The University of Edinburgh

Department of Computer Science

Combinators and Bisimulation Proofs for Restartable Systems

by

K.V.S. Prasad

CST-77-91
(also published as ECS-LFCS-91-149)

James Clerk Maxwell Building,
The King's Buildings,
Mayfield Road,
Edinburgh,
EH9 3JZ.

April, 1991

Combinators and Bisimulation Proofs for Restartable Systems K.V.S. Prasad

Combinators and Bisimulation Proofs for Restartable Systems

K. V. S. Prasad

Doctor of Philosophy
University of Edinburgh
December 1987

(graduation date July 1988)

A dissertation for the
Ph.D. degree in Computer Science at
The University of Edinburgh

Department of Computer Science, University of Edinburgh, Edinburgh EH9 3JZ, Scotland.

Submission: December 1987. Graduation: July 1988. Printing: March 1991.

Author's current address: Dept. of Computer Sciences, Chalmers University of Technology, S-412 96 Göteborg, Sweden.

Email: prasad@cs.chalmers.se

Declaration:
This thesis has been composed by myself, and the work is my own, done under the guidance of my supervisor, Robin Milner. The Appendix appeared as Technical Report CSR-178-84, Department of Computer Science, University of Edinburgh. Otherwise the work is previously unpublished.

Preface: three years on

Apart from the shortened Appendix and the incorporation of a few minor corrections suggested by my examiners Colin Stirling and Mike Shields, this report is what was submitted and accepted as my Ph.D. thesis. For a variety of personal reasons, I was not able to find the enthusiasm to organise the printing of this report before now. I apologise for this to all concerned.

I am not aware of any major advances over the last three years in the area this thesis deals with. But the reader will notice that the references are quaint. Milner's book, *Communication and Concurrency*, Prentice-Hall, 1989, has since appeared, and most references to [Mil80] can be taken to refer to the new book instead. Similarly, the material needed from [Lar86] is more easily accessible in Larsen's paper, *A Context Dependent Equivalence between Processes*, *TCS 49*, 1987.

Of the Appendix to the submitted thesis, I have included only a few pages. The relevance of the Appendix to the rest of the thesis is explained in Sections 1.2.3 and 1.3. Please note that the bibliography for the main body of the thesis appears *before* the Appendix.

Göteborg
21 March 1991

Abstract

During recent years, calculi for reasoning about concurrent systems have been developed; examples are CCS (Calculus of Communicating Systems), CSP (Communicating Sequential Processes) and ACP (Algebra of Communicating Processes). Their theory has been studied intensively; rather less has been done in applying these calculi to problems of significant size. The reported work in this direction suggests that we should be prepared to develop new formal systems and notation for the problems we attack, and also that the sheer mechanics are difficult enough that we need new techniques to organise and structure proofs.

In this thesis we study bisimulation proofs, applied to a small set of related problems. We define a class of new operators to capture the structure in these problems, and use them to produce significantly smaller and clearer proofs than were previously possible. In order to avoid complexities due to interleavings, we use a new calculus, MCCS, that can be seen as lying between CCS and ASCCS (a variant of CCS); like the latter, MCCS allows synchronisation but is not a synchronous calculus.

We begin by summarising the new layout for bisimulation proofs developed for a proof of a simple fault tolerant system carried out in CCS. This proof forms the Appendix to the thesis.

Next, we define MCCS and study its properties, deriving most of them from the related calculi, SCCS, ASCCS and CCS. This study also clarifies the

relations between these older calculi, finding them to be closer than had been previously expressed in [Mil83]. In the process, we contribute to the study of ASCCS, the least studied of the three.

Our new operators are designed specifically to help analyse the structure of systems into normal and exception processing; the major theorem in each case is the Synchronised Displacement Theorem. We comment on other possible design choices. We use the new operators and MCCS to reformulate and prove a variant of the example in the Appendix.

A natural question that arises when we deal with new operators is whether they are derivable from the old. We study this matter, contributing a new technique for proving operators non-derivable. We prove many of the operators in SCCS and CCS to be independent, i.e. not definable in terms of the other operators.

Acknowledgements

I cannot express sufficiently how much I owe to my supervisor and teacher, Robin Milner. Many of the ideas here are his, or arose in discussion with him, and certainly any mathematical elegance the work may have owes to him. I thank him for inspiration, guidance, patience, and constant encouragement and support.

The work reported here also owes to many other people, more than I can list. To name but a few of those in the concurrency groups at Edinburgh and Göteborg: Kim Larsen, Sören Holmström, Nik Traub, Ilaria Castellani, Colin Stirling, Matthew Hennessy, Joachim Parrow, Uno Holmer, Wang Yi, and Michael Hedberg. The work has also been discussed with many others, particularly Glynn Winskel, Tony Hoare, Roland Backhouse, and everyone at the Computer Science group at TIFR, Bombay. And to name some who are not officially in the same field, but from whom I learnt a great deal: Allen Stoughton, Alistair Sinclair, Edmund Robinson, Lincoln Wallen, Peter Dybjer and Dan Synek.

I thank everyone at the Departments of Computer Science both at the University of Edinburgh and at Chalmers University of Technology, Göteborg, for providing such friendly and stimulating environments.

The work was supported by a Postgraduate Studentship and subsequently a part-time job at the University of Edinburgh, and then by a research position at Chalmers.

At a more personal level, I owe once again so much to so many people that I cannot express my thanks to all here, but I owe most of all to Rita, Jan, Allen, Ilaria, Alistair, Ursula and Marie. Thank you all for your love and friendship.

The one constant factor through all the changes in my life, particularly over the last few years, has been the love and affection I have received from afar, from my parents and my sisters. They have had faith even when I have lacked it. To them this thesis is dedicated.

Contents

1

Introduction

1.1 Modelling concurrent systems

The study of concurrent systems has advanced greatly over the past few years, but is still in a very confused state compared to the study of sequential systems. This is to be expected, given the greater complexity of behaviour concurrent systems can show. The present situation is characterised most strikingly by a profusion of models and methods for reasoning about them. Thus the best thing to do when trying to solve a practical problem that seems to have to do with concurrency is to look around at the available theories, and use the one that seems best suited. Net theory, ADA, and the various algebraic calculi now available are all so different from one another that the point needs no further emphasis.

The concurrent programs we can formally reason about and prove are all much smaller than the sequential programs we can handle. This is one factor common to all theories of concurrency. As researchers, we therefore do not expect to make a contribution to "practice" in the sense of providing tools directly applicable to the building of large concurrent systems. We hope only to provide more insight than was previously available.

In that spirit, there have been a large number of problems analysed over the years: various timing and synchronisation problems, consistency problems for distributed data bases, and so on. A common factor here is that while light is shed on the problem, even more is usually shed on the method of

analysis. The problems can usually be described quite reasonably in English, even though it is obviously dangerous to reason with it. Whether they can be described at all in any given theory is usually not obvious to start with, and what the problem will look like once it has been formulated is of course even less obvious. Goethe said: "Mathematicians are like Frenchmen: whenever you say something to them, they translate it into their own language, and at once it is entirely different." (Quoted by Don Knuth in his TEXbook). The same could certainly be said of what concurrency theoreticians do to practical problems.

With these preliminaries, let us come to the theory and the problem this thesis deals with.

1.1.1 Algebraic calculi of processes

In recent years, several algebraic calculi of processes have been developed, such as CCS (Calculus of Communicating Systems)[Mil80], [Mil83], CSP (Communicating Sequential Processes) [Hoa85] and ACP (Algebra of Communicating Processes) [BK86]. Even these show striking differences from one another, such as for example in the large number of different equivalences proposed, and in whether the semantics is given by axioms or operationally, etc. All have been studied intensively from theoretical viewpoints, but rather less has been done to apply them to problems of significant size. Since they are mathematically appealing and now substantial theories, they seem obvious choices for attempts at application.

The work that has begun in that direction, for example [Bac83], [HH86], [LM86], [SW83], and systems such as CIRCAL [Mil85] and LOTOS [OIS85], have all yielded feedback about the theoretical systems used. The common discovery here is that we should be prepared to develop new formalisms and notation to suit the problem at hand—it is unlikely that ready-made ones will serve. With each such new formalism or notation, experience with different problems will tell how flexible or general it is. As for the theoretical framework, the question is whether or not it allows such new formalisms to be easily developed, continuing to provide the foundation on which the work proceeds.

In this thesis, we study one of these calculi, CCS, concentrating entirely on one particular proof method, bisimulation [Par81]. (Formal definitions follow later on). Bisimulation is an operational equivalence between processes, or *agents* as we shall call them. They do not have to be CCS agents, but can belong to any sort of labelled transition system, [Plo81]. If two agents are in a bisimulation, each can match the moves of the other, and such matching

moves take the agents into states that are again in bisimulation. The proof technique therefore consists merely of producing a relation and checking that it is a bisimulation. Sometimes, indeed, one need do no more than produce the relation and leave it to the reader to carry out the check. There are also mechanical methods for producing and checking bisimulations.

Here we are concerned not with systems that are simple enough that it is enough to produce a relation by inspection, nor with bisimulations that are so large and unstructured that only mechanical aids make them tractable. We study in this thesis how to use bisimulation as an acceptable method of proof that can be communicated from one person to another, like any other mathematical proof. Thus we shall usually show even for small bisimulations, why they are indeed bisimulations, i.e., we shall carry out the check. The layout we develop for displaying the bisimulation and the check is an important part of the contribution this thesis makes. The other part of the story is that we shall not be content to display large bisimulations. Instead, we try to break them down into understandable bits. This shows up the structure in the agents being studied. The reader will recognise that this is no more than standard mathematical practice.

We now describe the problem we model and analyse. As we have indicated, the end result is more a study of bisimulation proof techniques than a study of the problem itself, which in retrospect is a little like the stone on which the tools were honed.

1.1.2 Restartable systems

The area we borrow our simple problem from is Fault Tolerance. This is a vast area, [AL81], and no unifying methodology or formalism is in sight. Many of the issues here in fact seem to defy formalisation. However, it is a subject of obvious importance, particularly for distributed systems. Timing and synchronisation problems, as well as consistency or consensus problems, all have elements of fault tolerance built into them. Here we shall be concerned with a different problem, that of *restartable systems.*

We use the term to mean a deterministic system that is equipped with reliable storage, so that its state can be restored after failure. We shall assume throughout that the failure being modelled is caused by a transient hardware fault, so that it makes sense to restart the program. (Though anyone who has re-booted an operating system knows that it sometimes makes sense to re-run a program that is known to have bugs in it!)

Various versions of such systems have long been studied, for example [PB77],

and the idea is intuitively obvious. Operating systems have long provided routines that dump the current state upon trouble, in order to continue afterwards where they left off. Many text editors provide a log file (or "audit trail") mechanism for error recovery. All the user's commands are logged in a file, and if the current editing session has to be abandoned for any reason, the user can recover most of the lost work simply by running the editor through the commands in the log file. An important efficiency consideration is to allow *checkpoints*, where the user writes out the current state of the editing buffer, so that all of the log file upto that point can be thrown away. After future failures there will not be so much recovery to do.

An important idea that we shall use in conjunction with the log-file mechanism is that of *failstop* processes [SS81]. These are merely processes that fail in a very benign way, simply stopping altogether rather than put out incorrect information.

These ideas were used in connection with a message switching system and reported in [PRA82]. The key idea was to quarantine a failed process; it was allowed to recover by feeding it back past messages secretly, and its own output during recovery was filtered out. Thus other processes in the system observed no more than a longer delay than usual before they got their responses. Failstop-ness was approximated because it turned out to be possible to identify failed processes very soon after failure. Thus minor patches to other processes sufficed to limit damage. Another important observation was that a large number of messages did not have to be logged because they did not alter the state of the receiver.

These are all the ideas we shall need of fault tolerance, error recovery, state restoration, failstop processes and the like. "Fault tolerant system" and "restartable system", in particular will be used interchangeably throughout the thesis, but they both mean no more than the very simple, restricted kind of systems we shall formally define. The terms are merely labels, and should not be taken to suggest wider applicability of our techniques. That is of course to be hoped for, but at present we can say no more than that.

1.1.3 Bisimulation proofs for restartable systems

The message switching application lends itself well to modelling in CCS. This was carried out in [Pra84], important design decisions on the way being how to model failure, and how to detect the failure. These decisions have survived unchanged upto the present; we describe them in detail in the next section. With the modelling done, the problem was formulated thus: Is the restartable

system (called "FTS") equivalent to a perfect system that never fails? And if so, what is the equivalence?

The answer, also reported in [Pra84], is that the two systems are in a *weak bisimulation*, a form of bisimulation that ignores internal moves, such as the secret feeding back of messages to the quarantined recovering process. This means that an observer can see no difference between the two, thus agreeing with our experience with the message switching system.

The proof, even after much effort to break it down, remained long and unsatisfactory in that it failed to display much structure, though there is plenty to be intuitively seen in the system. It does not seem at the moment that one can do much better than that in classical CCS, and hence the original proof appears as an Appendix to this thesis.

The main body of the thesis develops two new tools to attack this problem, and solve it satisfactorily. The first is a new calculus, MCCS, a variant of CCS that permits synchronisation. This allows us to model as synchronous actions, many events that we would be prepared to describe in English as "happening simultaneously" or corresponding to the notion of "atomic action" in other formal systems. We thus avoid a great many intermediate states that arise out of the interleaving forced by CCS, and are of no intuitive significance.

SCCS of course allows synchronisation, but is of no use to us since our problem is basically asynchronous. Of the established calculi, MCCS is actually closest to ASCCS, an asynchronous subcalculus of SCCS. In order to establish the properties of MCCS, we study the relationship between MCCS and ASCCS, and between MCCS and CCS. We also study the equational laws and congruences of MCCS.

The second tool is a set of combinators that allow us to separate the normal and exceptional behaviour of agents. We define these combinators and study their equational properties, the most important of which is that they factorise FTS into its natural divisions. MCCS is important here too, because our basic factorisation is expressed by a theorem we call the Synchronised Displacement Theorem (SDT), which needs synchronous actions for its formulation. Roughly, the equivalence set up by the SDT is valid only in contexts that synchronise failure and discovery. Thus we need a version of *context dependent bisimulation*, a theory developed by [Lar86], though only an extremely simple special case that we can set up briefly.

With these tools, we solve the FTS problem satisfactorily, showing up all the structure one expects intuitively. A bonus is that most of the work turns

out to be about data structures, thus isolating the concurrency component of the problem.

Finally, a natural question when new combinators (operators) are proposed is whether they are definable from the existing ones. We are led to study this generally, and discover a method to prove operators *non*-definable. Several operators of CCS are then proved non-definable from the others.

We are now ready to look at these ideas in a more concrete setting.

1.2 Motivation

We begin with a look at a toy fault tolerant system to set the scene and to motivate the formal apparatus we will be introducing in the subsequent chapters. We assume a familiarity with CCS, [Mil80] or [Mil83]. "CCS" in this thesis usually means specifically the calculus introduced in [Mil80], but occasionally, it means any of the calculi SCCS, ASCCS, CCS (all from [Mil83]), or MCCS (to be introduced later). Formal descriptions of all these calculi appear later.

1.2.1 A toy FTS in different calculi

"FTS" abbreviates the phrase "fault tolerant system", but really refers to various kinds of simple restartable systems, all related.

Consider a deterministic agent P, and let Q be the behaviour of an implementation of P running on unreliable hardware. We assume that the failures of the hardware are of a particularly benign kind, resulting merely in stoppage rather than incorrect computation. We also assume that Q emits a *failure signal*, f, or, what comes to the same thing, that Q fails in response to a f^{-1} signal from a *demon agent*, F. Thus Q behaves just like P except that it may, at any stage, do an f and stop. Q is said to be a *fail-stop* version of the *perfect agent*, P. The idea of fail-stop agents was first described by [SS81]. We are interested in fail-stop agents because they never send out erroneous information, so errors do not propagate in a system consisting only of fault-free or fail-stop agents.

Here is an example in CCS:

$P \Leftarrow a.b.P$

$Q \Leftarrow f.0 + a.b.Q$

$F \Leftarrow f^{-1}.F$

The left pointing double arrow means "is defined to be". The perfect agent P does a's and b's alternately. Our fail-stop Q mimics P, until it fails. It refuses to fail after an a, a subtlety we shall return to later.

Now consider error recovery by means of a fault-free *backup agent*. With such a simple P and Q, all the backup agent need do, when it receives news of Q's failure, is spawn a new Q to replace the failed one. The failure of a fail-stop agent can only be detected by recognising that it did not communicate when it should have done, i.e. by time-outs. These cannot be programmed in CCS, so we introduce an action g to stand for "time-out". There is nothing to be gained by introducing another agent, so we have the demon agent send out the detection signal as well. We can think of g as "information about failure".

We call the whole system T; remember that we have changed F:

$$P \Leftarrow a.b.P$$
$$Q \Leftarrow f.\mathbf{0} + a.b.Q$$
$$S \Leftarrow g.(S \mid Q)$$
$$F \Leftarrow f^{-1}.g^{-1}.F$$
$$T \Leftarrow (Q \mid S \mid F) \setminus\setminus \{f, g\}$$

Can we prove the toy system T equivalent to P? Yes, by the expansion theorem:

$$\begin{aligned} T &\sim a.b.T + 1.\left(\mathbf{0} \mid S \mid g^{-1}.F\right) \setminus\setminus \{f, g\} \\ &\sim a.b.T + 1.1.(\mathbf{0} \mid S \mid Q \mid F) \setminus\setminus \{f, g\} \\ &\sim a.b.T + 1.1.T \end{aligned}$$

and a simple bisimulation now shows that $T \approx P$.

Those familiar only with [Mil80] will have noticed that the notation has changed: we use roman letters rather than greek for actions, write 1 instead of τ, f^{-1} instead of $\overline{f}$, $\mathbf{0}$ instead of NIL, and $\setminus\setminus\{f,g\}$ instead of $\setminus\{f,g\}$. The latter now means only f and g are restricted away, not also f^{-1} and g^{-1}. Our notation is from [Mil83] and we will use it for all the calculi we look at. A more serious updating is that we now use bisimulations to define both strong congruence and observational equivalence. Formal definitions follow later on in this chapter.

Now, S might time out before Q fails. While we cannot express time-outs in CCS, we can at least recognise that g might occur before f:

$$F \Leftarrow f^{-1}.g^{-1}.F + g^{-1}.f^{-1}.F$$

We can also interpret the second term here as warning of impending failure.

If we try to carry out our previous proof now, we fail! This is because S will spawn the new Q right away, which might start work before the old one fails, producing two consecutive a's. We have run into our first protocol problem; F and S must synchronize after the failure:

$$Q \Leftarrow f.\mathbf{0} + a.b.Q$$
$$S \Leftarrow g.h.(S \mid Q)$$
$$F \Leftarrow f^{-1}.g^{-1}.h^{-1}.F + g^{-1}.f^{-1}.h^{-1}.F$$
$$T \Leftarrow (Q \mid S \mid F) \backslash\backslash \{f,g,h\}$$
$$T' \Leftarrow \left(Q \mid h.(S \mid Q) \mid f^{-1}.h^{-1}.F\right) \backslash\backslash \{f,g,h\}$$

where T' is the state after time-out but before failure.

$$\begin{aligned} T &\sim a.b.T + 1.\left(\mathbf{0} \mid S \mid g^{-1}h^{-1}.F\right) \backslash\backslash \{f,g,h\} + 1.T' \\ &\sim a.b.T + 1.1.(\mathbf{0} \mid S \mid Q \mid F) \backslash\backslash \{f,g,h\} + 1.T' \\ &\sim a.b.T + 1.1.T + 1.T' \end{aligned}$$

and

$$T' \sim a.b.T' + 1.1.T$$

The rest follows by bisimulation; we get $P \approx T \approx T'$.

We shall not pursue this example further. But let us see what happened. The protocol problem did not show up earlier because when f happens first, g follows immediately; if g is allowed to happen first, we have to ensure it has no effect until f occurs; S has to be held up. This example is too small to be troublesome, but programming errors become increasingly likely with larger ones. For example, other orderings of the actions of F can result in deadlocks. The reader can see the beginnings of a combinatorial blow up in the number of states because of the different interleavings of actions.

Synchronisation is one way to avoid this unplanned protocol. Consider the following program in SCCS:

$$P \Leftarrow a{:}\, b{:}\, P$$
$$Q \Leftarrow f{:}\, \mathbf{1} + a{:}\, b{:}\, Q$$
$$S \Leftarrow g{:}\, (S \times Q) + 1{:}\, S$$
$$F \Leftarrow f^{-1}g^{-1}{:}\, F + 1{:}\, F$$
$$T \Leftarrow (Q \times S \times F) \backslash\backslash \{f,g\}$$

The proof is

$$\begin{aligned} T &\sim a{:}\, b{:}\, T + 1{:}\, (\mathbf{1} \times (S \times Q) \times F) \backslash\backslash \{f,g\} \\ &\sim a{:}\, b{:}\, T + 1{:}\, T \end{aligned}$$

A brief explanation for those unfamiliar with SCCS: In $A \times B$ both agents have to act together, and the resulting action is the product of their individual actions. So $A \times \mathbf{0} \sim \mathbf{0}$. We have $a \times 1 = a$ for all actions a, and $\mathbf{1}$ is the agent that does only 1's. So $A \times \mathbf{1} \sim A$. Note the $\mathbf{1}$ after failure; if we wrote $\mathbf{0}$ instead, the system would deadlock after the first failure.

f and g are synchronised, so there is no protocol needed. We could now program the time-out, but haven't done so; this is not the issue at hand, and

we shall not be using SCCS anyway. There are improvements (no unplanned protocol, no new states due to interleaving of actions), but we have had to acknowledge the synchrony by putting in $1{:}F$ and $1{:}S$ to mark time, and this could vastly increase the number of terms in our agents when we try to model a primarily asynchronous system. All the systems we look at in this thesis are primarily asynchronous.

