(E_1' \mid F, E_2' \mid F) \in \mathcal{S}$. Now suppose $E_2 \mid F \stackrel{b,m_2}{\rightarrow} H$ for some H and some $m_2 \sqsupseteq m_1$. Then we distinguish:

(a) $E_2 \stackrel{b,n_2}{\rightarrow} E_2''$, $H \equiv E_2'' \mid H$ and $m_2 = n_2 \cdot \mathbf{0}^{\#F}$. If $m_1 = m_2$, then $n_1 = n_2$ and by Proposition 3.12, $E_1, \stackrel{b,k_1}{\rightarrow} E_1''$ and thus $E_1 \mid F \stackrel{b,l_2}{\rightarrow} E_1'' \mid F$ with $l_2 = k_1 \cdot \mathbf{0}^{\#F}$. Then we have $l_1 = l_2$. If $m_1 \sqsubset m_2$ then $n_1 \sqsubset n_2$ and by Proposition 3.12, $E_1 \stackrel{b,k_2}{\rightarrow} E_1''$ with $k_2 \sqsupset k_1$ and thus $E_1 \mid F \stackrel{b,l_2}{\rightarrow} E_1'' \mid F$ with $l_2 = k_2 \cdot \mathbf{0}^{\#F}$. Clearly, $l_1 \sqsubset l_2$.

(b) $F \stackrel{b,o_2}{\rightarrow} F'$, $H \equiv E_2 \mid F'$ and $m_2 = \mathbf{0}^{\#E_2} \cdot o_2$. If $m_1 = m_2$, then $n_1 \cdot \mathbf{0}^{\#F} = \mathbf{0}^{\#E_2} \cdot o_2$ and thus, because $\#n_1 = \#E_2$ and $\#F = \#o_2$ by Lemma 3.1, $o_2 = \mathbf{0}^{\#F}$, which is impossible by Lemma 3.2. If $m_1 \sqsubset m_2$ then also $o_2 = \mathbf{0}^{\#F}$. Consequently, case (b) can never occur.

(c) $E_2 \stackrel{b_1,n_2}{\rightarrow} E_2''$, $F \stackrel{b_2,o_2}{\rightarrow} F'$, $H \equiv E_2'' \mid F'$, b is a p.s. of b_1 and b_2 and $m_2 = n_2 \cdot o_2$. If $m_1 = m_2$ then $n_1 \cdot \mathbf{0}^{\#F} = n_2 \cdot o_2$ and thus (using Lemma 1) $n_1 = n_2$ and $\mathbf{0}^{\#F} = o_2$, which is impossible by Lemma 3.2. If $m_1 \sqsubset m_2$ then $n_1 \cdot \mathbf{0}^{\#F} \sqsubset n_2 \cdot o_2$, so (using Lemma 3.1)

- $n_1 = n_2$ and $\mathbf{0}^{\#F} \sqsubset o_2$. Then (by Proposition 3.12) $E_1 \stackrel{b_1,k_1}{\rightarrow} E_1''$ and thus $E_1 \mid F \stackrel{b,l_2}{\rightarrow} E_1'' \mid F'$ with $l_2 = k_1 \cdot o_2$. Then $l_1 = k_1 \cdot \mathbf{0}^{\#F} \sqsubset k_1 \cdot o_2 = l_2$.
- $n_1 \sqsubset n_2$ and $\mathbf{0}^{\#F} = o_2$, but this cannot occur by Lemma 3.2.
- $n_1 \sqsubset n_2$ and $\mathbf{0}^{\#F} \sqsubset o_2$. Then $E_1 \stackrel{b_1,k_2}{\rightarrow} E_1''$ with $k_1 \sqsubset k_2$. So $E_1 \mid F \stackrel{b,l_2}{\rightarrow} E_1'' \mid F'$ with $l_2 = k_2 \cdot o_2$. And then $l_1 = k_1 \cdot \mathbf{0}^{\#F} \sqsubset k_2 \cdot o_2 = l_2$.