This "marking time" problem is what ASCCS sets out to tackle, so let's write out our toy system in it:

$$P \Leftarrow a.b.P$$
$$Q \Leftarrow f.\mathbf{0} + a.b.Q$$
$$S \Leftarrow \delta(g.(S \times Q))$$
$$F \Leftarrow \delta\left(f^{-1}g^{-1}.F\right)$$
$$T \Leftarrow (Q \times S \times F) \setminus\setminus \{f, g\}$$

Explanation: δA does an indefinite number of 1's (including possibly none) before behaving like A. The agent $a.A$ does an a right away and then behaves like δA. If we hadn't put δ's on F and S, the system would have been forced to start with a failure. We could also have put delays on the P and Q. The proof is like that for SCCS.

ASCCS is obviously very different from the other calculi. We conclude that it is not quite what we want. More reasons are given later, but here we note only that there is a lot of subtlety about delays that we have to get used to before we can work easily with ASCCS.

Wouldn't it be lovely if we could write, in a calculus that looked very like CCS except that it allowed product actions, the following definitions and proof?

$$P \Leftarrow a.b.P$$
$$Q \Leftarrow f.\mathbf{0} + a.b.Q$$
$$S \Leftarrow g.(S \mid Q)$$
$$F \Leftarrow f^{-1}g^{-1}.F$$
$$T \Leftarrow (Q \mid S \mid F) \setminus\setminus \{f, g\}$$
$$T \sim a.b.T + 1.(\mathbf{0} \mid (S \mid Q) \mid F) \setminus\setminus \{f, g\}$$
$$\sim a.b.T + 1.T$$

Indeed it would, and the next chapter is devoted to establishing such a calculus, MCCS. $A \mid B$ in MCCS means that A or B can act alone as in CCS, or that they can act together as in SCCS.

Such an operator can be written as a derived operator in ASCCS, and [Mil83] in fact uses it in defining a subcalculus called the particulate calculus, which is

isomorphic to CCS. Even with this derived operator, we would still rather not use ASCCS because of the delays, and also the problem that $\approx^c$, the largest congruence contained in $\approx$, is significantly different from CCS. In contrast, MCCS is like CCS and SCCS in that there are no implicit delays, and the congruence looks just like that for CCS as well. The new calculus can be seen as a generalisation of the particulate calculus or as lying between the particulate calculus and ASCCS. A disadvantage that it shares with ASCCS is that the expansion theorem has even more terms than in CCS, because all products are allowed, not just inverses producing 1's. In practice, this has to be controlled by restriction, as above.

Originally the name MCCS was supposed to stand for "mixed CCS", but there is no general agreement about terms like "mixed calculus" and "interleaved calculus" and so on, so the M in MCCS is now never expanded. It is fair to say right away that from the user's point of view the most dramatic differences between the new calculus and the old ones are those we have already sketched above; MCCS is a minor variant. However, there are a lot of details to be filled in, and the relations between the various calculi are fascinatingly subtle. It is better to deal with these subtleties once and for all in a theoretical framework, rather than have them turn up unexpectedly in practice. This we do in Chapter 2.

We should also say that the specific example we have given here could also be dealt with by some sort of **join** operator, where if P and Q can each do a, so can $P\mathbf{join}Q$.

DEFINITION 1.2.1. *The operational semantics of the* ***join*** *operator.*

$$\frac{E \xrightarrow{a} E' \quad F \xrightarrow{a} F'}{E\mathbf{join}F \xrightarrow{a} E'\mathbf{join}F'}$$

□

Indeed this is one of the operators in CSP, and is also taken to be part of CCS by [Lar86]. The γ-conjunction operator in [Mil83], which behaves like synchronous product $E \times F$ except that it requires each γ-action of E to synchronise with a γ-action of F, can also be considered for use.

DEFINITION 1.2.2. *The operational semantics of the γ-conjunction operator.*

$$\frac{E \xrightarrow{a} E' \quad F \xrightarrow{b} F'}{E\&_\gamma F \xrightarrow{ab} E'\&_\gamma F'} \qquad \frac{E \xrightarrow{a\gamma} E' \quad F \xrightarrow{b\gamma} F'}{E\&_\gamma F \xrightarrow{ab\gamma} E'\&_\gamma F'}$$

In both rules, a and b are actions with no γ or γ^{-1} particles

□

A generalisation, Γ-conjunction, requires synchronisation on any $\gamma \in \Gamma$. By setting $\Gamma = Act$, we can get it to behave exactly like **join**. We shall have occasion to refer to these last two in a different connection later on, but for now we point out that in our reworking of FTS in MCCS, we do use the more general possibilities of synchronous composition, and find that it helps to cut down the size of the problem. Thus we choose not to use the **join** operator.

1.2.2 Checkpoint operators

Consider the following definitions in MCCS.

$P \Leftarrow a.b.P$
$Q \Leftarrow f.\mathbf{0} + a.(b.Q + f.\mathbf{0})$
$S \Leftarrow g.(S \mid Q)$
$F \Leftarrow f^{-1}g^{-1}.F$
$T \Leftarrow (Q \mid S \mid F) \backslash\backslash \{f, g\}$

This is exactly the same system as we had earlier, except that we have changed the definition of Q. (Previously, we had $f.\mathbf{0} + a.b.Q$.) The new Q, compared to the old, accepts failures in any state rather than just alternate states.

But now we no longer have $T \approx P$, because S will always start up Q in its first state regardless of where it failed. S has to keep track of Q, perhaps by direct communication, so that it will start it up in the proper state.

If we try to express all this using our existing notation, we will find the number of states alarming, considering how small the example is. A glance at [Pra84] (included as an Appendix to this thesis) where a simple but general FTS is worked out in ordinary CCS, will give an idea of how bad matters can get. With the benefit of hindsight, we observe that our problem is that we have no way of elucidating the structure of the system in the expressions we write down. How helpful such elucidation can be is shown by the chaining operator in [Mil80].

The structure we wish to capture is that of normal execution versus exception handling. Suppose we invent an operator $\dagger$ to do this. For example,

$Q \Leftarrow Q_N \dagger Q_E$ *where* $Q_N \Leftarrow a.b.Q_N$ *and* $Q_E \Leftarrow f.\mathbf{0}$ *i.e.,* $Q \equiv Q_N \dagger f.\mathbf{0}$

Q_N describes the behaviour of Q in the absence of faults; it behaves like the perfect system. The idea is that Q should now be able to fail after a as well as after b, an action by Q_E constituting failure. $\dagger$ has to be defined to express this structure: Q_N acts until failure, when Q_E takes over the behaviour of the entire expression.

We had

$$Q \Leftarrow f.\mathbf{0} + a.b.Q$$
$$S \Leftarrow g.(S \mid Q)$$
$$F \Leftarrow f^{-1}g^{-1}.F$$
$$T \Leftarrow (Q \mid S \mid F) \setminus\setminus \{f, g\}$$
$$T \sim a.b.T + 1.(\mathbf{0} \mid (S \mid Q) \mid F) \setminus\setminus \{f, g\}$$
$$\sim a.b.T + 1.T$$

With the redefined Q, we expect

$$T \sim T_N \leftthreetimes 1.T \qquad \text{, where } T_N \Leftarrow a.b.T_N$$

So T does a's and b's until it hiccups (internal failure and recovery seen externally as 1), and starts over. Because we are not keeping track of Q's state, we can get sequences like a, b, a, 1, a, ... from T. So an observer can see consecutive a's. We will worry about this later, but first how do we prove that T thus defined is equivalent to $(Q \mid S \mid F) \setminus\setminus \{f, g\}$ with the new definition of Q? Obviously, we would like to do this using algebraic laws for $\leftthreetimes$.

The key here is to guess that the behaviour of $Q \mid S$ can now be expressed as $Q_N \leftthreetimes fg.(Q \mid S)$. We are looking for a theorem of the form

$$\text{If } \; A \Leftarrow A_N \leftthreetimes f.A \; \text{ then } \; \left(A \mid \left(f^{-1}\right)^*\right) \setminus\setminus \{f\} \approx A_N \leftthreetimes 1.A_N$$

Indeed we will have such a theorem later.

It is more difficult to see how to prove correct our guess about $Q|S$. We could ask if $\mid$ distributes over $\leftthreetimes$; it does not. The key is to write S as $\mathbf{0} \leftthreetimes g.(S \mid Q)$, and to look for a theorem of the form

$$(A_N \leftthreetimes A_E) \mid (B_N \leftthreetimes B_E) \sim (A_N \mid B_N) \leftthreetimes (A_E \mid B_E)$$

to separate normal and exception behaviour throughout. In its full blown version, this *Synchronised Displacement Theorem* (SDT) is the major result of our work with *checkpoint operators* of which $\leftthreetimes$ is our first example.

This is about as far as we can go before formal definitions become necessary to make any sense of our story. We will say only a little more so as to be able to give an overview of the thesis.

To reprogram our system to give us a sequence of alternating a's and b's, one way is to relabel Q so that it sends its output to a new S that picks them up, thus allowing it to keep track of Q's state. S then relays the output on to the outside world. We will do this in a later chapter. For now we just point out that if the *restart* state is to depend on the state of the normal processing, we must allow the exception agent to notice what the normal agent is doing. If we define a new operator to do this, we don't have to program this explicitly:

$$Q \Leftarrow Q_N \dagger_{\mathcal{C}} Q_E$$
$$Q' \Leftarrow b.Q_N \dagger_{\mathcal{C}} Q'_E$$
$$Q_E \Leftarrow f.Q + a.Q'_E$$
$$Q'_E \Leftarrow f.Q' + b.Q_E$$

where the set $\mathcal{C} = \{a, b\}$. This new operator has a third derivation rule in addition to the two of $\dagger$. It says that whenever both normal agent and the exception agent have an action in the subscript set, the whole agent may do a combined action, while both normal and exception parts evolve. Failure now can be made to yield a different restart state after such an action.

We define two such operators later that look like $\dagger_{\mathcal{C}}$ above, and find the conditions under which the corresponding generalised SDT holds. This programme turns out to motivate essentially all the work in the thesis. Before we give an overview, we look at the example that started it all.

1.2.3 FTS without MCCS and checkpoint operators

The FTS formulated in CCS in the Appendix is more general than the toy system above. Since it motivated the work in the main body of the thesis, we shall say a little more about it here.

Error recovery for any deterministic fail-safe agent is possible using checkpoints and audit trails of its transactions; the "deterministic" refers to the fault-free behaviour, not to the occurrence of failures! As in our toy system, we assume that the necessary backup agents are fault-free. A generalisation is to allow P and Q to accept input. S must then be generalised to a *stable storage* agent that eavesdrops on all inputs to Q. When S starts up a new incarnation of Q, it passes it anew any input that the older incarnation consumed before failing. For a system generating more than a predictable sequence of a's and b's, we need another backup agent: a *duplication suppressor*, D, to filter out output from Q while it is recovering from failure. It is intuitively clear that $(Q \mid S)$, $(Q \mid S \mid D)$ and $(Q \mid S \mid D \mid F)$, with internal labels restricted away, should be strongly related to P in some sense, with the last being intuitively *equivalent* to P, though again the precise equivalence was not obvious to start with.

FTS is observationally equivalent to the perfect system; with hindsight, we can say that this could have been guessed from our toy system, but the full FTS result came first. In any case, this is rather a strong result at first sight, because observational equivalence is a very strong equivalence. In fact, we found that observational equivalence is particularly suited to describing the relation between the two. For example, testing equivalences [dNH84] are not

suitable because they distinguish FTS from the perfect system—because of the possibility of repeated failure, the former is capable of infinite chatter while the latter is not. Observational equivalence ignores this difference, but is otherwise much stronger than testing equivalences. Thus if we have $P \approx Q$, it would be unnecessarily weak to describe the relation between P and Q in terms of testing equivalences, though we would want to assert that both hold if that were the case; it is not, in our example. We have a situation where we can use a strong relation which happens to be lax in just the one case we need.

We have said that the proof in CCS [Pra84] was difficult to do, and long and tedious to read. We can now see that among other things, this is because we did not have the notation to capture features of our agents, and prove them once and for all. As a result, we have comparatively large chunks of proof that do not reveal all of the underlying structure. We also have very large numbers of states because of interleaving, and the need for protocols.

Since the main body of the thesis consists largely of one set of remedies to these two problems, and of some investigations that arise from the new formal machinery that we use, the old CCS proof constitutes an integral and very substantial part of the work reported. There are also parts of it that have not been recast in the new notation; these could be the starting point for further work. Substantially better proofs in CCS are not very likely, but cannot be ruled out. The old proof should not be regarded as entirely superseded even for the recast parts, for while the newer formulations and proofs represent a marked improvement in clarity, the old proof is the best that we could do in CCS, and therefore is still of some interest in its own right. More importantly, it might suggest new formulations different from those we work out in this thesis.

Despite all this, we have relegated the CCS specification and proof to the Appendix. The main reason is that it is a self contained piece of work. The reader is encouraged to look at the first few pages, and the last few, but the rest of the thesis can be read independently of it. We have merely made the old proof available to those readers who wish to draw their own conclusions from it, and to those who wish to follow more closely our comparisons with the new one.

Minor notational differences are another reason for separating the old proof from the new. For example, the Appendix uses traditional CCS notation for actions and values, which is not suitable for the new calculus we develop. The CCS proof has to deal with large numbers of intermediate states for the agents. The new proof does not, and can therefore use more concise names. While the

two notations suit the respective proofs well enough, the differences could be distracting if the Appendix were to become Chapters 2 and 3. The alternative of adopting a common notation would cramp both proofs, forcing each to abide by conventions necessary for the other. Indeed, our experience has convinced us that we should be prepared to use different formal systems, and therefore different notations, for different problems. [LM86] report a similar experience with another problem solved in a CCS setting. There was thus little motivation to go through a long and difficult typesetting exercise to move the old proof into the main body of the thesis.

The ideal solution would have been to abridge the old proof. That we have not been able to do this in CCS for even subproblems of any significance, and that we have found concise proofs in the new calculus, are among the major facts to emerge from this thesis.

1.3 Overview of the thesis

Even the FTS in the Appendix is extremely simple, and our constraints are very strong. Nonetheless, the motivation to formulate it actually arose from practice, as we saw; a few more details can be found in the Section 2.1.1 of the Appendix. We shall not concern ourselves further with such issues; "practice" in this thesis refers to using formal systems to draw conclusions about examples expressed in them. Nor shall we be concerned very much with trying to generalise to other kinds of fault tolerant systems. Instead, we shall focus on very simple-minded models of failure and recovery, and carry out formal proofs about them.

We motivated MCCS and the checkpoint operators as if they were independent, and indeed MCCS does not depend in any way on these new operators. But our major use for the operators is via the Synchronised Displacement theorem, which as its name suggests, is crucially dependent on synchronous actions. Also, the particulate nature of the (freely generated) action set we use is fundamental to the definition of the checkpoint operator.

We have already emphasised that these operators are only one interesting formulation of the problems at hand. Now we should go further and say that while we do not see any obvious alternatives to the SDT as the major mode of applying these operators to FTS-like systems, that theorem too is only one formulation. Thus one might almost say that the formal structures set up in this thesis are really the result of a study to find sufficient conditions for some form of SDT to hold.

Chronologically, the FTS example in the Appendix came first, then we dis-

covered checkpoint operators in an attempt to improve the example, then we discovered we needed synchronous actions, then we settled for MCCS after many unsuccessful attempts at formulating a mixed calculus that dealt with both synchronous and asynchronous actions and thus had a tamer expansion theorem, and finally we discovered that it was non-trivial to set up MCCS formally even though it is so closely related to ASCCS and the particulate calculus. Thus many versions of the FTS example and the Synchronised Displacement Theorem were in existence, some in very improbable notation, before we formulated MCCS.

It is fortunate and pleasing that MCCS turns out to be of value independently of its use for our FTS proofs. In contrast, the design choices for the checkpoint operators and the use of SDT as a tool for our proof are still very closely linked. We hope this brief chronology explains to the reader how so much formal apparatus came about from one large example. For FTS is still the only large example we have.

In the next section of this chapter we give examples of bisimulation proofs. This also serves to establish the new layout and notation developed in [Pra84], and to demonstrate some of their advantages. The last section of this chapter briefly refers to a Prolog system that checks bisimulation proofs.

Chapter 2 defines MCCS formally and establishes some of its main properties. It also examines the connections between it and the calculi in [Mil83], clarifying in the process the connection between ASCCS and CCS, as well as contributing to the study of ASCCS.

Chapter 3 introduces the Displace and Audit operators, establishes their basic properties, and solves simple examples with their help.

Chapter 4 introduces the Checkpoint operator, a generalisation of the Audit operator. It also uses this to carry out the proof of our major example, the FTS. Because of all the new formalism we use, this is in some ways not the "same" example as in the Appendix: our new version banishes protocol problems by fiat as we saw. As a consolation, we discuss checkpointing, a feature that the old FTS did not have, and which seemed then intractable.

A question that arises naturally out of the previous work is whether we can define these new operators in terms of existing operators in our various calculi. This is the topic of Chapter 5, which presents a new technique to prove the *non*-derivability of operators. We manage to answer several questions, but not all, about the derivability or otherwise not only of our new operators, but also of many older operators in CCS. Chapters 2 and 5 are both thus independent of any considerations about FTS and our new operators, and could be of interest

to a wider CCS audience.

Chapter 6 presents conclusions and directions for future work.

1.4 Background

The only real prerequisite is familiarity with CCS, [Mil83] or [Mil80]. We have summarised the necessary definitions and theorems from [Mil83], here and in Chapter 2. Thus the thesis is formally self-contained.

1.4.1 Bisimulation

Consider a set of agents $\mathbf{P}$ and a family of action relations $\xrightarrow{a}$ defined over them. We shall always take the actions a to be elements of an abelian group, *Act*. This is usually not needed for most theorems, but turns out to be very convenient in setting up the calculi; in any case we have no use for a more general action set.

DEFINITION 1.4.1. *A binary relation $\mathcal{R} \subseteq \mathbf{P} \times \mathbf{P}$ is a* strong bisimulation *if, whenever $P\mathcal{R}Q$ and $a \in Act$,*
(i) if $P \xrightarrow{a} P'$ then, for some Q', $Q \xrightarrow{a} Q'$ and $P'\mathcal{R}Q'$,
(ii) if $Q \xrightarrow{a} Q'$ then, for some P', $P \xrightarrow{a} P'$ and $P'\mathcal{R}Q'$ □

For any $\mathcal{R} \subseteq \mathbf{P} \times \mathbf{P}$ we may define $\mathcal{F}(\mathcal{R})$ to be the set of pairs $\langle P, Q\rangle$ satisfying clauses (i) and (ii). Clearly, $\mathcal{R}$ is a strong bisimulation if $\mathcal{R} \subseteq \mathcal{F}(\mathcal{R})$.

PROPOSITION 1.4.2. *$\mathcal{F}$ is monotonic over the lattice of binary relations under inclusion.* □

PROPOSITION 1.4.3. *There exists a maximum bisimulation given by $\sim = \cup\{\mathcal{R} \mid \mathcal{R} \subseteq \mathcal{F}(\mathcal{R})\}$. $\sim$ is the maximum fixed point of $\mathcal{F}$.* □

PROPOSITION 1.4.4. *$\sim$ is an equivalence relation.* □

With this equivalence goes an elegant proof technique; to show $P \sim Q$, it is necessary and sufficient to find a bisimulation containing the pair $\langle P, Q\rangle$. Examples follow later in this chapter.

The action 1 is called the "silent" or "invisible" action. Observational equivalence between agents is concerned only with their visible behaviour.

DEFINITION 1.4.5. *Let $u = \langle a_1, a_2, \ldots, a_n\rangle \in Act^*$. Then $P \stackrel{u}{\Longrightarrow} P'$ iff*
$P\left(\xrightarrow{1}\right)^* \xrightarrow{a_1} \left(\xrightarrow{1}\right)^* \cdots \xrightarrow{a_n} \left(\xrightarrow{1}\right)^* P'$. □

Note the special case $u = \varepsilon$, the empty sequence. We shall often write $P \Rightarrow P'$ for $P \stackrel{\varepsilon}{\Longrightarrow} P'$, i.e. for $P \left(\stackrel{1}{\longrightarrow}\right)^* P'$; in particular we have $P \Rightarrow P$. Note that $P \stackrel{1}{\Longrightarrow} P'$ stands for $P \left(\stackrel{1}{\longrightarrow}\right)^* \stackrel{1}{\longrightarrow} P'$. We usually have $u \in (Act - \{1\})^*$.

DEFINITION 1.4.6. *A binary relation $\mathcal{R} \subseteq \mathbf{P} \times \mathbf{P}$ is a weak (or observational) bisimulation if, whenever $P\mathcal{R}Q$ and $u \in (Act - \{1\})^*$,*
(i) if $P \stackrel{u}{\Longrightarrow} P'$ then, for some Q', $Q \stackrel{u}{\Longrightarrow} Q'$ and $P'\mathcal{R}Q'$,
(ii) if $Q \stackrel{u}{\Longrightarrow} Q'$ then, for some P', $P \stackrel{u}{\Longrightarrow} P'$ and $P'\mathcal{R}Q'$ □

Propositions analogous to those for strong bisimulation follow, and we have

PROPOSITION 1.4.7. *$\approx$ is an equivalence relation.* □

The following proposition allows us to work conveniently.

PROPOSITION 1.4.8. *A binary relation $\mathcal{R} \subseteq \mathbf{P} \times \mathbf{P}$ is a weak bisimulation iff, whenever $P\mathcal{R}Q$ and $a \in Act$,*
(i) if $P \stackrel{a}{\longrightarrow} P'$ then, either $a = 1$ and $P'\mathcal{R}Q$ or for some Q', $Q \stackrel{a}{\longrightarrow} Q'$ and $P'\mathcal{R}Q'$,
(ii) if $Q \stackrel{a}{\longrightarrow} Q'$ then, either $a = 1$ and $P'\mathcal{R}Q$ or for some P', $P \stackrel{a}{\longrightarrow} P'$ and $P'\mathcal{R}Q'$ □

PROPOSITION 1.4.9. *$P \sim Q$ implies $P \approx Q$.* □

1.4.2 The action set

The action set *Act* is an abelian group, as for SCCS and ASCCS. As for the particulate calculus in [Mil83], we shall further assume that *Act* is freely generated by a set Σ of *names*. Let us define, for any $A \subseteq Act$, $\overline{A} = \{a^{-1} \mid a \in A\}$; then $\overline{\Sigma} = \{\sigma^{-1} \mid \sigma \in \Sigma\}$ are the *conames*, and we call $\Lambda = \Sigma \cup \overline{\Sigma}$ the *particles*, or *particulate actions*. In CCS, Λ are called the *labels*, and the only actions are $\Lambda \cup \{1\}$. Such actions are called *simple*. We shall use roman letters $a, b, c, \ldots$ to represent actions in general, λ to represent a particle or particulate action, and μ to represent a simple action.

DEFINITION 1.4.10. *Particles λ, λ' are independent if $\lambda \neq \lambda' \neq \lambda^{-1}$. Then every action a is uniquely expressible (up to order of factors) as a product $\lambda_1^{n_1} \ldots \lambda_k^{n_k}$ $(n_i \geq 0)$ of powers of independent particles. We denote $\{\lambda_1, \ldots, \lambda_k\}$ by Part(a). Part(1) is defined to be $\emptyset$. Particles λ, λ' are non-opposing if $\lambda' \neq \lambda^{-1}$. Actions a, a' are non-opposing if Part(a), Part(a') are pairwise non-opposing.* □

Clearly, any action is a unique product (up to order) of non-opposing particles, not necessarily distinct.

LEMMA 1.4.11. *Let $a = a_1 a_2 \ldots a_n$ be a product of non-opposing actions, and let $a = bc$. Then there are products $b = b_1 b_2 \ldots b_n$ and $c = c_1 c_2 \ldots c_n$ of non-opposing actions such that $a_i = b_i c_i$, $1 \leq i \leq n$.* □

1.4.3 Sorts and restriction

We introduce more notation. If we are representing a set of particles by a single symbol, we try to use script letters. Also, whenever possible, we represent the *sort* of an agent P by $\mathcal{P}$. To classify agents according to the actions they may perform, we choose the submonoids generated by subsets of Λ.

DEFINITION 1.4.12. *Any $\mathcal{L} \subseteq \Lambda$ generates a submonoid of Act, denoted $\mathcal{L}^\times$. Let $\mathbf{P}_\mathcal{L}$ be the set of agents P such that for any derivative Q of P, if $Q \xrightarrow{a} Q'$ then $a \in \mathcal{L}^\times$. If $P \in \mathbf{P}_\mathcal{L}$, we say P has sort $\mathcal{L}$, or write $P{:}\,\mathcal{L}$. Thus if $P{:}\,\mathcal{L}$, then Part$(a) \subset \mathcal{L}$ for any action a of any derivative of P.* □

Thus $a \in \mathcal{L}$ makes sense if a is a particle, but not otherwise. But the following abuse of notation leads to no confusion.

DEFINITION 1.4.13. *$a \in \mathcal{L}$ means that all the particles of a are in $\mathcal{L}$. If P is an agent, $a \notin \mathcal{P}$ implies that P has no a action.* □

Note that $1 \in \mathcal{L}$, for any $\mathcal{L}$.

The next definition is much more important. We often need conditions of the form Part$(a) \cap \mathcal{C} \neq \emptyset$ (read "a has particles in $\mathcal{C}$") and Part$(a) \cap \mathcal{C} = \emptyset$ ("a has no particles in $\mathcal{C}$"). We define convenient abbreviations.

DEFINITION 1.4.14. *Let a be an action, and $\mathcal{C}$ a set of particles. Then $a \lessdot \mathcal{C}$ is defined to be* Part$(a) \cap \mathcal{C} \neq \emptyset$, *and $a \not\lessdot \mathcal{C}$ to be* Part$(a) \cap \mathcal{C} = \emptyset$. *We also define $a, b \lessdot \mathcal{C}$ to mean $a \lessdot \mathcal{C}$ and $b \lessdot \mathcal{C}$.* □

Note that $1 \not\lessdot \mathcal{C}$, for any $\mathcal{C}$.

For example, $\dfrac{P \xrightarrow{a} P' \quad a \not\lessdot \mathcal{A}}{P\backslash\mathcal{A} \xrightarrow{a} P'\backslash\mathcal{A}}$ is our definition of the restriction operator. It means exactly the same as the usual definition in [Mil83].

Note that again we are using only the particulate form of restriction rather than the more general one, and that $\backslash\mathcal{N}$ means that only actions in $(Act - \mathcal{N})^\times$ may be performed, where this last is the pair of *monoids* generated by $(Act - \mathcal{N})$, *not* the groups as well. The traditional CCS restriction, which we show below, hides both names and conames. First we need the following definition.

DEFINITION 1.4.15. *If C is a set of particles, we denote by $\overline{C}$ the set of the complements of the particles in it:* $\overline{C} = \{p^{-1} \mid p \in C\}$. □

Note that C and $\overline{C}$ are not necessarily disjoint. We shall not need the complements of sets of particles very often because of the following:

PROPOSITION 1.4.16. $a \lessdot C \iff a^{-1} \lessdot \overline{C}$ *and* $a \not\lessdot C \iff a^{-1} \not\lessdot \overline{C}$

PROOF. $\Longrightarrow$ For the first, let $p \in \mathrm{Part}(a)$ such that $p \in C$. Then $p^{-1} \in \mathrm{Part}(a^{-1})$ and $p^{-1} \in \overline{C}$. For the second, if C is empty, so is $\overline{C}$. If not, take any $p^{-1} \in \overline{C}$. Then $p \in C$, so $p \notin \mathrm{Part}(a)$ and therefore $p^{-1} \notin \mathrm{Part}(a^{-1})$. $\Longleftarrow$ similar. □

Thus the CCS-style restriction, written $\backslash\!\backslash \mathcal{N}$ in MCCS, is defined as follows: $P \backslash\!\backslash \mathcal{N} \equiv P \backslash \mathcal{N} \cup \overline{\mathcal{N}}$. The *derived* rule for $\backslash\!\backslash \mathcal{N}$ then reads

$$\frac{P \xrightarrow{a} P' \quad a, a^{-1} \notin \mathcal{N}}{P \backslash\!\backslash \mathcal{N} \xrightarrow{a} P' \backslash\!\backslash \mathcal{N}}$$

In CCS, the side condition reads "$a \notin \mathcal{N}, \overline{\mathcal{N}}$", but because of the proposition above, our definition is equivalent. One of the ways in which the CCS lineage of our work shows is that we use $\backslash\!\backslash \mathcal{N}$ almost exclusively, $\backslash \mathcal{N}$ almost never.

1.4.4 Morphism and relabelling

Morphisms from one MCCS action set to another are just group morphisms, as in SCCS or ASCCS. In CCS, morphisms preserve simple action; they satisfy the condition $\phi(\Lambda) \subseteq \Lambda \cup \{1\}$. In [Mil80], they are bijections, from one set of names to another, that preserve complements and 1. All group morphisms preserve 1 and inverses, so in our terms, relabelling is a bijective morphism, i.e., an isomorphism. Strong as this constraint of bijectivity is, it is not unrealistic, and the morphisms we use in practice are all relabellings.

1.5 New layout and notation for bisimulation proofs

We shall often use explicit inference trees inside bisimulations. This is useful if we are demonstrating a bisimulation between *expressions*. The syntax of CCS defines a set of expressions, and those with no free variables are called *agents*.

For example, $P + Q$ cannot be said to have any moves at all except under the premise that P or Q has a move. In such a case, we want to match this move by one on the other side, *made under the same premises*. The inference trees make it easy to check that we have not inadvertently made additional assumptions, a definite danger once we get to new and complicated operators

with side conditions. Inference trees also help to check that we have exhausted all the possibilities for an agent. An example follows.

PROPOSITION 1.5.1. *If* $\mathcal{P} \cap \overline{\mathcal{Q}} \cap \mathcal{N} = \overline{\mathcal{P}} \cap \mathcal{Q} \cap \mathcal{N} = \emptyset$ *then* $(P \mid Q)\backslash\mathcal{N} \sim (P\backslash\mathcal{N}) \mid (Q\backslash\mathcal{N})$

PROOF. By the following bisimulation

$$\mathcal{R}(P,Q) \qquad \langle\, (P \mid Q)\backslash\mathcal{N},\quad (P\backslash\mathcal{N}) \mid (Q\backslash\mathcal{N}) \,\rangle$$

Case $P \leftrightarrow$*:*

$$\frac{P \xrightarrow{a} P'}{\dfrac{P \mid Q \xrightarrow{a} P' \mid Q \quad a \notin \mathcal{N}}{(P \mid Q)\backslash\mathcal{N} \xrightarrow{a} (P' \mid Q)\backslash\mathcal{N}}} \qquad \frac{\dfrac{P \xrightarrow{a} P' \quad a \notin \mathcal{N}}{P\backslash\mathcal{N} \xrightarrow{a} P'\backslash\mathcal{N}}}{(P\backslash\mathcal{N}) \mid (Q\backslash\mathcal{N}) \xrightarrow{a} (P'\backslash\mathcal{N}) \mid (Q\backslash\mathcal{N})}$$

leading to $\mathcal{R}(P',Q)$

Case $Q \leftrightarrow$*:*

$$\frac{Q \xrightarrow{a} Q'}{\dfrac{P \mid Q \xrightarrow{a} P \mid Q' \quad a \notin \mathcal{N}}{(P \mid Q)\backslash\mathcal{N} \xrightarrow{a} (P \mid Q')\backslash\mathcal{N}}} \qquad \frac{\dfrac{Q \xrightarrow{a} Q' \quad a \notin \mathcal{N}}{Q\backslash\mathcal{N} \xrightarrow{a} Q'\backslash\mathcal{N}}}{(P\backslash\mathcal{N}) \mid (Q\backslash\mathcal{N}) \xrightarrow{a} (P\backslash\mathcal{N}) \mid (Q'\backslash\mathcal{N})}$$

leading to $\mathcal{R}(P,Q')$

Case $PQ \leftrightarrow a \notin \mathcal{N} \wedge b \notin \mathcal{N} \iff ab \notin \mathcal{N}$ *follows from the constraint on the sorts of* P *and* Q *:*

$$\frac{P \xrightarrow{a} P' \quad Q \xrightarrow{b} Q'}{\dfrac{P \mid Q \xrightarrow{ab} P' \mid Q' \quad ab \notin \mathcal{N}}{(P \mid Q)\backslash\mathcal{N} \xrightarrow{ab} (P' \mid Q')\backslash\mathcal{N}}} \qquad \frac{\dfrac{P \xrightarrow{a} P' \quad a \notin \mathcal{N}}{P\backslash\mathcal{N} \xrightarrow{a} P'\backslash\mathcal{N}} \quad \dfrac{Q \xrightarrow{b} Q' \quad b \notin \mathcal{N}}{Q\backslash\mathcal{N} \xrightarrow{b} Q'\backslash\mathcal{N}}}{(P\backslash\mathcal{N}) \mid (Q\backslash\mathcal{N}) \xrightarrow{ab} (P'\backslash\mathcal{N}) \mid (Q'\backslash\mathcal{N})}$$

leading to $\mathcal{R}(P',Q')$

□

Each class of pairs in the bisimulation is shown as a parameterised form such as $\mathcal{R}(\mathcal{P},\mathcal{Q})$ above. The bisimulation here consists of pairs all of which have the same form. There could be more forms, in general. A class of pairs that occurs quite often is $\mathcal{I}$, which is the class of all pairs $\langle P,P\rangle$. These pairs are not explicitly listed in the bisimulation.

The proof clearly shows up the fact the side conditions on the two sides are not the same for the last case, and that for the proof to be valid, we need to check that the two imply each other, which they do in this case under

the assumptions of the proposition. We can also check, by working from the outermost operator inwards, that all the derivations on both sides have been accounted for.

For strong bisimulations, each agent has to match every action of the other, so each pair of trees is actually two proofs, that the lhs can match the rhs, and vice-versa. With weak bisimulations, we may need to show that $lhs \xrightarrow{a} lhs'$ and that the rhs matches this by $rhs \stackrel{a}{\Longrightarrow} rhs'$. Thus we can have pairs of trees (or just the bottom lines showing the resulting inference) that are unidirectional. Whenever necessary, we indicate this in the heading line preceding the pair, as in the example above. For strong bisimulations, we shall usually not have any arrows in the heading. The heading line also serves to label the derivations being considered, and sometimes to indicate briefly why the side-conditions imply each other. Sometimes such proofs are tricky enough to merit a separate table to themselves, which is to be read in conjunction with the bisimulation. In such cases, the side-condition table will usually also deal with inferences that are ruled out by the conditions of the proposition.

One of the interesting effects of the work in this thesis is that the general theorems we have are nearly all *strong* bisimulations, as are most of the equivalences we prove in the FTS example! The proof in the Appendix also uses strong bisimulations for all the data transformations, but does not manage to reduce the weak bisimulations down to essentially one.

1.6 A Prolog system for checking bisimulation proofs

As part of early efforts to carry out the FTS example, the author developed a Prolog system to check bisimulations. [Lar86] describes with characteristic thoroughness a different system that carries out bisimulation proofs. Since many of the ideas are the same, we will only make general remarks here based on experience with the proof checker. Mechanical aids can easily be more detrimental than useful to work such as that reported in this thesis. The reason is that the development here depended crucially on ability to rapidly change notation, develop temporary notation to prove lemmas, and so on. At a different level, success even depends on willingness to change formalism and calculus! Few mechanical systems are flexible enough to permit all this, and could well tie the user down to one particular formalism and notation, and some are clumsy and restrictive even within that.

A further danger is that one might start regarding a problem as "solved" if it can be pushed through the machine. The problem might be solved, if

the system is implemented correctly, but it will almost surely not be really understood. When to rest content with such a state of affairs, and when to search for shorter proofs in a different framework is a judgement that comes only with experience, but it is as well to be aware of this factor.

Future work based on this thesis could include further experimentation with mechanical aids. The notation used in the thesis now seems to have stabilised sufficiently.

2

A New Link in a Chain of Calculi

In Chapter 1, we looked informally at the old calculi CCS, SCCS, and ASCCS, and motivated a new one, MCCS. We set up an example where CCS is unsuitable because it forces us to always specify sequences (interleavings) of actions, and SCCS is unsuitable because we have an essentially asynchronous system with only occasional synchronisation needed. ASCCS is the most interesting of the older calculi, for it allows us to enforce synchronisation between actions as well as to allow arbitrary interleavings as in CCS. But we noted problems with it as well: that delays turn up in subtle ways, that it uses $\times$ rather than $|$, and that its weak congruence is stronger than that of CCS. We concluded that the best calculus for our purposes would be one that looked exactly like CCS except that it allowed all actions rather than just particulate ones, and had a $|$ operator that allowed synchronisation rather than just communication. (In the synchronisation rule $\dfrac{P \xrightarrow{a} P' \quad Q \xrightarrow{b} Q'}{P \mid Q \xrightarrow{ab} P' \mid Q'}$, communication is the special case $a = b^{-1}$). This informal specification for a calculus was named MCCS.

From the users point of view, it remains only to ensure that MCCS runs into no formal troubles, establish equational properties for both strong and weak bisimulation, and characterise the weak congruence; we hope this turns out to look like that for CCS.

But it would be a pity to treat MCCS entirely independently when it can

be seen as a minor variant of either ASCCS or CCS. [Mil83] creates a chain of calculi: ASCCS is a subcalculus of SCCS, and CCS is a subcalculus of ASCCS. If we note that δ is a derived operator in all four calculi, and think of it for the moment as being included in the syntax of all four, then ASCCS is one step away from SCCS (replace a: by a.), and the particulate calculus, which turns out to be isomorphic to CCS, is two steps away from ASCCS (replace $\times$ by $|$, and restrict observers to seeing only single particles instead of all actions). Surely if we take only the first of these steps from ASCCS we get something isomorphic to MCCS? (Actually $|$ is just a derived operator in ASCCS, so the real change is to make a. a primitive operator rather than define it in terms of a:). And the second of these steps from MCCS should give us something isomorphic to CCS?

In this chapter we carry out the programme in the previous two paragraphs. We formalise MCCS; in the sequel, "MCCS" refers to the calculus directly defined, to distinguish it from "derived MCCS", derived from ASCCS. We work out the properties of derived MCCS, the relation between it and MCCS, the relation between particulate MCCS (i.e. the particulate calculus derived from MCCS) and CCS, and the characterisation theorem for weak congruence in MCCS, all using the methods of [Mil83]. Not surprisingly, derived MCCS and MCCS turn out to be very closely related, as are MCCS and CCS. But all is not routine and boredom.

The first interesting fact is that weak congruence for MCCS turns out to look exactly like that for CCS. Whether this should have been entirely expected is not clear: Milner comments that the observation congruence depends on both the operators admitted and what can be observed. Moving directly from ASCCS to CCS changes both factors at once. By separating the factors out we hope to shed some light on what effect each has independently.

MCCS also turns out to be unexpectedly interesting for the light it casts on the relationship between ASCCS and CCS, more specifically, on the translation between the particulate calculus and CCS. [Mil83] shows that $\approx$ for CCS is the restriction of $\approx$ for ASCCS. But this is somewhat disappointing, because it mixes up delay discrepancies in the translation and 1 actions within the individual calculi. For example, why can't we carry over even simple laws like the associativity of $+$, when $+$ behaves exactly alike in both calculi?

If we look more closely at why the translation does not preserve $\sim$, we see that it introduces delay discrepancies for two distinct reasons. We will establish closer relations between the calculi which keep the two kinds apart

as the delay discrepancies between MCCS and derived MCCS, and between particulate MCCS and CCS respectively. We also keep out the 1's in the individual calculi to a large extent; thus isolating the sources of discrepancy. It is a conjecture that there exists a useful subclass of expressions in (A)SCCS for which we can carry several congruence laws from SCCS all the way through to CCS via MCCS.

A final bonus is that we get a couple of new non-derivability results for operators. This work is carried out in Chapter 5.

Readers who are not interested in the details of our new chain of calculi can glance through the statements of the theorems in the next few sections and skip directly to Section 2.4 for a summary of facts about MCCS for later use.

We begin by summarising the necessary facts about SCCS and ASCCS. This allows us not only to produce a self-contained description of MCCS, while building up the formal apparatus in its traditional setting, but also to gather together facts for immediate reference and for comparison with the new calculus. The material in the next two sections is quoted directly and without proof from [Mil83], with a little simplification to suit our needs. Those familiar with that paper should therefore skip these sections.

2.1 SCCS

The syntax of expressions in our restricted version of this calculus is as follows.

$$E ::= X \mid \mathbf{0} \mid a{:}E \mid E+E \mid E \times E \mid E\backslash\mathcal{N} \mid \mathrm{fix}\, XE$$

Here E is an expression, X is a variable. $\mathcal{N}$ is a set of particles. Let $\mathbf{E}^S$ and $\mathbf{P}^S$ be respectively the expressions and agents (closed expressions) of SCCS.

"fix" is actually a family of constructors, one for each variable. If t is a term, fixXt is also a term. Readers who prefer recursive definitions of agents as in [Mil80] can think of the term fix$X\,(a{:}X + b{:}X)$ as the agent defined by the recursive definition $X \Leftarrow a{:}X + b{:}X$.

Free and bound variables arise in the usual way from the recursive terms.

DEFINITION 2.1.1. *Let Free(E) denote the set of variables free in E. Let* $E[\tilde{G}/\tilde{X}]$ *denote the result of simultaneously replacing free occurrences of* X_i *in E by* G_i*, for each i, with changes of bound variables as necessary to avoid clashes.* □

The reader should note that we have made several trivial simplifications from the definition given in [Mil83]. There, sums of the form $\Sigma_{i\in I} E_i$ are allowed, where $\langle E_i \mid i \in I\rangle$ is some indexed family of expressions (possibly infinite).