(ii) $F \stackrel{a,o_1}{\rightarrow} F'$, $G \equiv E_1 \mid F'$ and $l_1 = \mathbf{0}^{\#E_1} \cdot o_1$. Then $E_2 \mid F \stackrel{a,m_1}{\rightarrow} E_2 \mid F'$ with $m_1 = \mathbf{0}^{\#E_2} \cdot o_1$ and clearly $(E_1 \mid F', E_2 \mid F') \in \mathcal{S}$. Now assume $E_2 \mid F \stackrel{b,m_2}{\rightarrow} H$, for some H and some $m_2 \sqsupseteq m_1$. Then we have the following cases:

(a) $E_2 \stackrel{b,n_2}{\rightarrow} E_2'$, $H \equiv E_2' \mid F$ and $m_2 = n_2 \cdot \mathbf{0}^{\#F}$. However, if $m_1 \sqsubseteq m_2$, then $o_1 = \mathbf{0}^{\#F}$, which is not possible by Lemma 3.2, so this case cannot apply.

(b) $F \stackrel{b,o_2}{\rightarrow} F''$, $H \equiv E_2 \mid F''$ and $m_2 = \mathbf{0}^{\#E_2} \cdot o_2$. Then $E_1 \mid F \stackrel{b,l_2}{\rightarrow} E_1 \mid F''$ with $l_2 = \mathbf{0}^{\#E_1} \cdot o_2$. If $m_1 = m_2$, then $o_1 = o_2$ and evidently $l_1 = l_2$. If $m_1 \sqsubset m_2$, then $o_1 \sqsubset o_2$ and thus $l_1 \sqsubset l_2$.

(c) $E_2 \stackrel{b_1,n_2}{\rightarrow} E_2'$, $F \stackrel{b_2,o_2}{\rightarrow} F''$, $H \equiv E_2' \mid F''$, b is a p.s. of b_1 and b_2, and $m_2 = n_2 \cdot o_2$. Then, because $E_1 \stackrel{v}{\sim} E_2$, $E_1 \stackrel{b_1,k_2}{\rightarrow} E_1'$ for some $k_2 \sqsupset \mathbf{0}^{\#E_1}$, and thus $E_1 \mid F \stackrel{b,l_2}{\rightarrow} E_1' \mid F''$ with $l_2 = k_2 \cdot o_2$. If $m_1 = m_2$, then $n_2 = \mathbf{0}^{\#E_2}$, which is impossible. If $m_1 \sqsubset m_2$, then ($\mathbf{0}^{\#E_2} \sqsubset n_2$ and $o_1 = o_2$) or ($\mathbf{0}^{\#E_2} \sqsubset n_2$ and $o_1 \sqsubset o_2$). In both cases we see that $l_1 \sqsubset l_2$.

(iii) $E_1 \stackrel{a_1,k_1}{\rightarrow} E_1'$, $F \stackrel{a_2,o_1}{\rightarrow} F'$, $G \equiv E_1' \mid F'$, a is a p.s. of a_1 and a_2, and $l_1 = k_1 \cdot o_1$. Then $E_2 \stackrel{a_1,n_1}{\rightarrow} E_2'$ with $E_1' \stackrel{v}{\sim} E_2'$ and thus $E_2 \mid F \stackrel{a,m_1}{\rightarrow} E_2' \mid F$ with $m_1 = n_1 \cdot o_1$.

$(E_1' \mid F', E_2' \mid F') \in \mathcal{S}$. Suppose $E_2 \mid F \stackrel{b,m_2}{\rightarrow} H$ for some H and some $m_2 \sqsupseteq m_1$. Because $m_1 = n_1 \cdot o_1$ and because $n_1 \neq \mathbf{0}^{\#E_2}$ and $o_1 \neq \mathbf{0}^{\#F}$, the only possibility is that $E_2 \stackrel{b_1,n_2}{\rightarrow} E_2''$, $F \stackrel{b_2,o_2}{\rightarrow} F''$, $m_2 = n_2 \cdot o_2$, b is a p.s. of b_1 and b_2, and $H \equiv E_2'' \mid F''$. If $m_1 = m_2$, then $n_1 = n_2$ and $o_1 = o_2$ and then by Proposition 3.12, $E_1 \stackrel{b_1,k_1}{\rightarrow} E_1''$ and thus $E_1 \mid F \stackrel{b,l_2}{\rightarrow} E_1'' \mid F''$ with $l_2 = k_1 \cdot o_2$. And indeed $l_1 = k_1 \cdot o_1 = k_1 \cdot o_2 = l_2$. If $m_1 \sqsubset m_2$, then we distinguish:

- $n_1 = n_2$ and $o_1 \sqsubset o_2$. Then $E_1 \stackrel{b_1,k_1}{\rightarrow} E_1''$ and $E_1 \mid F \stackrel{b,l_2}{\rightarrow} E_1'' \mid F''$ with $l_2 = k_1 \cdot o_2$. Clearly, $l_1 = k_1 \cdot o_1 \sqsubset k_1 \cdot o_2 = l_2$.
- $n_1 \sqsubset n_2$ and $o_1 = o_2$. In this case $E_1 \stackrel{b_1,k_2}{\rightarrow} E_1''$ with $k_2 \sqsupset k_1$ and $E_1 \mid F \stackrel{b,l_2}{\rightarrow} E_1'' \mid F''$ with $l_2 = k_2 \cdot o_2$. Again $l_1 = k_1 \cdot o_1 \sqsupset k_2 \cdot o_1 = k_2 \cdot o_2 = l_2$.
- $n_1 \sqsubset n_2$ and $o_1 \sqsubset o_2$. Then $E_1 \stackrel{b_1,k_2}{\rightarrow} E_1''$ with $k_2 \sqsupset k_1$ and $E_1 \mid F \stackrel{b,l_2}{\rightarrow} E_1'' \mid F''$ with $l_2 = k_2 \cdot o_2$. Also $l_1 = k_1 \cdot o_1 \sqsubset k_2 \cdot o_2 = l_2$.

This proves condition (i) of Definition 3.8. Condition (ii) follows by a symmetric argument.

For part (2) we will proof that $\mathcal{S} = \{(E_1 \setminus L, E_2 \setminus L) \mid E_1 \stackrel{v}{\sim} E_2\}$ is a v.s.d.t. bisimulation. Let $(E_1 \setminus L, E_2 \setminus L) \in \mathcal{S}$ and assume $E_1 \setminus L \stackrel{a,l_1}{\rightarrow} G$. Then $E_1 \stackrel{a,l_1}{\rightarrow} E_1'$, $G \equiv E_1' \setminus L$ and $a \cap (L \cup \overline{L}) = \varnothing$. But then because $E_1 \stackrel{v}{\sim} E_2$ we have by Proposition 3.12 $E_2 \stackrel{a,n_1}{\rightarrow} E_2'$ with $E_1' \stackrel{v}{\sim} E_2'$ and thus $E_2 \setminus L \stackrel{a,n_1}{\rightarrow} E_2' \setminus L$. Evidently $(E_1' \setminus L, E_2' \setminus L) \in \mathcal{S}$. Now suppose $E_2 \setminus L \stackrel{b,n_2}{\rightarrow} H$ for some H and some $n_2 \sqsupseteq n_1$. Then $E_2 \stackrel{b,n_2}{\rightarrow} E_2''$, $H \equiv E_2'' \setminus L$ and $b \cap (L \cup \overline{L}) = \varnothing$. If $n_1 = n_2$ then (by Proposition 3.12) $E_1 \stackrel{b,l_1}{\rightarrow} E_1''$ and thus, because $b \cap (L \cup \overline{L}) = \varnothing$, $E_1 \setminus L \stackrel{b,l_1}{\rightarrow} E_1'' \setminus L$. If $n_1 \sqsubset n_2$ then $E_1 \stackrel{b,l_2}{\rightarrow} E_1''$ with $l_2 \sqsupset l_1$. And thus $E_1 \setminus L \stackrel{b,l_2}{\rightarrow} E_1'' \setminus L$.

Part (i) of Definition 3.8 is now proven. Part (ii) is proved in an analogous way.

For part (3) we can show that $\mathcal{S} = \{(E_1[f], E_2[f]) \mid E_1 \stackrel{v}{\sim} E_2\}$ is a v.s.d.t. bisimulation. The proof is straightforward and we will not consider it here.