Then **0** and $+$ are just the special cases $I = \emptyset$ and $I = \{0,1\}$. Since these are all we need for finitely expressed agents, we have chosen to stay with a syntax reminiscent of CCS. The set of actions, *Act*, is taken in SCCS to be any abelian group. Here we assume that it is freely generated by a set of names, as described in Chapter 1, so that our restriction operator is a special case of that used in SCCS.

DEFINITION 2.1.2. *The operational semantics of SCCS.*

Action	$a{:}E \xrightarrow{a} E$	
Sum	$\dfrac{E \xrightarrow{a} E'}{E+F \xrightarrow{a} E'}$	$\dfrac{E \xrightarrow{a} E'}{F+E \xrightarrow{a} E'}$
Product	$\dfrac{E \xrightarrow{a} E' \quad F \xrightarrow{b} F'}{E \times F \xrightarrow{ab} E' \times F'}$	
Restriction	$\dfrac{E \xrightarrow{a} E' \quad a \notin \mathcal{A}}{E\backslash\mathcal{A} \xrightarrow{a} E'\backslash\mathcal{A}}$	
Morphism	$\dfrac{E \xrightarrow{a} E'}{E[\phi] \xrightarrow{\phi(a)} E'[\phi]}$	
Recursion	$\dfrac{E[\text{fix } XE/X] \xrightarrow{a} E'}{\text{fix } XE \xrightarrow{a} E'}$	
Delay	$\delta E \xrightarrow{1} \delta E$	$\dfrac{E \xrightarrow{a} E'}{\delta E \xrightarrow{a} E'}$

□

For more general definitions of sum, restriction and recursion, see [Mil83].

We included in the semantics two derived operators, morphism and delay, which were not in the syntax at the start. Morphism, which corresponds to the relabelling operator of CCS, is written postfix. In the term $P[\phi]$, ϕ is just a group morphism from one set of actions to another. For a proof of derivability of morphism, see the last section of [Mil83]. Delay, δP, can be easily shown to be strongly congruent to fix $X\,(1{:}X + P)$. We shall use both operators freely from now on.

Strong bisimulation yields the most important facts about SCCS. First, let us extend the relation to expressions.

DEFINITION 2.1.3. *Let $\tilde{X}$ include the free variables of E and F. Then $E \sim F$ if $E[\tilde{P}/\tilde{X}] \sim F[\tilde{P}/\tilde{X}]$ for all agents $\tilde{P}$.* □

PROPOSITION 2.1.4. *$\sim$ is a congruence, that is, $E \sim F$ implies $a{:}E \sim a{:}F$, $E+G \sim F+G$, $E \times G \sim F \times G$, $E\backslash\mathcal{N} \sim F\backslash\mathcal{N}$, and* fix $XE \sim$ fix XF. □

Because of this extremely important fact, we shall often refer to $\sim$ as *strong congruence.*

We state below a few equational properties so that readers can compare $\times$ and the action prefix $a{:}$ with the $|$ and $a.$ operators we shall be using in MCCS. The δ operator will not be used in MCCS, but is important for proofs later in this chapter. Summation, restriction and morphism are all used in MCCS; we shall study them later.

We define $\mathbf{1} \equiv \text{fix}\, X\,(1{:}X)$.

PROPOSITION 2.1.5.

1. $(\mathbf{P}^S/\sim, \times, \mathbf{1})$ *is an abelian monoid.*
2. $P \times \mathbf{0} \sim \mathbf{0}$
3. $P \times (Q+R) \sim (P \times Q) + (P \times R)$
4. $a{:}P \times b{:}Q \sim ab{:}(P \times Q)$
5. $\delta P \times \delta Q \sim \delta\,(P \times \delta Q + \delta P \times Q)$
6. $\delta\delta P \sim \delta P$
7. $\delta P \sim P + 1{:}\delta P \sim P + \delta P$

□

2.2 ASCCS

We can think of the agents of SCCS as synchronised with a universal clock. $a{:}b{:}\mathbf{0}$ must perform b on the next clock tick after a, and $a{:}1{:}b{:}\mathbf{0}$ must perform b with a delay of exactly one tick after a. By contrast, $a{:}\delta\,(b{:}\mathbf{0})$ can *idle* indefinitely between the two actions. The following definition ensures that idleness is preserved by strong congruence.

DEFINITION 2.2.1. *P is idle if $P \sim \delta P$.* □

PROPOSITION 2.2.2. *(1) δP is idle, for any P.*

(2) If $P \xrightarrow{1} P$ then P is idle.

(3) If $P \sim Q$ and P is idle, then Q is idle □

A natural way to build an asynchronous calculus might appear to be to restrict ourselves to agents whose derivatives are all idle. But this leads to the following problem. $a{:}P$ and $b{:}Q$ are not idle, and their sum $a{:}P + b{:}Q$ is very different from the sum $\delta\,(a{:}P) + \delta\,(b{:}Q)$ since the latter can discard one of its summands autonomously, whereas the former is controlled externally. We wish to preserve this external control, and so we use the following

DEFINITION 2.2.3. *P is asynchronous if all its proper derivatives are idle.* □

This property is preserved by strong congruence. But it is not preserved by the action operator of SCCS; $b{:}\,\delta\,(\mathbf{0})$ is asynchronous but $a{:}\,b{:}\,\delta\,(\mathbf{0})$ is not. To get an asynchronous calculus we replace this action operator by the derived operator "$a.$" defined by

$$a.P \equiv a{:}\,\delta P$$

We now have ASCCS. Morphism and delay are still derived operators in this calculus; it is easily shown that $\delta P \sim \mathrm{fix}X\,(1.X + P)$. It also turns out that $\delta P \sim 1.P + P$, but it is better not to take this simpler fact as the definition of δ. Our definition of δ keeps the same form in all the calculi, and the extra fact, which turns out to be bothersome, is a property only of ASCCS. We need δ so frequently that we include it in the syntax of the calculus.

$$E ::= X \mid \mathbf{0} \mid a.E \mid \delta E \mid E + E \mid E \times E \mid E\backslash\mathcal{N} \mid \mathrm{fix}\,XE$$

Let $\mathbf{E}^A$ and $\mathbf{P}^A$ be respectively the expressions and agents (closed expressions) of ASCCS.

PROPOSITION 2.2.4. *If $P \in \mathbf{P}^A$ and $P \xrightarrow{a} P'$, then $P' \in \mathbf{P}^A$ and $P' \xrightarrow{1} P'$. Every P in $\mathbf{P}^A$ is asynchronous.* □

Now consider $P \equiv a.b.\mathbf{0}$ and $Q \equiv a.1.b.\mathbf{0}$; both can delay indefinitely between the actions a and b, but P *can* perform b immediately after a whereas Q cannot. P and Q are not strongly congruent, but cannot be distinguished by an observer who cannot measure the length of a sequence of invisible actions. Observational equivalence captures precisely this abstraction. The following proposition relates the action operators and the delay operator, and is in contrast to strong congruence.

PROPOSITION 2.2.5. $P \approx 1{:}\,P \approx 1.P \approx \delta P$ □

Some operators of SCCS preserve observational equivalence:

PROPOSITION 2.2.6. *If $P \approx Q$ then $a{:}\,P \approx a{:}\,Q$, $\delta P \approx \delta Q$ and $P\backslash\mathcal{N} \approx Q\backslash\mathcal{N}$.* □

However, $P \approx Q$ does not imply $P \times R \approx Q \times R$ *even within ASCCS*; consider $P \equiv a.\mathbf{0}$, $Q \equiv 1.a.\mathbf{0}$, $R \equiv b.\mathbf{0}$; $P \times R$ can do ab, $Q \times R$ cannot. The same values of P, Q, R show also that $+$ does not preserve $\approx$. As in [Mil80], we look for a congruence $\approx^c$ that on general grounds is defined by

$$P \approx^c Q \text{ iff } \quad \text{for all contexts } C[\,], C[P] \approx C[Q]$$

A context $C[\,]$ is an expression with a hole, to be filled in by another expression. *The definition of $\approx^c$ depends on which contexts are allowed.* If any

SCCS context is allowed, $\approx^c$ turns out to be just $\sim$. However, if we allow only ASCCS contexts, the congruence $\approx^{cA}$, where the A in the superscript indicates the allowed contexts, turns out to be very close to $\approx$. We first extend the definition of $\approx$ to expressions before defining $\approx^{cA}$.

DEFINITION 2.2.7. *Let $\tilde{X}$ include the free variables of E and F. Then $E \approx F$ if $E[\tilde{P}/\tilde{X}] \approx F[\tilde{P}/\tilde{X}]$ for all agents $\tilde{P}$.* □

DEFINITION 2.2.8. *$E \approx^{cA} F$ iff for all asynchronous (ASCCS) contexts $C[\ ]$, $C[E] \approx C[F]$.* □

The following theorem characterises $\approx^{cA}$.

THEOREM 2.2.9. *The following are equivalent in ASCCS:*

1. $P \approx^{cA} Q$
2. *For all R in $\mathbf{P}^A$, $P \times R \approx Q \times R$*
3. *For all $a \in Act$,*
   (i) if $P \xrightarrow{a} P'$ then, for some Q', $Q \xrightarrow{a} \Rightarrow Q'$ and $P' \approx Q'$,
   (ii) if $Q \xrightarrow{a} Q'$ then, for some P', $P \xrightarrow{a} \Rightarrow P'$ and $P' \approx Q'$.

Moreover $\approx^{cA}$ is a congruence, and, if $\tilde{X}$ contains the free variables of E and F, then $E \approx^{cA} F$ iff, for all agents $\tilde{P}$, $E[\tilde{P}/\tilde{X}] \approx^{cA} F[\tilde{P}/\tilde{X}]$. □

Note that clause 3 imposes a stronger condition than weak bisimulation, and that this extra demand pertains only to the initial actions of P and Q.

2.3 MCCS in relation to ASCCS

We first present derived MCCS, an intermediate subcalculus in the derivation of CCS from ASCCS, and then compare it with MCCS (i.e. the calculus defined directly). This section follows the treatment of the relation between the particulate calculus and CCS in section 9 of [Mil83]. We point out the differences as we go along.

2.3.1 Derived MCCS

To derive MCCS from ASCCS we make only one change. Instead of the product operator $\times$, we use the composition operator, $|$, defined by

$$P \mid Q \equiv P \times \delta Q + \delta P \times Q$$

The following simple properties are used later in proofs.

PROPOSITION 2.3.1. $\delta(P \mid Q) \sim \delta P \mid Q \sim P \mid \delta Q \sim \delta P \times \delta Q$

PROOF. Use Proposition 2.1.5 □

The syntax is

$$E ::= X \mid \mathbf{0} \mid a.E \mid \delta E \mid E + E \mid E \mid E \mid E \backslash \mathcal{N} \mid \text{fix}\, XE$$

Note that δE is still available as a derived operator, $\delta E \sim \text{fix}X\,(1.X + E)$. We retain it in the syntax to facilitate comparison with ASCCS. There will be a profusion of δ's when we study how MCCS is related to derived MCCS, and it helps to know that none of them arose merely because of syntax.

Let $\mathbf{E}^M$ and $\mathbf{P}^M$ be respectively the expressions and agents (closed expressions) of MCCS. For the rest of this section, we shall let P and Q stand for members of $\mathbf{P}^A$ and $\mathbf{P}^M$ respectively. Clearly, $\mathbf{P}^M \subseteq \mathbf{P}^A$, and by Proposition 2.2.4, every P in $\mathbf{P}^M$ is asynchronous, i.e. if $P \xrightarrow{a} P'$ then P' is idle, i.e., $P' \sim \delta P'$.

Derived MCCS is not closed under $\rightarrow$, because $\times$ is used in the definition of $\mid$, and shows up in the derivatives of agents involving this operator; however, it appears only in limited contexts, and the calculus is very nearly closed.

PROPOSITION 2.3.2. *If $Q \in \mathbf{P}^M$ has a proper derivative P', then $P' \sim \delta Q'$ for some $Q' \in \mathbf{P}^M$. That is, $P' \sim Q''$ for some $Q'' \in \mathbf{P}^M$.*

PROOF. First show, by induction on inference, that if $Q \xrightarrow{a} P'$ then $P' \sim \delta Q'$ for some Q' in $\mathbf{P}^M$. Two cases are interesting. If Q is of the form $a.Q'$, where $Q' \in \mathbf{P}^M$, we have $Q \xrightarrow{a} \delta Q'$and $P' \equiv \delta Q'$. If Q is of the form $Q_1 \mid Q_2$, we could have $\dfrac{\dfrac{Q_1 \xrightarrow{a} P_1' \quad \delta Q_2 \xrightarrow{1} \delta Q_2}{Q_1 \times \delta Q_2 \xrightarrow{a} P_1' \times \delta Q_2}}{Q_1 \mid Q_2 \xrightarrow{a} P_1' \times \delta Q_2}$ where by induction hypothesis, $P_1' \sim \delta Q_1'$ and $Q_1' \in \mathbf{P}^M$. The induction step follows using using Proposition 2.3.1; $\delta Q_1' \times \delta Q_2 \sim \delta\,(Q_1' \mid Q_2)$.

The rest is easy by induction on the length of derivation of P'. □

This lemma is just like the one [Mil83] has between the particulate calculus and CCS. But because δ is in MCCS syntax we get the second statement in the proposition as well.

If A, B are sets and $A \subseteq B$, then an equivalence relation over A, restricted to B, is an equivalence relation over B. Thus $\sim$ and $\approx$ restricted to derived MCCS are equivalence relations, but because the calculus is not closed, they

cannot be bisimulations. But this will not delay us, for we do not intend to work with derived MCCS; we only want it to compare with MCCS proper, and thereby relate that calculus to ASCCS.

ASCCS is very different from CCS or SCCS in some ways, and these differences make it perhaps difficult to use. Its most striking feature is that action prefix is a derived operator, $a.P \equiv a{:}\,\delta P$, and therefore $a.P \xrightarrow{a} \delta P$. This rule is unusual; for users unaccustomed to δ's turning up implicitly in derivatives, the situation is initially error prone, particularly because they have to remember to put in explicit δ's to avoid enforced synchronisation on the first actions of non-idle agents. It also has the consequence that even agents without "fix" or explicit δ's can have infinite derivations, and usually do. A related odd feature of ASCCS is that non-idle agents (e.g. $\mathbf{0}$) can be used as building blocks (e.g. $a.\mathbf{0}$), but cannot be proper derivatives of any agent, because all such are idle. Lastly, $1.P + P \sim \delta P$, which means that if you ask for one 1, you might get an infinity of them, and it is hard to tell which ones were explicitly asked for.

We can understand these properties of ASCCS as coming about because it is simulating an asynchronous machine on a synchronous one. Derived MCCS inherits them. Its implementation of MCCS is thus imperfect, and the last property above will shortly get us into a lot of trouble.

2.3.2 Relating derived MCCS to primitive MCCS

We define MCCS directly, treating all of its operators as primitive, by means of a new family of action relations $\xrightarrow{a}\!\!\!\!\!\rightarrow$. The syntax is the same as for the derived version, so we can still use $\mathbf{P}^M$ and $\mathbf{E}^M$ as the set of agents and expressions respectively. The new derivation rules are the same as for $\xrightarrow{a}$ in SCCS except for the MCCS operators $a.$ and $\mid$ (which are derived operators in SCCS); for these we have the following rules:

$$\textit{Action} \qquad a.E \xrightarrow{a}\!\!\!\!\!\rightarrow E$$

$$\textit{Parallel} \qquad \frac{E \xrightarrow{a}\!\!\!\!\!\rightarrow E'}{E \mid F \xrightarrow{a}\!\!\!\!\!\rightarrow E' \mid F} \qquad \frac{E \xrightarrow{a}\!\!\!\!\!\rightarrow E'}{F \mid E \xrightarrow{a}\!\!\!\!\!\rightarrow F \mid E'}$$

$$\textit{Synchronisation} \qquad \frac{E \xrightarrow{a}\!\!\!\!\!\rightarrow E' \quad F \xrightarrow{b}\!\!\!\!\!\rightarrow F'}{E \mid F \xrightarrow{ab}\!\!\!\!\!\rightarrow E' \mid F'}$$

Note that δ can still be produced as a derived operator: $\delta P \equiv \mathrm{fix}X\,(1.X + P)$ as usual, but there are no $a{:}$ and $\times$ from which to derive $a.$ and $\mid$. The synchronisation rule is a generalisation of the communication rule for CCS. We refer to the special case of synchronisation where $a = b^{-1}$ as communication. The

real difference between the $\longrightarrow$ calculus (derived MCCS) and the $\longrightarrow\!\!\!\!\!\rightarrow$ calculus (MCCS) is in the action rule, because the $\longrightarrow\!\!\!\!\!\rightarrow$ rules for $\mid$ look like the $\longrightarrow$ rules for the derived operator.

MCCS thus defined is of course a closed calculus; we can show by induction that if $Q \in \mathbf{P}^M$ and $Q \stackrel{a}{\longrightarrow\!\!\!\!\!\rightarrow} Q'$ then $Q' \in \mathbf{P}^M$.

DEFINITION 2.3.3. *Let $\sim_\Pi$ be the strong bisimulation corresponding to the action relations $\longrightarrow\!\!\!\!\!\rightarrow$.* □

The following two lemmas show that the $\longrightarrow$ calculus differs from the $\longrightarrow\!\!\!\!\!\rightarrow$ calculus by only a δ factor. It is worth noting that if $Q \in \mathbf{P}^M$, its $\longrightarrow$ derivatives are idle, but usually not its $\longrightarrow\!\!\!\!\!\rightarrow$ derivatives.

LEMMA 2.3.4. *If $Q \stackrel{a}{\longrightarrow\!\!\!\!\!\rightarrow} Q'$ then, for some P', $Q \stackrel{a}{\longrightarrow} P' \sim \delta Q'$.*

PROOF. By induction on inference. In the case $Q \equiv Q_1 \mid Q_2$ Proposition 2.3.1 is needed. If $\dfrac{Q_1 \stackrel{a_1}{\longrightarrow\!\!\!\!\!\rightarrow} Q_1' \quad Q_2 \stackrel{a_2}{\longrightarrow\!\!\!\!\!\rightarrow} Q_2'}{Q_1 \mid Q_2 \stackrel{a_1 a_2}{\longrightarrow\!\!\!\!\!\rightarrow} Q_1' \mid Q_2'}$, we have by the induction hypothesis, $Q_i \stackrel{a_i}{\longrightarrow} P_i'$ with $P_i' \sim \delta Q_i'$ for $i = 1, 2$. Then

$Q_1 \mid Q_2 \equiv Q_1 \times \delta Q_2 + Q_2 \times \delta Q_1 \stackrel{a_1 a_2}{\longrightarrow} P_1' \times P_2' \sim \delta Q_1' \times \delta Q_2' \sim \delta\,(Q_1' \mid Q_2')$. □

LEMMA 2.3.5. *If $Q \stackrel{a}{\longrightarrow} P'$, then, for some Q', $Q \stackrel{a}{\longrightarrow\!\!\!\!\!\rightarrow} Q'$ and $P' \sim \delta Q'$.*

PROOF. By induction on inference. Two cases are of interest:

(i) $Q \equiv b.Q_1$. Then $a = b$, $P' \equiv \delta Q_1$ and $Q \stackrel{b}{\longrightarrow\!\!\!\!\!\rightarrow} Q_1$.

(ii) $Q \equiv Q_1 \mid Q_2 \equiv Q_1 \times \delta Q_2 + \delta Q_1 \times Q_2$. Then by shorter inferences we have, without loss of generality, $Q_1 \stackrel{b}{\longrightarrow} P_1'$, $\delta Q_2 \stackrel{c}{\longrightarrow} P_2'$, $P' \equiv P_1' \times P_2'$ and $a = bc$.

There are now two subcases. If $c = 1$ and $P_2' \equiv \delta Q_2$, then $b = a$ and we use the induction hypothesis for Q_1 to obtain $Q_1 \stackrel{b}{\longrightarrow\!\!\!\!\!\rightarrow} Q_1'$ where $P_1' \sim \delta Q_1'$.

Then $\dfrac{Q_1 \stackrel{b}{\longrightarrow\!\!\!\!\!\rightarrow} Q_1'}{Q_1 \mid Q_2 \stackrel{b}{\longrightarrow\!\!\!\!\!\rightarrow} Q_1' \mid Q_2}$ and $P' \equiv P_1' \times \delta Q_2 \sim \delta Q_1' \times \delta Q_2 \sim \delta\,(Q_1' \mid Q_2)$ by Proposition 2.3.1.

In the other subcase, $Q_2 \stackrel{c}{\longrightarrow} P_2'$. By the induction hypothesis, this can be matched by $\dfrac{Q_1 \stackrel{b}{\longrightarrow\!\!\!\!\!\rightarrow} Q_1' \quad Q_2 \stackrel{c}{\longrightarrow\!\!\!\!\!\rightarrow} Q_2'}{Q_1 \mid Q_2 \stackrel{bc}{\longrightarrow\!\!\!\!\!\rightarrow} Q_1' \mid Q_2'}$ where $P_1' \sim \delta Q_1'$ and $P_2' \sim \delta Q_2'$, and so we have $P' \equiv P_1' \times P_2' \sim \delta Q_1' \times \delta Q_2' \sim \delta\,(Q_1' \mid Q_2')$ by Proposition 2.3.1.

□

Compare the above two lemmas with Lemma 9.8 and Lemma 9.9 of [Mil83] between CCS (the $\twoheadrightarrow$ calculus there) and the corresponding subcalculus of ASCCS ($\rightarrow$ there). We are taking smaller steps, and the smaller gap between our two calculi makes the previous lemma stronger than its counterpart (Lemma 9.9), and its proof simpler. CCS can match the actions of its $\rightarrow$ calculus only by arbitrary interleavings of the particles in the original action, because its observers are restricted to seeing only particles. MCCS observers are not thus restricted, so we get a closer link between the $\rightarrow$ and the $\twoheadrightarrow$ calculi.

The stronger lemma above leads to the following result. We can pick up the correspondence with CCS again only when we get to weak bisimulation.

THEOREM 2.3.6. *Let* $Q_1, Q_2 \in \mathbf{P}^M$. *Then* $Q_1 \sim_\Pi Q_2$ *implies* $Q_1 \sim Q_2$.

PROOF. Consider $\mathcal{S} \subseteq \mathbf{P}^A \times \mathbf{P}^A$ defined by

$$\mathcal{S} = \{\langle P_1, P_2\rangle \mid P_1 \sim_\Pi P_2, \text{ or } P_1 \equiv \delta Q_1 \text{ and } P_2 \equiv \delta Q_2 \text{ and } Q_1 \sim_\Pi Q_2\}$$

We have written P_1, P_2 and $\mathbf{P}^A$ to remind the reader that we are dealing with $\rightarrow$, but note that $\mathcal{S}$ is actually a subset of $\mathbf{P}^M \times \mathbf{P}^M$; $\sim_\Pi$ is defined only over $\mathbf{P}^M$, and if $Q \in \mathbf{P}^M$ then $\delta Q \in \mathbf{P}^M$. Thus including δ in MCCS syntax allows us to characterise easily the subset of ASCCS over which the induced $\sim$ is defined; it is just derived MCCS.

It is enough to show that $\mathcal{S}$ is a $\sim$-bisimulation upto $\sim$. That is, we show that if $P_1 \mathcal{S} P_2$ and $P_1 \xrightarrow{a} P_1'$ then for some P_2', $P_2 \xrightarrow{a} P_2'$ and $P_1' \sim \mathcal{S} \sim P_2'$.

If $\langle Q_1, Q_2\rangle \in \mathcal{S}$ and $Q_1 \xrightarrow{a} P_1'$ we have by Lemma 2.3.5 that $Q_1 \xrightarrow{a}\!\!\!\!\twoheadrightarrow Q_1'$ and $P_1' \sim \delta Q_1'$. Now because $Q_1 \sim_\Pi Q_2$, there is a Q_2' such that $Q_2 \xrightarrow{a}\!\!\!\!\twoheadrightarrow Q_2' \sim Q_1'$. Then by Lemma 2.3.4, there is a P_2' such that $Q_2 \xrightarrow{a} P_2' \sim \delta Q_2'$. Thus we have $P_1' \sim \delta Q_1'$, $Q_1' \sim_\Pi Q_2'$ and $\delta Q_2' \sim P_2'$. Then $\langle \delta Q_1', \delta Q_2'\rangle \in \mathcal{S}$ by the definition of $\mathcal{S}$, and so $P_1' \sim \mathcal{S} \sim P_2'$.

Starting from $\langle \delta Q_1, \delta Q_2\rangle \in \mathcal{S}$ leads to no new pairs. □

The converse of the previous theorem does not hold. Consider $Q_1 \equiv a.\delta\,(b.\mathbf{0})$ and $Q_2 \equiv a.b.\mathbf{0}$. Then $Q_1 \xrightarrow{a}\!\!\!\!\twoheadrightarrow \delta\,(b.\mathbf{0})$ and $Q_2 \xrightarrow{a}\!\!\!\!\twoheadrightarrow b.\mathbf{0}$, while $Q_1 \xrightarrow{a} \delta\delta\,(b.\mathbf{0})$ and $Q_2 \xrightarrow{a} \delta\,(b.\mathbf{0})$. Because $\delta\delta P \sim \delta P$, we have $Q_1 \sim Q_2$, but not $Q_1 \sim_\Pi Q_2$. We shall return later to conditions under which the converse does hold.

But first we look at $\approx$ and $\approx^{cA}$. When we have defined the corresponding relations on MCCS, we shall see that the translation preserves these in both directions. Corresponding to the action rules $\twoheadrightarrow$, we define relations $\overset{a}{\Longrightarrow}$.

DEFINITION 2.3.7. *Let* $u = \langle a_1, a_2, \ldots, a_n \rangle \in Act^*$. *Then* $Q \stackrel{u}{\Longrightarrow} Q'$ *iff* $Q \left(\stackrel{1}{\longrightarrow}\right)^* \stackrel{a_1}{\longrightarrow} \left(\stackrel{1}{\longrightarrow}\right)^* \cdots \stackrel{a_n}{\longrightarrow} \left(\stackrel{1}{\longrightarrow}\right)^* Q'$. □

Then, in terms of the corresponding notion of weak bisimulation, which we shall call Π-weak bisimulation, we obtain an equivalence $\approx_\Pi$ over MCCS, the maximum Π-weak bisimulation. We prove that $\approx$, restricted to MCCS, is identical to $\approx_\Pi$. We need a lemma first, which extends the results of the last two lemmas to the relations $\stackrel{a}{\Longrightarrow}$ and $\stackrel{a}{\Longrightarrow}$.

LEMMA 2.3.8. *Let* $Q \in \mathbf{P}^M$ *and* $u \in Act^*$.

(1) If $Q \stackrel{u}{\Longrightarrow} P'$ *then, for some* $Q' \in \mathbf{P}^M$, $Q \stackrel{u}{\Longrightarrow} Q'$ *and* $P' \sim \delta Q'$.

(2) If $Q \stackrel{u}{\Longrightarrow} Q'$ *then, for some* $P' \in \mathbf{P}^A$, $Q \stackrel{u}{\Longrightarrow} P'$ *and* $P' \sim \delta Q'$.

PROOF. (1) From Lemma 2.3.5, it is easy to show by induction on n that if $Q \stackrel{a_1}{\longrightarrow} \cdots \stackrel{a_n}{\longrightarrow} P'$ then, for some Q', $Q \stackrel{a_1}{\longrightarrow} \cdots \stackrel{a_n}{\longrightarrow} Q'$ with $P' \sim \delta Q'$. The proof is now immediate from the definitions of $\stackrel{u}{\Longrightarrow}$ and $\stackrel{u}{\Longrightarrow}$.
(2) Similar. □

Remark We can strengthen the lemma in the case $u = \langle 1 \rangle$. The lemma guarantees the existence of matching actions, but these could be $\stackrel{\varepsilon}{\Longrightarrow}$ and $\stackrel{\varepsilon}{\Longrightarrow}$. In fact there will exist matching actions $\stackrel{1}{\Longrightarrow}$ and $\stackrel{1}{\Longrightarrow}$.

Also, in both cases, if the left hand agent does a non-1 action to start with, there exists a match on the right hand side that also starts with a non-1 action.

THEOREM 2.3.9. *Let* $Q_1, Q_2 \in \mathbf{P}^M$. *Then* $Q_1 \approx Q_2$ *iff* $Q_1 \approx_\Pi Q_2$.

PROOF. $\Longrightarrow$ It is enough to show that $\approx$ restricted to $\mathbf{P}^M$ is a Π-bisimulation. By Lemma 2.3.8, we can easily show that if $Q_1 \approx Q_2$ and $Q_1 \stackrel{u}{\Longrightarrow} Q_1'$ then for some Q_2', $Q_2 \stackrel{u}{\Longrightarrow} Q_2'$, where $\delta Q_1' \sim P_1' \approx P_2' \sim \delta Q_2'$. We can complete the gap here to get $Q_1' \approx Q_2'$ because $P' \sim \delta Q'$ implies $P' \approx Q'$.

$\Longleftarrow$ It is enough to show that $\approx_\Pi$ is a bisimulation upto $\approx$; the argument is similar to that in Theorem 2.3.6, except that we don't have to add pairs $\langle \delta Q_1, \delta Q_2 \rangle$ to $\approx_\Pi$ because $P' \sim \delta Q'$ implies $P' \approx Q'$. □

Now define a relation $\approx_\Pi^\times$ on MCCS as follows.
$P \approx_\Pi^\times Q$ iff $\forall a \in Act$,
(i) if $P \stackrel{a}{\longrightarrow} P'$ then, for some Q', $Q \stackrel{a}{\longrightarrow} \Longrightarrow Q'$ and $P' \approx Q'$,
(ii) if $Q \stackrel{a}{\longrightarrow} Q'$ then, for some P', $P \stackrel{a}{\longrightarrow} \Longrightarrow P'$ and $P' \approx Q'$.

This mimics the definition of weak congruence for ASCCS by means of clause 3 of Theorem 2.2.9. Now that we can move freely back and forth between $\approx$ and $\approx_\Pi$, the following theorem is easy. It shows that the relation $\approx_\Pi^\times$ carries over to weak congruence in ASCCS. It is a stronger relation than the weak congruence we shall define for MCCS shortly.

THEOREM 2.3.10. *Let* $Q_1, Q_2 \in \mathbf{P}^M$. *Then* $Q_1 \approx^{cA} Q_2$ *iff* $Q_1 \approx_\Pi^\times Q_2$.

PROOF. $\Longrightarrow$ If $Q_1 \xrightarrow{a}\!\!\!\!\rightarrow Q_1'$ we have by Lemma 2.3.4 that $Q_1 \xrightarrow{a} P_1'$ and $P_1' \sim \delta Q_1'$, that is, $P_1' \approx Q_1'$. Now because $Q_1 \approx^{cA} Q_2$, there is a P_2' such that $Q_2 \xrightarrow{a} \Rightarrow P_2'$ and $P_1' \approx P_2'$. Then by Lemma 2.3.8 and the following remark, there is a Q_2' such that $Q_2 \xrightarrow{a}\!\!\!\!\rightarrow \Rightarrow Q_2'$ and $P_2' \sim \delta Q_2'$. Thus we have $Q_2' \approx P_2' \approx P_1' \approx Q_1'$, and $Q_1 \approx_\Pi^\times Q_2$.

$\Longleftarrow$ Similar. □

In Theorem 2.4.8, we shall characterise $\approx^{cM}$, the largest congruence contained in $\approx_\Pi$, by the following kind of clause:

if $P \xrightarrow{a}\!\!\!\!\rightarrow P'$ then, for some Q', $Q \overset{a}{\Longrightarrow} Q'$ and $P' \approx Q'$

We see that $\approx_\Pi^\times \subseteq \approx^{cM}$, and that therefore $\approx_\Pi^\times$ is a congruence, and more important, that the previous theorem lets us carry weak congruence results over from ASCCS to MCCS.

Going the other way, define $\approx^{A+}$ on ASCCS by this characterising clause of $\approx^{cM}$. By a very similar proof, we then have

THEOREM 2.3.11. *Let* $Q_1, Q_2 \in \mathbf{P}^M$. *Then* $Q_1 \approx^{cM} Q_2$ *iff* $Q_1 \approx^{A+} Q_2$. □

Now if $Q_1 \approx^{A+} Q_2$, we have $Q_1 \approx^{cM} Q_2$, therefore $Q_1 + R \approx^{cM} Q_2 + R$ for any R in MCCS, and so $Q_1 + R \approx^{A+} Q_2 + R$. Thus, if we keep to MCCS syntax (i.e. do not use $\times$), in ASCCS, we have an equivalence preserved by $+$ that is larger than $\approx^{cA}$. In this case, at any rate, congruence seems to depend only on the operators admitted.

2.3.3 Why the translation does not preserve strong congruence

We have seen that $Q_1 \sim Q_2$ does not imply $Q_1 \sim_\Pi Q_2$. We will now study this a little more. The main goal is to illuminate the relation between the two calculi.

Examples: We have $1.Q + Q \sim \delta Q$, $a.(1.Q + Q) \sim a.Q$ and $a.Q \sim a.\delta Q$ and in none of these cases can we replace $\sim$ by $\sim_\Pi$. We seek a theorem that will characterise either the $\sim$ in these examples, or the agents involved, as *unsafe*; for all *safe* cases, we should have $Q_1 \sim Q_2$ implies $Q_1 \sim_\Pi Q_2$.

We need a lemma first. If $\delta P_1 \sim \delta P_2$, we can of course have $P_1 \sim P_2$, but if not, matters can be subtle:

LEMMA 2.3.12. *If $\delta P_1 \sim \delta P_2$ and $P_1 \not\sim P_2$, then either $P_1 \sim \delta P_1$ or $P_2 \sim \delta P_2$ but not both.*

PROOF. Since $P_1 \not\sim P_2$, either P_1 or P_2 has an action that the other cannot match. Assume this is P_1. Then there is an a such that $P_1 \xrightarrow{a} P_1'$, say, but whenever $P_2 \xrightarrow{a} P_2'$, we have $P_1' \not\sim P_2'$. If $a \neq 1$ then δP_2 cannot match the following derivation: $\dfrac{P_1 \xrightarrow{a} P_1'}{\delta P_1 \xrightarrow{a} P_1'}$, and we would have $\delta P_1 \not\sim \delta P_2$. So $a = 1$. Then δP_1 has the derivation $\dfrac{P_1 \xrightarrow{1} P_1'}{\delta P_1 \xrightarrow{1} P_1'}$. Now δP_2 cannot match this by $\dfrac{P_2 \xrightarrow{1} P_2'}{\delta P_2 \xrightarrow{1} P_2'}$; it can only match it by $\delta P_2 \xrightarrow{1} \delta P_2$. Therefore $P_1' \sim \delta P_2 \sim \delta P_1$. We have shown that there is a P_1' such that $P_1 \xrightarrow{1} P_1'$ and $P_1' \sim \delta P_1$. Now $P_1 \sim \delta P_1$ follows by straightforward bisimulation.

If P_2 were the agent with the extra action, we would get $P_2 \sim \delta P_2$ by a symmetric argument.

If both $P_1 \sim \delta P_1$ and $P_2 \sim \delta P_2$, we have $P_1 \sim P_2$. □

PROPOSITION 2.3.13. *Let $Q_1, Q_2 \in \mathbf{P}^M$, and $Q_1 \sim Q_2$. If no $\twoheadrightarrow$ derivatives of Q_1, Q_2 are idle, then $Q_1 \sim_{\Pi} Q_2$.*

PROOF. We proceed by restricting a subset of $\sim$ to MCCS and showing that is a II-bisimulation. The subset is defined by the condition on the $\twoheadrightarrow$ derivatives. Our condition requires $Q_1' \not\sim \delta Q_1'$, **not** $Q_1' \not\sim_{\Pi} \delta Q_1'$.

If $Q_1 \xrightarrow{a}\!\!\!\!\twoheadrightarrow Q_1'$, we have by Lemma 2.3.4 that $Q_1 \xrightarrow{a} P_1' \sim \delta Q_1'$. Now because $Q_1 \sim Q_2$, there is a P_2' such that $Q_2 \xrightarrow{a} P_2' \sim P_1'$. Then by Lemma 2.3.5, there is a Q_2' such that $Q_2 \xrightarrow{a}\!\!\!\!\twoheadrightarrow Q_2'$ and $P_2' \sim \delta Q_2'$. Thus we have $\delta Q_2' \sim P_2' \sim P_1' \sim \delta Q_1'$. From Proposition 2.3.12, and our condition, we conclude $Q_1' \sim Q_2'$. □

Thus we have a sufficient condition to carry $\sim$ over to $\sim_{\Pi}$, but it is far from satisfactory. Notice the awkwardness of the condition: it refers to derivatives in the $\twoheadrightarrow$ calculus and a property in the $\xrightarrow{}$ calculus. It is also much too strong: it would be enough if at corresponding stages of the bisimulation, the two expressions were either both idle, or neither. Unfortunately, this is not easy to express formally and concisely. Lastly, our condition is on each

expression separately rather than on the pair; thus we are unnecessarily ruling out some pairs.

Our examples show that a syntactic test to check for the condition in Lemma 2.3.13 is hard to find; for example, we cannot try to get the theorem to work for all expressions without "fix"—even though this guarantees finite derivations in the $\twoheadrightarrow$ calculus, $1.P$ is idle in the $\rightarrow$ calculus!

It would be nice to have a syntactic test to characterise a class of *safe expressions*, so that if E and F are safe expressions, and $F_1 \sim F_2$ then $F_1 \sim_\Pi F_2$. We could then take over facts from (A)SCCS without having to establish them directly by Π-bisimulation. The larger the class of safe expressions, the more facts we can borrow. It is just as easy to do bisimulations in MCCS as it is in SCCS or ASCCS, so the aim is not to make proofs easier; it is to avoid unnecessary work in future redoing bisimulations in MCCS if they can be taken over from SCCS; we already know that transfer in the other direction is always possible. The immediate bonus, the set of laws already available for the older calculi, is also a minor motivation.

In our three examples, we want to mark $1.E$ and δE as unsafe; the reader can check that this is all we need to deal with the examples.

Unfortunately, we do not have such a syntactic test at present. We only have a semantic test on agents. Thus we know $Q_1 + Q_2 \sim Q_2 + Q_1$ for all agents Q_1, Q_2, but cannot conclude $Q_1 + Q_2 \sim_\Pi Q_2 + Q_1$ for all agents Q_1, Q_2 in MCCS; this would only hold if $Q_1 + Q_2$ and $Q_2 + Q_1$ were safe agents. But the proof of the commutativity of $+$ in SCCS uses only properties of $+$, and our intuition is that nothing goes wrong in the translation of $+$ from the $\twoheadrightarrow$ calculus to the $\rightarrow$ calculus; therefore we expect the commutative law to hold in MCCS by virtue of the same law in SCCS. More dramatically, we cannot even conclude $1.Q + Q \sim_\Pi 1.Q + Q$; we have to prove that independently!

The author conjectures that it should be possible to extend Proposition 2.3.13 to expressions, defining F to be an idle expression if $F \sim \delta F$. The idea is that if F_1 and F_2 are safe, and $F_1 \sim F_2$, we should be able to substitute even unsafe agents for free variables in these expressions, and still have $F_1 \sim_\Pi F_2$.

Safe expressions, if we can define such a class satisfactorily, would be non-idle, and therefore not proper derivatives in ASCCS. However, their ASCCS derivatives would all be idle, and so would not be safe expressions! Any idleness in the ASCCS implementation of such an expression would thus be purely in the mimicry of asynchrony by a synchronous machine.

From now on, we shall drop the extra arrow heads, and the Π subscripts when writing about MCCS. The notation will be used again, of CCS, when we

compare it with particulate MCCS.

2.4 The facts of MCCS

The syntax is

$$E ::= X \mid \mathbf{0} \mid a.E \mid E+E \mid E \mid E \mid E\backslash\mathcal{N} \mid E[\phi] \mid \text{fix}\, XE$$

Note that we have dropped δ, so that the syntax is exactly that of CCS, except that $a.$ is not limited to $\mu.$, and so morphism is not limited either.

DEFINITION 2.4.1. *The operational semantics of MCCS.*

Action $\quad a.E \xrightarrow{a} E$

Sum
$$\frac{E \xrightarrow{a} E'}{E+F \xrightarrow{a} E'} \qquad \frac{E \xrightarrow{a} E'}{F+E \xrightarrow{a} E'}$$

Parallel
$$\frac{E \xrightarrow{a} E'}{E \mid F \xrightarrow{a} E' \mid F} \qquad \frac{E \xrightarrow{a} E'}{F \mid E \xrightarrow{a} F \mid E'}$$

Synchronisation
$$\frac{E \xrightarrow{a} E' \quad F \xrightarrow{b} F'}{E \mid F \xrightarrow{ab} E' \mid F'}$$

Restriction
$$\frac{E \xrightarrow{a} E' \quad a \notin \mathcal{A}}{E\backslash\mathcal{A} \xrightarrow{a} E'\backslash\mathcal{A}} \qquad \frac{E \xrightarrow{a} E' \quad a, a^{-1} \notin \mathcal{A}}{E\backslash\backslash\mathcal{A} \xrightarrow{a} E'\backslash\backslash\mathcal{A}}$$

Morphism
$$\frac{E \xrightarrow{a} E'}{E[\phi] \xrightarrow{\phi(a)} E'[\phi]}$$

Recursion
$$\frac{E[\text{fix}X.E/X] \xrightarrow{a} E'}{\text{fix}X.E \xrightarrow{a} E'}$$

□

The special case of the synchronisation rule arising if b is a^{-1}, giving $ab = 1$, corresponds to the usual communication rule of CCS. We do not need a second rule to establish that synchronisation is commutative, as we do for sum and parallel, because there are two premises to this rule, and by convention, these can be permuted.