Part (4) is proved by showing that $\mathcal{S} = \{(\mathcal{M}(E_1), \mathcal{M}(E_2)) \mid E_1 \stackrel{v}{\sim} E_2\}$ is a v.s.d.t. bisimulation. Let $(\mathcal{M}(E_1), \mathcal{M}(E_2)) \in \mathcal{S}$ and suppose $\mathcal{M}(E_1) \stackrel{a,l_1}{\rightarrow} G$. Then $E_1 \stackrel{a,l_1}{\rightarrow} E_1'$, $G \equiv \mathcal{M}(E_1')$ and $\neg\ (\exists\, b, m, F : E_1 \stackrel{b,m}{\rightarrow} F : l_1 \sqsubset m)^{(*)}$. But then $E_2 \stackrel{a,n_1}{\rightarrow} E_2'$ with $E_1' \stackrel{v}{\sim} E_2'$. For $\mathcal{M}(E_2) \stackrel{a,n_1}{\rightarrow} \mathcal{M}(E_2')$ to be true, we have to show that there does not exist a transition $E_2 \stackrel{b,n_2}{\rightarrow} E_2''$ such that $n_2 \sqsupset n_1$. Suppose there *does* exist a transition $E_2 \stackrel{b,n_2}{\rightarrow} E_2''$ with $n_2 \sqsupset n_1$. Then, by Proposition 3.12, we have $E_1 \stackrel{b,l_2}{\rightarrow} E_1''$ with $l_2 \sqsupset l_1$. But this contrasts $^{(*)}$, so, $\neg\ (\exists\, b, m, F : E_2 \stackrel{b,m}{\rightarrow} F : n_1 \sqsubset m)^{(**)}$ and thus $\mathcal{M}(E_2) \stackrel{a,n_1}{\rightarrow} \mathcal{M}(E_2')$. Clearly, $(\mathcal{M}(E_1'), \mathcal{M}(E_2')) \in \mathcal{S}$. Assume $\mathcal{M}(E_2) \stackrel{b,n_2}{\rightarrow} H$ for some H and some $n_2 \sqsupseteq n_1$. Then

$E_2 \stackrel{b,n_2}{\rightarrow} E_2''$ and $H \equiv \mathcal{M}(E_2'')$. By $^{(**)}$ we then have that $n_2 = n_1$, so $E_1 \stackrel{b,l_1}{\rightarrow} E_1''$. Using $^{(*)}$ we have $\mathcal{M}(E_1) \stackrel{b,l_1}{\rightarrow} \mathcal{M}(E_1'')$.

This proves part (i) of Definition 3.8. Again part (ii) is proved in an analogous way.

This ends the proof of Proposition 3.14.

□

Proof of Proposition 3.15

It is rather straightforward to prove that $\mathcal{S} = \{\mathcal{M}(E), \phi(\mathcal{M}(E)) \mid E \in \mathcal{E}\}$ is a v.s.d.t. bisimulation.

□

Proof of Proposition 3.16

This property can easily be proved by using Proposition 3.10 and inference rule **dtRes**.

□

References

[Bae86] Baeten, J.C.M.
Procesalgebra: Een formalisme voor parallel, communicerende processen.
Deventer : Kluwer, 1986.

[BB92] Baeten, J.C.M. and J.A. Bergstra
Discrete time process algebra.
University of Amsterdam, Faculty of Mathematics and Computer Science, 1992.
Technical Report, nr. P9208b.

[EVD89] Eijk, P.H.J. van and C. Vissers, M. Diaz.
The formal description technique LOTOS.
Amsterdam : North-Holland, 1989.

[IM76] Isaacson, D.L. and R.W. Madsen
Markov chains : Theory and applications.
New York : Wiley, 1976.

[Koo91] Koomen, C.J.
The design of communicating systems: A system engineering approach.
Dordrecht : Kluwer, 1991.

[Mil80] Milner, R.
A calculus of communicating systems.
Berlin : Springer, 1980.
(Lecture Notes in Computer Science, Vol. 92).

[Mil89] Milner, R.
Communication and concurrency.
London : Prentice Hall, 1989.

[MT90] Moller, F. and C. Tofts
A temporal calculus of communicating systems.
In: Proceedings of CONCUR'90, Amsterdam, The Netherlands, August 27–30, 1990. Ed. by J.C.M. Baeten and J.W. Klop. Berlin : Springer, 1990 (Lecture Notes in Computer Science, Vol. 458). P. 401–415.

[MV90] Mauw, S. and G.J. Veltink
A process specification formalism.
Fundamenta Informatica, Vol. 13(1990), iss. 2, p. 85–139.

[Van94] Van Rangelrooij, A.
Design and evaluation of system architectures for (application-specific) heap management processors using fsCCS and dtCCS.
Instituut Vervolgopleidingen, Eindhoven University of Technology, 1994.
Technical report (available as Technical Report AIO-EB-026 from Digital Information Systems Group, Department of Electrical Engineering, Eindhoven University of Technology, P.O. Box 513, NL-5600 MB Eindhoven, The Netherlands).