Note that the second rule for restriction above is only a notational convenience. For more general definitions of sum and recursion, see [Mil83]. We shall frequently use the form $[a'/a]$ which stands for the morphism which is the identity on *Act* except that $a \mapsto a'$, (and therefore $a^{-1} \mapsto a'^{-1}$ and so on).

2.4.1 Equational facts

The following are all standard for CCS and can be shown for MCCS also, by bisimulation. Here $\mathbf{P}$ means $\mathbf{P}^M$.

PROPOSITION 2.4.2.

1. $(\mathbf{P}/\sim, +, \mathbf{0})$ *is an abelian monoid.*
2. $P + P \sim P$
3. $(\mathbf{P}/\sim, |, \mathbf{0})$ *is an abelian monoid.*
4. $P\backslash\mathcal{N}\backslash\mathcal{M} \sim P\backslash\mathcal{M}\backslash\mathcal{N}$
5. $a.P\backslash\mathcal{N} \sim a.(P\backslash\mathcal{N})$ *if* $a \notin \mathcal{N}$, $\mathbf{0}$ *otherwise*
6. $(P+Q)\backslash\mathcal{N} \sim P\backslash\mathcal{N} + Q\backslash\mathcal{N}$
7. $P\backslash\mathcal{N} \sim P$ *if* $\mathcal{P} \cap \mathcal{N} = \emptyset$
8. $(P \mid Q)\backslash\mathcal{N} \sim (P\backslash\mathcal{N}) \mid (Q\backslash\mathcal{N})$ *if* $\mathcal{P} \cap \overline{\mathcal{Q}} \cap \mathcal{N} = \overline{\mathcal{P}} \cap \mathcal{Q} \cap \mathcal{N} = \emptyset$
9. $a.P[\phi] \sim \phi(a).P[\phi]$
10. $(P+Q)[\phi] \sim P[\phi] + Q[\phi]$
11. $(P \mid Q)[\phi] \sim P[\phi] \mid Q[\phi]$

□

PROPOSITION 2.4.3.

1. $a.1.P \approx^{cM} a.P$
2. $P + 1.P \approx^{cM} 1.P$
3. $a.(P + 1.Q) + a.Q \approx^{cM} a.(P + 1.Q)$

□

2.4.2 Weak congruence for MCCS

Which operators of MCCS preserve $\approx$? Not $+$, as can be seen by taking $P \equiv a.\mathbf{0}$, $Q \equiv 1.a.\mathbf{0}$ and $R \equiv b.\mathbf{0}$. We then have $P \approx Q$ but $P + R \not\approx Q + R$. But in contrast to ASCCS, all other operators of MCCS preserve $\approx$. Thus MCCS behaves like CCS in this respect.

PROPOSITION 2.4.4. *If* $Q_1 \approx Q_2$ *then* $a.Q_1 \approx a.Q_2$, $Q_1 \mid Q_3 \approx Q_2 \mid Q_3$, $Q_1\backslash\mathcal{N} \approx Q_2\backslash\mathcal{N}$ *and* $Q_1[\phi] \approx Q_2[\phi]$.

PROOF. The proofs are by straightforward bisimulation; we only indicate one step: if $Q_1 \xrightarrow{a} Q_1'$ is matched by $Q_2 \xrightarrow{1^l} Q_2' \xrightarrow{a} Q_2'' \xrightarrow{1^m} Q_2'''$, then

$$\frac{Q_1 \xrightarrow{a} Q_1' \quad Q_3 \xrightarrow{b} Q_3'}{Q_1 \mid Q_3 \xrightarrow{ab} Q_1' \mid Q_3'} \text{ is matched by } \frac{Q_2 \xrightarrow{1^l} Q_2'}{Q_2 \mid Q_3 \xrightarrow{1^l} Q_2' \mid Q_3} \text{ followed by}$$

$$\frac{Q_2' \xrightarrow{a} Q_2'' \quad Q_3 \xrightarrow{b} Q_3'}{Q_2' \mid Q_3 \xrightarrow{ab} Q_2'' \mid Q_3'} \text{ followed by } \frac{Q_2'' \xrightarrow{1^m} Q_2'''}{Q_2'' \mid Q_3' \xrightarrow{1^m} Q_2''' \mid Q_3'}.$$

□

The largest congruence contained in $\approx$, written $\approx^{cM}$, is defined as follows.

DEFINITION 2.4.5. *$Q_1 \approx^{cM} Q_2$ iff for all contexts $\mathcal{C}[\ ]$, $\mathcal{C}[Q_1] \approx \mathcal{C}[Q_2]$* □

Pleasingly, it turns out that just as for CCS, and in contrast to ASCCS, the summation contexts are the only critical ones, resulting in the following characterisation theorem, which looks exactly like that for CCS even though MCCS is a larger calculus. The treatment follows that for ASCCS in section 8 of [Mil83].

We first extend the definition of $\approx$ and $\approx^{cM}$ to expressions.

DEFINITION 2.4.6. *Let $\tilde{X}$ include the free variables of E and F. Then $E \approx F$ if $E[\tilde{P}/\tilde{X}] \approx F[\tilde{P}/\tilde{X}]$ for all agents $\tilde{P}$.* □

DEFINITION 2.4.7. *$E \approx^{cM} F$ iff for all MCCS contexts $\mathcal{C}[\ ]$, $\mathcal{C}[E] \approx \mathcal{C}[F]$.* □

THEOREM 2.4.8. *The following are equivalent in MCCS:*

(1) $P \approx^{cM} Q$

(2) For all R in $\mathbf{P}^M$, $P + R \approx Q + R$

(3) For all $a \in Act$,

(i) if $P \xrightarrow{a} P'$ then, for some Q', $Q \stackrel{a}{\Longrightarrow} Q'$ and $P' \approx Q'$,

(ii) if $Q \xrightarrow{a} Q'$ then, for some P', $P \stackrel{a}{\Longrightarrow} P'$ and $P' \approx Q'$,

Moreover $\approx^{cM}$ is a congruence, and, if $\tilde{X}$ contains the free variables of E and F, then $E \approx^{cM} F$ iff, for all agents $\tilde{P}$, $E[\tilde{P}/\tilde{X}] \approx^{cM} F[\tilde{P}/\tilde{X}]$. □

Note that clause 3 imposes a stronger condition than weak bisimulation, in that 1 actions by the first agent cannot be matched by inaction on the part of the other; all actions, including 1, are treated the same. Comparing with $\approx^{cA}$, (2) there uses $\times$ instead of $+$, and (3) is stronger, insisting that a must be the first action by the matching agent.

Clause (3) gives us the following corollary.

COROLLARY 2.4.9. *$E \sim_{\Pi} F$ implies $E \approx^{cM} F$ implies $E \approx F$.* □

The first main step towards the theorem is to show (2)⇔(3).

LEMMA 2.4.10. *In the statement of Theorem 2.4.8, (2)⇔(3).*

PROOF. $\Longleftarrow$ Assume (3), and let $P + R \xrightarrow{a} P'$. Then this action can come either from P or from R. If it comes from R, we have $Q + R \xrightarrow{a} P'$. If it comes from P, we have $Q \stackrel{a}{\Longrightarrow} Q'$and therefore $Q + R \stackrel{a}{\Longrightarrow} Q'$, where $Q' \approx P'$. Thus (2) holds.

$\Longrightarrow$ Assume (3) is false, so that for example $P \xrightarrow{a} P'$, but whenever $Q \stackrel{a}{\Longrightarrow} Q'$ then $P' \not\approx Q'$. Can (2) be true now? Choose any R that is not observationally equivalent to P. Then $Q + R$ cannot match the actions of $P + R$ that arise from P; this is ruled out by our assumption. Therefore (2) is false whenever (3) is. □

DEFINITION 2.4.11. *$E \approx^+ F$ iff, for all R in $\mathbf{P}^M$, $E + R \approx F + R$.* □

The second main step towards Theorem 2.4.8 is to show that $\approx^+$ is a congruence; it will then follow easily that it is identical with $\approx^{cM}$.

LEMMA 2.4.12. *Let $\tilde{X}$ include the free variables of E and F. Then $E \approx^+ F$ iff $E[\tilde{P}/\tilde{X}] \approx^+ F[\tilde{P}/\tilde{X}]$ for all agents $\tilde{P}$.*

PROOF. Immediate from the definitions. □

LEMMA 2.4.13. *$E \approx^+ F$ implies $E \approx F$.*

PROOF. Take $R \equiv \mathbf{0}$ in the preceding definition. □

For the following lemma, we have written out only the parts that are different from both ASCCS and CCS.

LEMMA 2.4.14. *The relation $\approx^+$ is a congruence in MCCS; that is, $E \approx^+ F$ implies $a.E \approx^+ a.F$, $E + G \approx^+ F + G$, $E \mid G \approx^+ F \mid G$, $E\backslash\mathcal{N} \approx^+ F\backslash\mathcal{N}$, $E[\phi] \approx^+ F[\phi]$ and $\mathrm{fix}XE \approx^+ \mathrm{fix}XF$.*

PROOF. Except for the "fix" combinator, it is enough to do the proof for agents only. In each case, we may work with the characterisation of clause (3) of the theorem, and the details are routine with the help of Lemma 2.4.13 and Proposition 2.4.4.

An example: We show that if $P \approx^+ Q$, then $P \mid R \approx^+ Q \mid R$; we do only the synchronisation case. If $P \xrightarrow{a} P'$ then there is a Q' such that $Q \stackrel{1^l}{\Longrightarrow} Q' \stackrel{a}{\Longrightarrow} Q'' \stackrel{1^m}{\Longrightarrow} Q'''$ and $P' \approx Q'''$, because $P \approx^+ Q$. Then the derivation $\dfrac{P \xrightarrow{a} P' \quad R \xrightarrow{b} R'}{P \mid R \xrightarrow{ab} P' \mid R'}$ is matched by $Q \mid R \xrightarrow{1^l} Q' \mid R \xrightarrow{a} Q'' \mid R' \xrightarrow{1^m} Q''' \mid R'$ where $Q''' \mid R' \approx P' \mid R'$ by Proposition 2.4.4.

To prove that $P \approx^+ Q$ implies $P + R \approx^+ Q + R$ it is even simpler to reason as follows: for every $S \in \mathbf{P}^M$, $(P + R) + S \sim_\Pi P + (R + S) \approx Q + (R + S) \sim_\Pi (Q + R) + S$.

To prove that $E \approx^+ F$ implies $\mathrm{fix}XE \approx^+ \mathrm{fix}XF$, where at most X is free in E or F, consider

$$\mathcal{S} = \{\langle G[\mathrm{fix}XE/X], G[\mathrm{fix}XF/X]\rangle \mid \mathrm{Free}\,(G) \subseteq \{X\}\}$$

We will show that if $G[\mathrm{fix}XE/X] \xrightarrow{a} P'$ then, for some Q' and R', $G[\mathrm{fix}XF/X] \stackrel{a}{\Longrightarrow} R'$ and $P'\mathcal{S}Q' \approx R'$. So $\mathcal{S}$ is a bisimulation upto $\approx$, and taking $G \equiv X$ gives us $\mathrm{fix}XE \approx^+ \mathrm{fix}XF$, from clause (3).

We shall only do one step. Let $G \equiv G_0 \mid G_1$ and we consider only the derivation by the synchronisation rule. Then by shorter inferences, $G_i[\mathrm{fix}XE/X] \xrightarrow{a_i} P_i'$ $(i = 0,1)$, with $a_0 a_1 = a$ and $P \equiv P_0 \mid P_1$. By induction, there are Q_i' and R_i' such that $G_i[\mathrm{fix}XF/X] \stackrel{a_i}{\Longrightarrow} R_i'$ and $P_i'\mathcal{S}Q_i' \approx R_i'$ for $i = 0,1$. Then G has the derivation $G[\mathrm{fix}XF/X] \stackrel{a}{\Longrightarrow} R_0' \mid R_1'$. Take $Q' \equiv Q_0' \mid Q_1'$ and $R' \equiv R_0' \mid R_1'$. Clearly, $P'\mathcal{S}Q'$, and $Q' \approx R'$ follows from Proposition 2.4.4. □

PROPOSITION 2.4.15. *$E \approx^+ F$ iff $E \approx^{cM} F$.*

PROOF. If $E \approx^+ F$, the preceding lemma tells us that for any MCCS context, $C[E] \approx^+ C[F]$. By Lemma 2.4.13, it follows that $E \approx^{cM} F$.

If $E \approx^{cM} F$, then by taking all contexts of the form $[\,] + R$ we have, by definition, $E + R \approx F + R$, and thus $E \approx^+ F$. □

PROOF. **of Theorem 2.4.8.** The preceding proposition shows that (1)⇔(2), and (2)⇔(3) was shown earlier. The preceding proposition and lemma show that $\approx^{cM}$ and $\approx^+$ are the same congruence. □

COROLLARY 2.4.16. *If $E \approx F$ then $a.E \approx^{cM} a.F$.* □

COROLLARY 2.4.17. *If $E \approx^{\times}_{\Pi} F$ then $E \approx^{cM} F$.*

The relation $\approx^{\times}_{\Pi}$ was defined over MCCS by using a clause like that of ASCCS congruence. It is now perhaps better called $\approx^{\times M}$. This last corollary completes the weak congruence link from derived MCCS to MCCS. We have already used the characterisation of $\approx^{cM}$ to go the other way, carrying $\approx^{cM}$ over to ASCCS as $\approx^{A+}$.

2.5 MCCS in relation to CCS

Going from ASCCS to derived MCCS was just a restriction to a nearly closed sub-calculus; going from derived MCCS to MCCS involved just redefining the $a.$ operator as primitive, and resulted in near isomorphism between the two.

Now we will define particulate MCCS as a sub-calculus of MCCS by limiting action prefixes to simple actions $\mu.$, and morphisms to simple morphisms that preserve simple actions. This gives us the syntax of CCS. CCS is still two steps away: we have to redefine $|$ to allow only communication and not synchronisation generally, and we have to restrict observers to particles. We can carry out the second step without reference to CCS, thus getting a new equivalence on particulate MCCS. Armed with this additional equivalence, we study the relation between CCS and particulate MCCS and find that while it is not an isomorphism to within $\sim$, we can get closer than the expected isomorphism to within $\approx$.

Again, we loosely follow section 9 of [Mil83]. Particulate MCCS differs from Milner's particulate calculus basically only in that it has $\mu.$ as a primitive operator, not derived.

2.5.1 Particulate MCCS

The syntax is exactly that of CCS.

$$E ::= X \mid \mathbf{0} \mid \mu.E \mid E+E \mid E|E \mid E\backslash\mathcal{N} \mid E[\phi] \mid \text{fix}\, XE$$

Let $\mathbf{E}^C$ and $\mathbf{P}^C$ be respectively the expressions and agents (closed expressions) of particulate MCCS. For the rest of this section, we shall let P and Q stand for members of $\mathbf{P}^M$ and $\mathbf{P}^C$ respectively.

Particulate MCCS is closed under $\longrightarrow$, because $\mathbf{P}^C$ is just an algebra generated by a subset of the operators that generate $\mathbf{P}^M$. and our rules for MCCS are well behaved—they do not take us from an expression involving one operator to one involving another! This is in effect what a derived operator does, and that is why derived MCCS and Milner's particulate calculus are not closed ($|$ is defined in terms of $\times$).

PROPOSITION 2.5.1. *If $Q \in \mathbf{P}^C$ has a proper derivative P', then $P' \in \mathbf{P}^C$.*

PROOF. First show, by induction on inference, that if $Q \xrightarrow{a} P'$ then $P' \in \mathbf{P}^C$; then do induction on the length of derivation of P'. □

Thus $\sim$ and $\approx$ are strong and weak bisimulations when restricted to particulate MCCS; $\sim$ and $\approx^{cM}$ are congruences.

Particulate MCCS is closed, so we never get a non-simple action prefix; however, we do have non-particulate actions because of the $|$ operator. But like Milner's particulate calculus, particulate MCCS also cannot enforce simultaneity. The following proposition tells us that whenever an agent can do a

non-particulate action, it can also do the particles in any order. The reader might want to review the definitions of non-opposing actions etc. from Chapter 1.

PROPOSITION 2.5.2. *Let* $Q \in \mathbf{P}^C$ *and* $Q \xrightarrow{a} Q'$, *where* $a = \lambda_1 \ldots \lambda_n$ *is a product of non-opposing particles,* $n \geq 1$. *Then* $Q \xrightarrow{\lambda_1} \cdots \xrightarrow{\lambda_n} Q'$.

PROOF. By induction on inference. The only interesting case is when $Q \equiv Q_1 \mid Q_2$, $Q_1 \xrightarrow{b} Q_1'$ and $Q_2 \xrightarrow{c} Q_2'$ with $bc = a$. Now suppose $b = \nu_1 \ldots \nu_k$ and $a = \nu_{k+1} \ldots \nu_m$; by induction we have $Q_1 \xrightarrow{\nu_1} \cdots \xrightarrow{\nu_k} Q_1'$ and $Q_2 \xrightarrow{\nu_{k+1}} \cdots \xrightarrow{\nu_m} Q_2'$. The non-opposing particles from ν_1 to ν_m include those of a; permute the ν's of Q_1 and Q_2 so that the remaining particles are paired off into complements and are first. Then Q has a derivation that begins with an application of the synchronisation rule to cancel out these pairs as well as do λ_1; the rest of the particles are then produced by the parallel rule. □

Now we can define a series of new bisimulations on particulate MCCS by restricting observers to particles, i.e. we systematically change "for all $a \in Act$" in the definitions to "for all $\mu \in \Lambda \cup \{1\}$". Thus we get $\sim_\Lambda$, $\approx_\Lambda$, $\approx_\Lambda^+$ and $\approx_\Lambda^\times$. It is easily seen that $P \sim Q$ implies $P \sim_\Lambda Q$, and so on for the other three.

Now if $P \equiv \lambda_1.\mathbf{0}$, $Q \equiv \lambda_2.\mathbf{0}$, and $R \equiv \lambda_1.\lambda_2.\mathbf{0} + \lambda_2.\lambda_1.\mathbf{0}$ we have $P \mid Q \sim_\Lambda R$ but $P \mid Q \not\sim R$ and $P \mid Q \not\approx R$. Also $P \approx 1.P$ but $P \not\sim_\Lambda 1.P$. Thus $\sim_\Lambda$ and $\approx$ are neither contained in the other, but both are contained in $\approx_\Lambda$.

$\sim_\Lambda$ is not a congruence; consider P, Q and R as above and $S \equiv \lambda_2^{-1}.\mathbf{0}$. Then $P \mid Q \mid S \xrightarrow{\lambda_1} \mathbf{0}$ but we only have $R \mid S \xrightarrow{\lambda_1} 1.\mathbf{0}$. And of course $\approx_\Lambda$ is not a congruence, because $\approx$ is not; the reason is the usual one, $P \approx 1.P$ but $P + Q \not\approx 1.P + Q$. We shall see later that $\approx_\Lambda^+$ is a congruence, and that therefore $\approx_\Lambda^\times$ is one as well, since it is contained in $\approx_\Lambda^+$.

We shall use these $_\Lambda$ relations to establish the connection between CCS and particulate MCCS. The key difference between the two calculi is caught by the following: $\lambda.\mathbf{0} \mid 1.\mathbf{0} \sim 1.\lambda.\mathbf{0} + \lambda.1.\mathbf{0} + \lambda.\mathbf{0}$ and $\lambda.\mathbf{0} \mid 1.\mathbf{0} \sim_\Lambda 1.\lambda.\mathbf{0} + \lambda.1.\mathbf{0} + \lambda.\mathbf{0}$ but we will only have $\lambda.\mathbf{0} \mid 1.\mathbf{0} \sim_\Pi 1.\lambda.\mathbf{0} + \lambda.1.\mathbf{0}$, where $\sim_\Pi$ is the strong congruence over CCS. The difference comes about because $\mid$ in MCCS allows synchronisation but $\mid$ in CCS allows only communication.

2.5.2 Relating particulate MCCS and CCS

We define CCS to have the same syntax as particulate MCCS and the same inference rules, except that instead of the synchronisation rule for $\mid$ we have

the following:

$$\textit{Communication}\quad \frac{E \xrightarrow{\lambda} E' \quad F \xrightarrow{\lambda^{-1}} F'}{E \mid F \xrightarrow{\lambda\lambda^{-1}} E' \mid F'}$$

Note the double arrow heads for the directly defined calculus, and that we say λ instead of μ in the new rule.

The two calculi are related by the following two lemmas. We shall use Q to stand for agents in CCS and P for agents in (full) MCCS.

LEMMA 2.5.3. *If* $Q \xrightarrow{a} Q'$ *then* $Q \xrightarrow{a} Q'$.

PROOF. Trivial. Every $\longrightarrow$ rule has a corresponding $\longrightarrow$ rule. □

LEMMA 2.5.4. *If* $Q \xrightarrow{a} Q'$, *where* $a = \lambda_1 \ldots \lambda_n$ *is a product of non-opposing particles,* $n \geq 0$ *then* $Q \left(\xrightarrow{1}\right)^* \xrightarrow{\lambda_1} \left(\xrightarrow{1}\right)^* \cdots \xrightarrow{\lambda_n} \left(\xrightarrow{1}\right)^* Q'$.

PROOF. By induction on inference. The only interesting case is when $Q \equiv Q_1 | Q_2$, $Q_1 \xrightarrow{b} Q_1'$ and $Q_2 \xrightarrow{c} Q_2'$ with $bc = a$. Compared with Proposition 2.5.2, the difference is that we cannot do the cancelling of opposed particles together with anything else; each such cancellation is a communication and is a separate action. □

COROLLARY 2.5.5. *If* $Q \xrightarrow{\mu} Q'$ *then* $Q \xrightarrow{\mu} \left(\xrightarrow{1}\right)^* Q'$.

PROOF. The particles put out by the shorter inferences before the synchronisation can be permuted so that the lone non-opposed particle, μ, is put out first, and all the cancelling pairs come afterwards. □

This corollary is what we shall mostly use, rather than the lemma preceding it. It looks like the defining clause for $\approx_\Lambda^\times$. The two lemmas here show that particulate MCCS is closer to CCS than Milner's particulate calculus, but the corollary holds for Milner's calculus as well, and so does a corresponding proposition in the opposite direction. Thus the results that follow also hold between CCS and Milner's particulate calculus.

We have seen that $P \sim_\Lambda Q$ and $P \sim_\Pi Q$ do not imply each other. But of course $P \approx Q$ iff $P \approx_\Pi Q$.

DEFINITION 2.5.6. *Let* $u = \langle \lambda_1, \lambda_2, \ldots, \lambda_n \rangle \in \Lambda^*$. *Then* $Q \stackrel{u}{\Longrightarrow} Q'$ *iff* $Q \left(\xrightarrow{1}\right)^* \xrightarrow{\lambda_1} \left(\xrightarrow{1}\right)^* \cdots \xrightarrow{\lambda_n} \left(\xrightarrow{1}\right)^* Q'$.

We shall call the corresponding notion of weak bisimulation Π-weak bisimulation.

LEMMA 2.5.7. *Let $Q \in \mathbf{P}^C$ and $u \in \Lambda^*$.*
(1) If $Q \stackrel{u}{\Longrightarrow} Q'$ then $Q \stackrel{u}{\Longrightarrow} Q'$.
(2) If $Q \stackrel{u}{\Longrightarrow} Q'$ then $Q \stackrel{u}{\Longrightarrow} Q'$. □

THEOREM 2.5.8. *Let $Q_1, Q_2 \in \mathbf{P}^C$. Then $Q_1 \approx Q_2$ iff $Q_1 \approx_\Pi Q_2$.* □

The above lemma and theorem are very similar to the ones in [Mil83] except that they are simpler because particulate MCCS is so close to CCS in form. They are also similar to the ones we saw earlier linking derived MCCS and MCCS.

Now that we can move freely back and forth between $\approx$ and $\approx_\Pi$, the following theorems are easy.

THEOREM 2.5.9. *Let $Q_1, Q_2 \in \mathbf{P}^C$. Then $Q_1 \approx_\Lambda^\dagger Q_2$ iff $Q_1 \approx_\Pi^\dagger Q_2$.* □

THEOREM 2.5.10. *Let $Q_1, Q_2 \in \mathbf{P}^C$. Then $Q_1 \approx_\Lambda^\times Q_2$ iff $Q_1 \approx_\Pi^\times Q_2$.* □

Lastly, an obvious theorem:

THEOREM 2.5.11. *If E, F are expressions in $\mathbf{P}^C$ that do not involve $|$, then $P \sim Q$ iff $P \sim_\Pi Q$.*

Here we have a simple syntactic test to say when $\sim$ carries over to $\sim_\Pi$.

2.6 Discussion

We have defined MCCS formally and established the congruence $\approx^{cM}$. As to the relations between ASCCS, MCCS and CCS, we have said nothing which is not in some sense obvious. Most of the theorems of the last section can be seen on a closer look directly from [Mil83], at least with hindsight. Even the restricted forms that permit strong congruence results to be carried from MCCS to CCS now involve no |'s, and while we have no simple syntactic characterisation of expressions for which we can carry over strong congruence from SCCS to CCS, we know at any rate that 1. prefixes would not be acceptable. This too is utterly unsurprising.

The major interest appears to be the very existence of MCCS, and its place in the chain of calculi. On the evidence of the last section its particulate subcalculus is close in form to CCS, but as far from it in substance as Milner's particulate calculus of ASCCS is. Thus we might say MCCS is closer to ASCCS than to CCS.

Compared to ASCCS, MCCS seems to the author to have several advantages. Certainly to the user the two look utterly unlike in form. MCCS has no implicit δ's, and no undesirable equivalences like $1.P + P \sim \delta P$. ASCCS has, of course,

an additional operator, $\times$. The price for the $\times$ is a stricter congruence. More light will be thrown on these connections when we study the various operators in Chapter 5.

Derived MCCS (ASCCS without $\times$) has the same congruence $\approx^{cM}$ as primitive MCCS; it looks like that for CCS. But otherwise derived MCCS shares all the disadvantages of ASCCS. In contrast, MCCS seems usable, on the evidence of the few programs in this thesis.

A brief word about the use we shall make of MCCS. We have seen that it suits our needs particularly by not forcing either synchrony or interleaving on us. We shall usually reserve the term "communication" for synchronisation on complementary actions. The great advantage of programming in MCCS (or ASCCS or SCCS) as opposed to CCS is that synchronisation is not restricted to communication; we can get agents to synchronise on actions that are not complements.

However, we don't want to just permit synchronisation, for that would just add to the number of terms in the expansion theorem; we need to be able to enforce synchronisation. Thus the use of restriction to do this is even more important in MCCS and ASCCS than in CCS.

In CCS, $(P \mid Q) \backslash\backslash \{z\}$ has no inferences that result in a z or z^{-1} action. If either P or Q has such an action, it can only be used to communicate with the other. In the other calculi, the restriction is that any action with a z particle in it is forced to synchronise with one with a z^{-1} in it.

By thus enforcing synchronisation between actions that we would be forced to interleave in CCS, we not only reduce the number of states, but also enforce a structure on our systems that permits laws that we could not isolate in CCS. The Synchronised Displacement Theorems of later chapters are our main tools in dealing with the FTS example, and are made possible by the use of synchronising contexts (see Section 3.2.2). That MCCS permits us to do this while retaining a CCS-like feel (no synchrony or idling), is the key to its applicability.

2.7 Attempts to define a two-sorted calculus

A two-sorted calculus is attractive for at least two reasons. One is that it allows us to disallow synchronisation between, say the displace actions of the operators we will define in the next chapter, and other actions: we could do this by making displace actions belong to one sort, and all others to another, and disallow products between the two sorts. Another is that if we want a

primarily asynchronous calculus, one way to achieve this is to have two sorts of actions, one synchronous and the other asynchronous, with products (other than communication) defined only for the former. Programmers could then use as much of each as they wanted. Keeping down the number of synchronous actions has the advantage that the expansion of parallel composition will be tamer.

The author spent several months attempting to define such a calculus. The basic idea was that it should be in some sense derivable from SCCS and CCS. This could be through defining a new (two-sorted) synchronisation algebra, and then using [Win84] to form the category of synchronisation trees induced by the new algebra. Among the many problems in defining such an algebra is that 1 and τ, the invisible action in CCS, are algebraically quite unlike, and it is not all obvious what to do with the two. If $(1, \tau)$ is the unit, for example, what are $(\_, \tau)$ and $(1, \_)$? There are also further complications arising from *inaction*, represented by $*$ in Winskel's paper, and how it affects the two silent actions.

The author was finally persuaded to give up only by the discovery of MCCS. But much of the enthusiasm went out of the project earlier when the following small bug was discovered in [Win84]. The way morphism is defined in Winskel's categories is that $P \xrightarrow{a} P'$ has a morphism to $Q \xrightarrow{b} Q'$ if "the label of the target divides the label of the source". While this works excellently for monoids that are not groups, it does not work for groups, because every element divides every other. Thus we have too many morphisms, resulting in some quite unwanted isomorphisms. We should stress that this does not seriously detract from the merits of [Win84]; to use that paper, we should merely note that we should not work with groups. But since we are specifically interested here in combining SCCS with some form of asynchronous calculus, it becomes impossible to use the definitions in that paper. We end by mentioning that no obvious patches worked. This is probably just as well because the patches were very artificial definitions of morphism.

3

The Displace and Audit Operators

We now develop two operators that together with MCCS will make it easy to reformulate and prove the FTS example of the Appendix. The first operator, *displace*, has counterparts in LOTOS and CSP. The second operator, *audit*, is a generalisation of displace. It does not appear to have any counterparts in the literature.

We deal with the FTS in the following chapter. We shall see that a natural extension to the problem, checkpointing, can be dealt with by means of a third operator, *checkpoint*. The new operators in this chapter are successive specialisations of checkpoint, but seem to be interesting in their own right.

3.1 The Displace operator

We informally met the displace operator, written $\dashv$, in Chapter 1. Consider the following definitions:

$$Q' \Leftarrow f.\mathbf{0} + a.(b.Q' + f.\mathbf{0})$$
$$Q_N \Leftarrow a.b.Q_N$$

where Q_N is the *normal* behaviour of Q', doing a's and b's alternately, while Q' itself is a faulty agent that is prepared to fail at any stage by doing an f. As Q' is written above, the relation between it and Q_N is obscure. There is no obvious way to express this relationship simply, using only the usual operators. For example,

$$Q' \not\dashv \mathrm{fix}X.(Q_N + f.X)$$

because the right hand agent is prepared to fail only in its initial state, and escapes into perfect behaviour as soon as it does an action of Q_N. What we need is some form of "persistent plus", $Q_N \dagger\!\!+ Q_E$, where Q_N cannot escape Q_E by doing an action. We guess that with such an operator we could define a faulty agent Q in the form

$$Q_E \Leftarrow f.\mathbf{0}$$
$$Q \Leftarrow Q_N \dagger\!\!+ Q_E$$

i.e. $Q \equiv Q_N \dagger\!\!+ f.\mathbf{0}$, which clarifies the relation between Q and Q_N. With the following definition of $\dagger\!\!+$, we have $Q \sim Q'$. The bisimulation is $\{\langle Q, Q'\rangle, \langle b.Q_N \dagger\!\!+ f.\mathbf{0}, b.Q' + f.\mathbf{0}\rangle, \langle \mathbf{0}, \mathbf{0}\rangle\}$.

DEFINITION 3.1.1. *The operational semantics of the displace operator.* $\dagger\!\!+$ *is a binary infix operator* $\mathbf{P}^M \times \mathbf{P}^M \to \mathbf{P}^M$.

$$\textit{Normal} \quad \frac{A \xrightarrow{a} A'}{A \dagger\!\!+ B \xrightarrow{a} A' \dagger\!\!+ B}$$

$$\textit{Displace} \quad \frac{B \xrightarrow{b} B'}{A \dagger\!\!+ B \xrightarrow{b} B'}$$

□

We call A above the *normal agent*, and B the *exception agent*. $\dagger\!\!+$ in $A \dagger\!\!+ B$ looks like $+$ to B, but rather like $|$ to A. A cannot get rid of B by doing an action, but B can wipe A out at any time. Synchronisation between the two is not permitted.

A word about the notation. The † symbol would have captured some of the properties of our operator: it is a little like $+$ and a little like $|$. It suitably connotes treachery on the part of B towards A, and foreshadows the doom that hangs perpetually over the latter's head. Unfortunately, it is not asymmetric. It would have been nice to have had a sideways †, but this is unavailable, so we have to accept the $\dagger\!\!+$ symbol as a pale shadow! Notice also that the idea of precedence is not irrelevant to the relation between A and B. However, the use of $\prec$ or any similar symbol would suggest a relation rather than an operator.

The name *displace* is non-committal about the relationship between A and B, but we shall often have in mind applications where B is an interrupt service routine, and the behaviour of $A \dagger\!\!+ B$ after a displace will ultimately be very like the normal behaviour of A again. For example, in $P \equiv \mathrm{fix}X\,(A \dagger\!\!+ f.X)$, the exception event actually *restarts* A. An interesting special case is when $f = 1$. If P is to look like A, then A restarted must look like A continued, a

very strong constraint. In more realistic situations, B will need to keep track of A's evolution, and this can be done with the *audit* and *checkpoint* operators we will look at later.

The "disable" operator of LOTOS [BdN86], [OIS85], [Bri85], has had in the past some proposed variants with exactly the semantics of Displace. Its current version has a third rule that allows the normal agent to get rid of the exception agent by doing a "termination" (a distinguished action).

[Hoa85] deals briefly with an operator called the "interrupt operator". It has exactly the same operational semantics as displace, though this is not formalised as such; the operator is defined instead by axioms. We shall point out connections between displace and CSP's interrupt.

A natural question at this stage is whether $\dagger$ is derivable from MCCS; it is. See Chapter 5.

Let us now put $\dagger$ to use. Consider the following:

$P \Leftarrow pa.vb.P$

$S \Leftarrow p^{-1}.v^{-1}.S$

Here S is a semaphore, and P is a user of S. It is easy to show that

$(P \mid S) \backslash\backslash \{p, v\} \sim a.b.(P \mid S) \backslash\backslash \{p, v\}$

The particles a and b make communication between P and S visible, and we can see from outside that P observes the usual semaphore discipline. The equation above would still hold as a special case even if $a = b = 1$, but would tell us less. More interestingly,

$(P \mid P \mid S) \backslash\backslash \{p, v\} \sim (P \mid S) \backslash\backslash \{p, v\}$

We've deliberately left out any representation of what goes on inside or outside the critical region so as to get this simple characterisation of two identical users sharing a semaphore.

Now consider users Q susceptible to failure. Clearly if they fail inside their critical regions, they hold up other users. Restarting such a Q outside its critical region will deadlock the system, since S will be left in the wrong state.

The way out is to insist that S should be restarted at the same time. Call the restartable semaphore T. Thus we have

$Q \Leftarrow P \dagger f.Q$

$T \Leftarrow S \dagger g.T$

$F \Leftarrow f^{-1}g^{-1}.F$

where f and g represent failures and F is the demon agent. With f and g restricted away, Q and T are *synchronised on their displace actions.* We expect some simple relation between $(Q \mid T \mid F) \backslash\backslash \{f, g\}$ and $P \mid S$, and that

$(Q \mid Q \mid T \mid F) \backslash\backslash \{f,g\} \sim (Q \mid T \mid F) \backslash\backslash \{f,g\}$.

We develop some algebra and study synchronised displacements before we return to this example.

3.1.1 Substitutive properties of Displace

That $\dagger\!\!+$ preserves $\sim$ can be shown by trivial bisimulations. In fact, as we shall see in Chapter 5, any structured operator preserves $\sim$.

PROPOSITION 3.1.2. *$P \sim Q \Longrightarrow P \dagger\!\!+ R \sim Q \dagger\!\!+ R$ and $R \dagger\!\!+ P \sim R \dagger\!\!+ Q$.* □

$\dagger\!\!+$ preserves $\approx$ on the left. *But not on the right.* This is for the same reason that $+$ doesn't preserve $\approx$. We have $a.\mathbf{0} \approx 1.a.\mathbf{0}$ but while $P \dagger\!\!+ 1.a.\mathbf{0}$ can autonomously lose the ability to do P actions, $P \dagger\!\!+ a.\mathbf{0}$ cannot. If the equivalent agents are guarded, we can substitute on the right, as for $+$.

PROPOSITION 3.1.3. *$P \approx Q \Longrightarrow P \dagger\!\!+ R \approx Q \dagger\!\!+ R$ and*
$P \approx Q \Longrightarrow R \dagger\!\!+ a.P \approx R \dagger\!\!+ a.Q$

PROOF. By trivial bisimulations. □

As we would expect from a derived operator, $\dagger\!\!+$ preserves observational congruence, which for MCCS looks exactly like that for CCS; both differ from weak bisimulation only in that if $P \approx^c Q$, 1 actions by P cannot be matched by inaction by Q.

PROPOSITION 3.1.4. *$P \approx^{cM} Q \Longrightarrow P \dagger\!\!+ R \approx^{cM} Q \dagger\!\!+ R$ and $R \dagger\!\!+ P \approx^{cM} R \dagger\!\!+ Q$.*

PROOF. By trivial bisimulations. □

3.1.2 Equational properties of Displace

We can guess from the asymmetry in the definition of $\dagger\!\!+$ that it is not commutative: $P \dagger\!\!+ Q \not\sim Q \dagger\!\!+ P$ and $P \dagger\!\!+ Q \not\approx Q \dagger\!\!+ P$. After an action by P, $P \dagger\!\!+ Q$ can still do actions by Q, whereas $Q \dagger\!\!+ P$ would have lost the ability.

But $\dagger\!\!+$ is associative, and has $\mathbf{0}$ as an identity.

PROPOSITION 3.1.5. *$(P \dagger\!\!+ Q) \dagger\!\!+ R \sim P \dagger\!\!+ (Q \dagger\!\!+ R)$. Also, $P \dagger\!\!+ \mathbf{0} \sim P$ and $\mathbf{0} \dagger\!\!+ P \sim P$. Hence $(\mathbf{P}/\sim, \dagger\!\!+, \mathbf{0})$ is a monoid.*

PROOF. By trivial bisimulations. □

Thus we can write $P \dagger\!\!+ Q \dagger\!\!+ R$ without ambiguity.

Now for $\dagger\!\!+$ and $+$. A number of plausible equations do not hold.

1. $P + (Q \dagger R) \not\sim (P + Q) \dagger R$
2. $(Q \dagger R) + P \not\sim Q \dagger (R + P)$
3. $P + (Q \dagger R) \not\sim (P + Q) \dagger (P + R)$
4. $(Q \dagger R) + P \not\sim (Q + P) \dagger (R + P)$
5. $P \dagger (Q + R) \not\sim (P \dagger Q) + (P \dagger R)$

For 1, an action by P can lose R on the lhs. For 2, 3 and 4, an action by Q can lose P on the lhs. Note that 4 is the same as 3, because $+$ is commutative. For 5, an action by P can lose R on the rhs.

Thus the following is mildly surprising.

PROPOSITION 3.1.6. $(Q + R) \dagger P \sim (Q \dagger P) + (R \dagger P)$

PROOF. With normal actions, whichever of Q or R acts, the other is lost, and it makes no difference whether the choice is internal to the normal component of one agent, or between two displaceable agents. With displacements, the lhs has of course only one way to act; the rhs has the property that the displaced behaviour of both agents is the same.

$\mathcal{N}(P,Q,R) \qquad \langle\, (Q + R) \dagger P, \quad (Q \dagger P) + (R \dagger P)\,\rangle$

Case Normal:

$$\frac{\dfrac{Q \xrightarrow{q} Q'}{Q + R \xrightarrow{q} Q'}}{(Q + R) \dagger P \xrightarrow{q} Q' \dagger P} \qquad \frac{\dfrac{Q \xrightarrow{q} Q'}{Q \dagger P \xrightarrow{q} Q' \dagger P}}{(Q \dagger P) + (R \dagger P) \xrightarrow{q} Q' \dagger P}$$

leading to $\mathcal{I}$

The case where R acts alone is similar to Case Normal

Case Displace:

$$\frac{P \xrightarrow{p} P'}{(Q + R) \dagger P \xrightarrow{p} P'} \qquad \frac{\dfrac{P \xrightarrow{p} P'}{Q \dagger P \xrightarrow{p} P'}}{(Q \dagger P) + (R \dagger P) \xrightarrow{p} P'}$$

leading to $\mathcal{I}$

□

We also note that the following special case of absorption holds.

PROPOSITION 3.1.7. $(A \dagger B) + B \sim A \dagger B$ □

Counterparts to the laws we have seen so far are given in [Hoa85]. The semantics of CSP is not given using bisimulation, but the two theories agree

on all but one of the laws above. We have seen (in 5. above) that $P \upharpoonleft (Q + R) \not\sim (P \upharpoonleft Q) + (P \upharpoonleft R)$, but the corresponding equality in CSP holds if we translate the $+$ of CCS into the internal choice operator of CSP. This is not in itself an indication of the difference between the equivalences used in the two theories, because $+$ is not internal choice in CCS. It might be illuminating to notice that $P \upharpoonleft (a.Q + a.R) \not\sim (P \upharpoonleft a.Q) + (P \upharpoonleft a.R)$, where the $+$'s now are "internal" in the sense that the outside world cannot determine which choice is made. The inequality remains because the actions by P retain both Q and R possibilities on the left, but have to drop one on the right.

Next we look for exactly similar inequalities between $\upharpoonleft$ and $|$, and now we find that there are no exceptions; none of the laws hold.

1. $P \mid (Q \upharpoonleft R) \not\sim (P \mid Q) \upharpoonleft R$
2. $(Q \upharpoonleft R) \mid P \not\sim Q \upharpoonleft (R \mid P)$
3. $P \mid (Q \upharpoonleft R) \not\sim (P \mid Q) \upharpoonleft (P \mid R)$
4. $(Q \upharpoonleft R) \mid P \not\sim (Q \mid P) \upharpoonleft (R \mid P)$
5. $P \upharpoonleft (Q \mid R) \not\sim (P \upharpoonleft Q) \mid (P \upharpoonleft R)$
6. $(Q \mid R) \upharpoonleft P \not\sim (Q \upharpoonleft P) \mid (R \upharpoonleft P)$

For 1, an action by R can lose P on the rhs. For 2 and 3, an action by P can lose Q on the rhs. 4 is the same as 3, because $|$ is commutative. For 5, an action by Q can lose P on the lhs, whereas one of the P's on the rhs will remain. For 6, an action by P can lose both Q and R on the lhs, but can only lose one of these on the rhs.