[VV94] Van Rangelrooij, A. and J.P.M. Voeten.
CCStool2: An expansion, minimization, and verification tool for finite state CCS descriptions.
Eindhoven : Eindhoven University of Technology, Faculty of Electrical Engineering, 1994.
EUT Report 94-E-284.

[VV94b] Van Rangelrooij, A. and J.P.M. Voeten
CCStool2: An expansion, minimization, and verification tool for finite state CCS descriptions, User's Manual, version 2.4..
Section of Digital Information Systems, Faculty of Electrical Engineering, Eindhoven University of Technology, October 1994.
Internal Sectional Report MAN-EB-9402.

[VV94c] Van Rangelrooij, A. and J.P.M. Voeten
dtCCStool1: An unfolding and expansion tool for dtCCS descriptions, User's Manual, version 1.3..
Section of Digital Information Systems, Faculty of Electrical Engineering, Eindhoven University of Technology, October 1994.
Internal Sectional Report MAN-EB-9403.

[VV94d] Van Rangelrooij, A. and J.P.M. Voeten
PAtool1: A performance-analysis tool for dtCCS descriptions, User's Manual, version 1.1..
Section of Digital Information Systems, Faculty of Electrical Engineering, Eindhoven University of Technology, October 1994.
Internal Sectional Report MAN-EB-9404.

[ST92] Satoh, I. and M. Tokoro
A formalism for real-time concurrent object-oriented computing.
In: Proceedings of the 1992 conference on Object-Oriented Programming, Systems, Languages and Applications (OOPSLA'92), Vancouver, British Columbia, Canada, October 18–22, 1992. Ed. by A. Paepcke. New York : ACM Press, 1992 (ACM Sigplan Notices, Vol. 27/10). P. 315–326.

[Wol91] Wolfram, S.
Mathematica: A system for doing mathematics by computer.
Redwood City : Addison-Wesley, 1991 (second ed.).

Eindhoven University of Technology Research Reports ISSN 0167-9708
Coden: TEUEDE

Faculty of Electrical Engineering

(266) Cluitmans, L.J.M.
USING GENETIC ALGORITHMS FOR SCHEDULING DATA FLOW GRAPHS.
EUT Report 92-E-266. 1992. ISBN 90-6144-266-4

(267) Józwiak, L. and A.P.H. van Dijk
A METHOD FOR GENERAL SIMULTANEOUS FULL DECOMPOSITION OF SEQUENTIAL MACHINES: Algorithms and implementation.
EUT Report 92-E-267. 1992. ISBN 90-6144-267-2

(268) Boom, H. van den and W. van Etten, W.H.C. de Krom, P. van Bennekom, F. Huijskens, L. Niessen, F. de Leijer
AN OPTICAL ASK AND FSK PHASE DIVERSITY TRANSMISSION SYSTEM.
EUT Report 92-E-268. 1992. ISBN 90-6144-268-0

(269) Putten, P.H.A. van der
MULTIDISCIPLINAIR SPECIFICEREN EN ONTWERPEN VAN MICROELEKTRONICA IN PRODUKTEN (in Dutch).
EUT Report 93-E-269. 1993. ISBN 90-6144-269-9

(270) Bloks, R.H.J.
PROGRIL: A language for the definition of protocol grammars.
EUT Report 93-E-270. 1993. ISBN 90-6144-270-2

(271) Bloks, R.H.J.
CODE GENERATION FOR THE ATTRIBUTE EVALUATOR OF THE PROTOCOL ENGINE GRAMMAR PROCESSOR UNIT.
EUT Report 93-E-271. 1993. ISBN 90-6144-271-0

(272) Yan, Keping and E.M. van Veldhuizen
FLUE GAS CLEANING BY PULSE CORONA STREAMER.
EUT Report 93-E-272. 1993. ISBN 90-6144-272-9

(273) Smolders, A.B.
FINITE STACKED MICROSTRIP ARRAYS WITH THICK SUBSTRATES.
EUT Report 93-E-273. 1993. ISBN 90-6144-273-7

(274) Bollen, M.H.J. and M.A. van Houten
ON INSULAR POWER SYSTEMS: Drawing up an inventory of phenomena and research possibilities.
EUT Report 93-E-274. 1993. ISBN 90-6144-274-5

(275) Deursen, A.P.J. van
ELECTROMAGNETIC COMPATIBILITY: Part 5, installation and mitigation guidelines, section 3, cabling and wiring.
EUT Report 93-E-275. 1993. ISBN 90-6144-275-3

(276) Bollen, M.H.J.
LITERATURE SEARCH FOR RELIABILITY DATA OF COMPONENTS IN ELECTRIC DISTRIBUTION NETWORKS.
EUT Report 93-E-276. 1993. ISBN 90-6144-276-1

(277) Weiland, Siep
A BEHAVIORAL APPROACH TO BALANCED REPRESENTATIONS OF DYNAMICAL SYSTEMS.
EUT Report 93-E-277. 1993. ISBN 90-6144-277-X

(278) Gorshkov, Yu.A. and V.I. Vladimirov
LINE REVERSAL GAS FLOW TEMPERATURE MEASUREMENTS: Evaluations of the optical arrangements for the instrument.
EUT Report 93-E-278. 1993. ISBN 90-6144-278-8

Eindhoven University of Technology Research Reports ISSN 0167-9708
Coden: TEUEDE

Faculty of Electrical Engineering

(279) Creyghton, Y.L.M. and W.R. Rutgers, E.M. van Veldhuizen
IN-SITU INVESTIGATION OF PULSED CORONA DISCHARGE.
EUT Report 93-E-279. 1993. ISBN 90-6144-279-6

(280) Li, H.Q. and R.P.P. Smeets
GAP-LENGTH DEPENDENT PHENOMENA OF HIGH-FREQUENCY VACUUM ARCS.
EUT Report 93-E-280. 1993. ISBN 90-6144-280-X

(281) Di, Chennian and Jochen A.G. Jess
ON THE DEVELOPMENT OF A FAST AND ACCURATE BRIDGING FAULT SIMULATOR.
EUT Report 94-E-281. 1994. ISBN 90-6144-281-8

(282) Falkus, H.M. and A.A.H. Damen
MULTIVARIABLE H-INFINITY CONTROL DESIGN TOOLBOX: User manual.
EUT Report 94-E-282. 1994. ISBN 90-6144-282-6

(283) Meng, X.Z. and J.G.J. Sloot
THERMAL BUCKLING BEHAVIOUR OF FUSE WIRES.
EUT Report 94-E-283. 1994. ISBN 90-6144-283-4

(284) Rangelrooij, A. van and J.P.M. Voeten
CCSTOOL2: An expansion, minimization, and verification tool for finite state CCS descriptions.
EUT Report 94-E-284. 1994. ISBN 90-6144-284-2

(285) Roer, Th.G. van de
MODELING OF DOUBLE BARRIER RESONANT TUNNELING DIODES: D.C. and noise model.
EUT Report 95-E-285. 1995. ISBN 90-6144-285-0

(286) Dolmans, G.
ELECTROMAGNETIC FIELDS INSIDE A LARGE ROOM WITH PERFECTLY CONDUCTING WALLS.
EUT Report 95-E-286. 1995. ISBN 90-6144-286-9

(287) Liao, Boshu and P. Massee
RELIABILITY ANALYSIS OF AUXILIARY ELECTRICAL SYSTEMS AND GENERATING UNITS.
EUT Report 95-E-287. 1995. ISBN 90-6144-287-7

(288) Weiland, Siep and Anton A. Stoorvogel
OPTIMAL HANKEL NORM IDENTIFICATION OF DYNAMICAL SYSTEMS.
EUT Report 95-E-288. 1995. ISBN 90-6144-288-5

(289) Konieczny, Pawel A. and Lech Józwiak
MINIMAL INPUT SUPPORT PROBLEM AND ALGORITHMS TO SOLVE IT.
EUT Report 95-E-289. 1995. ISBN 90-6144-289-3

(290) Voeten, J.P.M.
POOSL: An object-oriented specification language for the analysis and design of hardware/software systems.
EUT Report 95-E-290. 1995. ISBN 90-6144-290-7

(291) Smeets, B.H.T. and M.H.J. Bollen
STOCHASTIC MODELLING OF PROTECTION SYSTEMS: Comparison of four mathematical techniques.
EUT Report 95-E-291. 1995. ISBN 90-6144-291-5

(292) Voeten, J.P.M. and A. van Rangelrooij
CCS AND TIME: A practical and comprehensible approach to a performance evaluation of finite state CCS descriptions.
EUT Report 95-E-292. 1995. ISBN 90-6144-292-3