In MCCS, as in CCS, $+$ is idempotent and $|$ is not. It is easy to see that $\upharpoonleft$ is neither, in general.

Restriction distributes over $\upharpoonleft$. As usual, we are interested only in the particulate form of restriction, and we can guess that this result is true because it holds for $+$ and for $|$ in the case where the agents on either side of the $|$ cannot communicate with each other.

PROPOSITION 3.1.8. $(P \dagger Q)\backslash\mathcal{N} \sim (P\backslash\mathcal{N}) \dagger (Q\backslash\mathcal{N})$

PROOF. By the following bisimulation.

$$\mathcal{R}(P,Q) \qquad \langle\, (P \dagger Q)\backslash\mathcal{N},\ \ (P\backslash\mathcal{N}) \dagger (Q\backslash\mathcal{N}) \,\rangle$$

Case Normal:

$$\frac{P \xrightarrow{a} P'}{\dfrac{P \dagger Q \xrightarrow{a} P' \dagger Q \quad a \notin \mathcal{N}}{(P \dagger Q)\backslash\mathcal{N} \xrightarrow{a} (P' \dagger Q)\backslash\mathcal{N}}} \qquad \frac{\dfrac{P \xrightarrow{a} P' \quad a \notin \mathcal{N}}{P\backslash\mathcal{N} \xrightarrow{a} P'\backslash\mathcal{N}}}{(P\backslash\mathcal{N}) \dagger (Q\backslash\mathcal{N}) \xrightarrow{a} (P'\backslash\mathcal{N}) \dagger (Q\backslash\mathcal{N})}$$

leading to $\mathcal{R}(P',Q)$

Case Displace:

$$\frac{Q \xrightarrow{a} Q'}{\dfrac{P \dagger Q \xrightarrow{a} Q' \quad a \notin \mathcal{N}}{(P \dagger Q)\backslash\mathcal{N} \xrightarrow{a} Q'\backslash\mathcal{N}}} \qquad \frac{\dfrac{Q \xrightarrow{a} Q' \quad a \notin \mathcal{N}}{Q\backslash\mathcal{N} \xrightarrow{a} Q'\backslash\mathcal{N}}}{(P\backslash\mathcal{N}) \dagger (Q\backslash\mathcal{N}) \xrightarrow{a} Q'\backslash\mathcal{N}}$$

leading to $\mathcal{R}(P,Q')$

□

An easy bisimulation shows that morphism distributes over $\dagger$.

PROPOSITION 3.1.9. $(P \dagger Q)[\phi] \sim (P[\phi]) \dagger (Q[\phi])$ □

3.2 Synchronised displacements

We now come to the Synchronised Displacement Theorem (SDT) for the displace operator. The statement of the theorem is: "In any context that synchronises the displace actions of $A \dagger B$ and $P \dagger Q$, $(A \dagger B) \mid (P \dagger Q) \sim (A \mid P) \dagger (B \mid Q)$."

We talked about synchronising contexts briefly in Section 2.6; they represent our first serious use of MCCS instead of CCS. We must now define what we mean by the term. Also, the form of the theorem is open to criticism: it does not establish a bisimulation pure and simple; it says rather that a bisimulation exists in certain contexts. This needs clarification as well.

[Lar86] provides us with an elegant and thoroughly worked out theory of *parameterised bisimulation* and *context dependent bisimulation*, which we can use to explain the SDT. We should say right away that we shall not be using the formal machinery of [Lar86]; our needs are too simple to warrant such elaborate apparatus. Indeed, early versions of the SDT with essentially the same content were developed before Larsen's theory was available, and even

the current version was arrived at without using it. However, we will outline a trivial version of the theory here to give a clear and secure interpretation of our work. The interpretation will help ensure that our informal arguments are nonetheless consistent and rigorous.

3.2.1 Context dependent and parameterised bisimulations

Consider the following definitions:

$Q \Leftarrow q.Q + fz.Q$

$S \Leftarrow s.S + gz^{-1}.S$

An observer of $(Q \mid S) \backslash\backslash \{z\}$ will see q's and s's in any order and either singly or together, but f and g always together. Now consider the following agents:

$N \Leftarrow q.N + s.N + qs.N$

$E \Leftarrow fg.E$

We have $(Q \mid S) \backslash\backslash \{z\} \sim (N \mid E) \backslash\backslash \{z\}$ but not $Q|S \sim N|E$. We say "$Q|S \sim N|E$ *in the context* $[\,]\backslash\backslash\{z\}$". In this example, we also have $(N \mid E) \backslash\backslash \{z\} \sim N \mid E$, but that is irrelevant to our present purposes.

$[\,]\backslash\backslash\{z\}$ is not the only context that makes the two systems $Q \mid S$ and $N \mid E$ equivalent. Let

$F \Leftarrow f^{-1}g^{-1}.F$

Then we have $(Q \mid S \mid F) \backslash\backslash \{f, g\} \sim (N \mid E \mid F) \backslash\backslash \{f, g\}$. Thus we have found another context in which $Q \mid S$ and $N \mid E$ are equivalent; $Q \mid S \sim N \mid E$ in the context $([\,] \mid F) \backslash\backslash \{f, g\}$.

These context dependent bisimulations are generalisations of the ordinary notion of bisimulation, which can now be seen as "equivalence in all contexts". If $P \sim Q$, we certainly have $P \backslash\backslash \{z\} \sim Q \backslash\backslash \{z\}$, but as we saw above the converse is not true: we have $(Q \mid S) \backslash\backslash \{z\} \sim (N \mid E) \backslash\backslash \{z\}$ but not $Q \mid S \sim N \mid E$.

Now, $Q \mid S$ by itself can do q, s, qs, fg, and fz, gz^{-1}, sfz and qgz^{-1}. $N \mid E$ can only do the first four. If we call the first four actions *acceptable* and the others unacceptable to the context $[\,]\backslash\backslash\{z\}$, we can say that $Q \mid S \not\sim N \mid E$ because of the unacceptable actions.

A context is a CCS expression with a hole in it. Contexts can be described operationally as transducers [Lar86]: they accept actions from the agent that is put in the hole, and transmit (other) actions to the outside. We are here concerned with only half of this description: each context can be thought of as inducing a set of acceptable actions.

DEFINITION 3.2.1. *An action a is acceptable to a context $C[\ ]$ iff $\forall P, P'$ such that $P \xrightarrow{a} P'$, $\exists b, C'$ such that $C[P] \xrightarrow{b} C'[P']$.* □

To take our other example, $([\ ] \mid F) \backslash\backslash \{f, g\}$ can be seen intuitively to accept only those that either contain no f, g, f^{-1}, or g^{-1} particles, or contain these in precisely the combination fg. The b of the definition is the same as the accepted action, except in the last case, where the fg is stripped off. In this example, C' is the same as C, but this need not be, as we will see in other examples. We can check that the unacceptable set includes the four actions fz, gz^{-1}, sfz, qgz^{-1}.

The reader can see intuitively that contexts that have unacceptable actions must involve the restriction operator.

Note that the definition cannot specify what action results, or what the resulting context and agent in the hole look like. Can a context cheat, by producing the b and $C'[P']$ from inside itself, and thus only pretend to accept the action? I.e., Can we get $C[P] \xrightarrow{b} C'[P]$, the inference not having used $P \xrightarrow{a} P'$, and the context C' happening to have P' in it, so that we can interpret $C'[P]$ as $C''[P']$? No, because of the following fact. [Lar86] proves (Proposition 3.2-4) by induction on the structure of C, that the number of pairs $\langle b, C' \rangle$ for given a is finite. (The proof assumes image finiteness). We can extend his proof for CCS to MCCS. Thus the context cannot cheat for all P, P'. It can mimic only a finite number of P's, whereas we can always find another pair of agents $Q \xrightarrow{a} Q'$; we only have to put $a.$ in front of an arbitrary agent.

Given any pair of agents, we can find out if they are equivalent in a given context, by using the idea of *parameterised bisimulation*. In the trivial version of parameterised bisimulation that we use, the agents need only be able to match each other on actions acceptable to the context. We temporarily write $\sim_{\mathcal{A}}$ to represent such a bisimulation, where the subscript specifies the acceptable subset of Act. (We have usually avoided naming sets of actions, and worked with sets of particles instead; the deviation here is a temporary expedient). $\sim_{\emptyset}$ would be the same as $\mathbf{P} \times \mathbf{P}$; since all actions are unacceptable and to be ignored, every agent can match every agent. At the other end, $\sim_{Act}$ would be the same as $\sim$.

In our example above, $Q|S \sim_{\mathcal{A}} N|E$ where $\mathcal{A} = Act - \{fz, gz^{-1}, sfz, qgz^{-1}\}$. The reader can see that if we used a larger set in the subscript, the statement would be false; if we used a smaller set, we would be making an unnecessarily

weak statement. For example, if we let $\mathcal{NZ}$ represent the subset of actions with neither a z or z^{-1} particle, then $\sim_{\mathcal{NZ}}$ has a smaller set of acceptable actions than $\sim_{\mathcal{A}}$ and so relates more pairs than the latter. In general, if $\mathcal{A} \subseteq \mathcal{B}$ then $\sim_{\mathcal{A}} \supseteq \sim_{\mathcal{B}}$. $\sim_{\mathcal{A}}$ is an equivalence relation for any set of actions $\mathcal{A}$.

A parameterised bisimulation itself refers only to the two agents involved; it tells us which actions we need consider. Then any context (that induces a parameterised bisimulation) that marks at most those actions acceptable, will equate the two agents we started with.

To return to the examples, we can see intuitively that $X \backslash\backslash \{z\} \sim Y \backslash\backslash \{z\}$ iff $X \sim_{\mathcal{NZ}} Y$. So because $\mathcal{NZ} \subseteq \mathcal{A}$, $X \sim_{\mathcal{A}} Y \Rightarrow X \sim_{\mathcal{NZ}} Y \Leftrightarrow X \backslash\backslash \{z\} \sim Y \backslash\backslash \{z\}$. Thus because $Q \mid S \sim_{\mathcal{A}} N \mid E$, we have $(Q \mid S) \backslash\backslash \{z\} \sim (N \mid E) \backslash\backslash \{z\}$. Similarly, the set of actions acceptable to $([\,] \mid F) \backslash\backslash \{f, g\}$ explains why $(Q \mid S \mid F) \backslash\backslash \{f, g\} \sim (N \mid E \mid F) \backslash\backslash \{f, g\}$.

Let us look again at the theorem to see how far we have got: "If the restart actions of $A \dashv\!\vdash B$ and $P \dashv\!\vdash Q$ are synchronised, then $(A \dashv\!\vdash B) \mid (P \dashv\!\vdash Q) \sim (A \mid P) \dashv\!\vdash (B \mid Q)$." We now think of this as establishing a parameterised bisimulation, one of the simplest sort: it marks a subset of *Act* unacceptable. It allows us to substitute the rhs for the lhs in any context that marks the same or a larger subset of *Act* unacceptable.

3.2.2 Synchronising contexts

We make a preliminary definition first, a verbal abbreviation.

DEFINITION 3.2.2. *Let $\mathcal{A}$ be a set of particles. We say a is an $\mathcal{A}$-action if $a \ll \mathcal{A}$.* □

Thus if f is a particle, an $\{f\}$-action is an action with an f particle in it. We abuse notation and write $\{a\}$-action for any action a, to mean $\mathrm{Part}(a)$-action.

Let us modify the example of the last subsection as follows.

$Q \Leftarrow q.Q + rf.Q$
$S \Leftarrow s.S + tg.S$
$F \Leftarrow f^{-1}g^{-1}.F$

We now refer to rf as Q's $\{f\}$-action, and to tg as S's $\{g\}$-action. Note that an agent P might have an $\{a\}$-action, but yet not have an action $P \xrightarrow{a} P'$.

In the context $([\,] \mid F) \backslash\backslash \{f, g\}$, the agent $Q \mid S$ is restricted in the following way: Q can do its $\{f\}$-action only when S does its $\{g\}$-action and vice-versa. We say the context *synchronises* $\{f\}$- and $\{g\}$-actions (of Q and S in this case, but the context doesn't care). The formal definition follows shortly; first what

an observer of $(Q \mid S \mid F) \backslash\backslash \{f, g\}$ sees:

DEFINITION 3.2.3. *Let $\mathcal{A}$ and $\mathcal{B}$ be sets of particles. The $\mathcal{A}$-actions of an agent P are said to be synchronised with its $\mathcal{B}$-actions if whenever $P \xrightarrow{a}$, $a \lessdot \mathcal{A} \Leftrightarrow a \lessdot \mathcal{B}$.* □

So the $\{r\}$-actions of $(Q \mid S \mid F) \backslash\backslash \{f, g\}$ are synchronised with its $\{t\}$-actions.

The definition of synchronisation allows other possibilities, but we shall only see cases where the P' uses either no $\mathcal{A}$-actions and no $\mathcal{B}$-actions, or precisely one $\mathcal{A}$-action and precisely one $\mathcal{B}$-action.

1 cannot be synchronised with any action, since $1 \notin \mathcal{A}$ for any $\mathcal{A}$. The result of a synchronisation can be a 1; the definition allows us to synchronise an $\{a\}$-action with an $\{a^{-1}\}$-action. In other words, communication is a special case of synchronisation, as usual. It could be of interest to define *external synchronisation* as a special case of synchronisation: we add the restriction that $\mathcal{A} \cap \overline{\mathcal{B}} = \emptyset = \overline{\mathcal{A}} \cap \mathcal{B}$. This means that the result of a synchronisation is always a visible action. Most of our applications will be of this nature, but we do not need the restriction for the SDT.

In the example above, $\{f\}$-, $\{g\}$-, $\{r\}$- and $\{t\}$-actions all occur together, but the most significant synchronisation there is that between $\{r\}$- and $\{t\}$-actions; this is what is seen outside the context, and a common programming scenario is that this synchronisation is the *raison d'etre* for F and the f and g particles in the actions of Q and S. That $\{f\}$- and $\{g\}$-actions are synchronised inside the context is characteristic of the context:

DEFINITION 3.2.4. *Let $\mathcal{A}$ and $\mathcal{B}$ be sets of particles, and $\mathcal{C}[\ ]$ a context. $\mathcal{A}$-actions and $\mathcal{B}$-actions are synchronised in the context $\mathcal{C}$ if whenever a is acceptable to $\mathcal{C}$, $a \lessdot \mathcal{A} \Leftrightarrow a \lessdot \mathcal{B}$.* □

If the condition above is the defining condition for acceptability of an action to $\mathcal{C}[\ ]$, we say $\mathcal{C}[\ ]$ is a synchronising context. Note that 1 is always an acceptable action to such a context since it fulfils the synchronisation condition vacuously.

We are almost done. To tie up the discussion, consider the following variation of the example of the last subsection.

$$Q \Leftarrow qr.Q + rf.Q$$
$$S \Leftarrow s.S + tg.S$$
$$N \Leftarrow qr.N + s.N + qrs.N$$
$$E \Leftarrow rtfg.(N \mid E)$$
$$F \Leftarrow f^{-1}g^{-1}.F$$

Now we cannot say that $([\,] \mid F) \backslash\backslash \{f, g\}$ synchronises the $\{r\}$-actions of Q with the $\{t\}$-actions of S, because both of Q's actions are $\{r\}$-actions, and only the second is synchronised with the $\{g\}$-action of S. While we can say that it synchronises the $\{f\}$-actions of Q with the $\{g\}$-actions of S, we have seen that this is not a very meaningful statement for the programmer. It is better to characterise the situation thus: the context synchronises the displace actions of Q and S.

We are at last ready to formulate the statement we want: "in any context $\mathcal{C}[\,]$ that synchronises the displace actions of Q and S, $\mathcal{C}[Q \mid S] \sim \mathcal{C}[N \mid E]$". We prove the statement by showing a parameterised bisimulation between $Q \mid S$ and $N \mid E$.

The acceptable actions of $Q \mid S$ are those generated by inferences that use either both or neither of the displace actions. Note that this is an intensional description. To find out the acceptable set, we have to look at the (insides of) the agents Q and S, in order to tell when they are doing displace actions. Thus the "if" part of the statement tells us explicitly which inferences of $Q \mid S$ are acceptable: normal actions singly or together, displace actions together. We can find matching inferences for $N \mid E$; since they match, they will be acceptable, so we don't have to worry about acceptability for this direction of the proof.

The question is: what are the acceptable inferences of the rhs (that the lhs has to match)? The condition does not tell us directly, since neither Q nor S appear in the rhs. One way is to list out the acceptable actions resulting from the lhs, and consider all inferences of the rhs that produce one of these. This is possible for concrete examples, but is not possible for the SDT, which, like all useful general theorems, sets up an equivalence between expressions rather than given agents. There, we use the structural relation between the lhs and the rhs to determine which inferences of the latter are acceptable.

Once we have proved the SDT, the next question is: how do we know when it is applicable? It is usually quite easy to see when we have a context that synchronises the displace actions of two agents, but let us formalise once and for all the situation we will meet in our applications of the SDT.

PROPOSITION 3.2.5. *Let Q, S, D be agents and f, g, h actions such that $g, h \notin \mathcal{Q}$, $f, h \notin \mathcal{S}$, and $f, g \notin \mathcal{D}$. Suppose also that $f^{-1}, g^{-1}, h^{-1} \notin \mathcal{Q}$, and similarly for $\mathcal{S}$ and $\mathcal{D}$. Let $F \Leftarrow f^{-1}g^{-1}h^{-1}.F$. Then the $\{f\}$-actions of Q and the $\{g\}$-actions of S are synchronised in the context $([\,] \mid D \mid F) \backslash\backslash \{f, g, h\}$.* □

In fact these actions will be synchronised with the $\{h\}$-actions of D, but that might not be relevant to the immediate application of the proposition. Putting $D = \mathbf{0}$ and $h = 1$ in the proposition gives us the case where Q and S are the only agents in the context $([\,] \mid F) \backslash\backslash \{f, g\}$.

We shall program so that an action of Q carries an f particle iff it is a displace; and similarly for S and D with g and h. Note in particular that none of the agents has a 1 as a displace action.

3.2.3 The Synchronised Displacement Theorem (SDT)

THEOREM 3.2.6. ***SDT (for the Displace operator)***
In any context $\mathcal{C}[\,]$ that synchronises the displace actions of $A \dotplus B$ and $P \dotplus Q$, $\mathcal{C}[(A \dotplus B) \mid (P \dotplus Q)] \sim \mathcal{C}[(A \mid P) \dotplus (B \mid Q)]$.

PROOF. The form of the statement is chosen to show up the relation to the general versions of the SDT to follow. The condition can immediately be simplified to "If B and Q are synchronised", because the displace actions of $A \dotplus B$ are precisely those of B.

The left hand agent can act in the following ways:
$A[P]$: Either component can do a normal action;
AP: two normal actions can synchronize;
BQ: two displacements can synchronize.
The displacements cannot occur independently because of the constraint; similarly, A cannot synchronize with Q because B is forced to act with Q, and that precludes an action by A.

We start with the acceptable inferences of the rhs. It can do a normal action in three ways: A alone or P alone (covered by case $A[P]$ above), or A and P together (case AP). A displace can happen with B and Q acting together (case BQ). Note that we do not bother to prove separately that these are acceptable inferences; we shall just see that they can be matched.

We have to show that inferences with B (or Q) acting alone are unacceptable. For this simple theorem, we need only say that this is directly given by the condition. Notice that the condition of the theorem implies that both B and Q are stable. If it were not for this fact, the rhs could have had an inference

$$\frac{\dfrac{B \xrightarrow{1} B'}{B \mid Q \xrightarrow{1} B' \mid Q}}{(A \mid P) \dotplus (B \mid Q) \xrightarrow{1} B' \mid Q}$$

which cannot be matched, and cannot be ruled unacceptable because 1 is always acceptable to a synchronising context.

The bisimulation for the acceptable inferences follows.

$$\mathcal{N}(A,B,P,Q) \qquad \langle\, (A \dagger B) \mid (P \dagger Q),\quad (A \mid P) \dagger (B \mid Q)\,\rangle$$

Case A[P]:

$$\frac{\dfrac{A \xrightarrow{a} A'}{A \dagger B \xrightarrow{a} A' \dagger B}}{(A \dagger B) \mid (P \dagger Q) \xrightarrow{a} (A' \dagger B) \mid (P \dagger Q)} \qquad \frac{\dfrac{A \xrightarrow{a} A'}{A \mid P \xrightarrow{a} A' \mid P}}{(A \mid P) \dagger (B \mid Q) \xrightarrow{a} (A' \mid P) \dagger (B \mid Q)}$$

leading to $\mathcal{N}(A', B, P, Q)$

Case AP:

$$\frac{\dfrac{A \xrightarrow{a} A'}{A \dagger B \xrightarrow{a} A' \dagger B} \quad \dfrac{P \xrightarrow{p} P'}{P \dagger Q \xrightarrow{p} P' \dagger Q}}{(A \dagger B) \mid (P \dagger Q) \xrightarrow{ap} (A' \dagger B) \mid (P' \dagger Q)} \qquad \frac{\dfrac{A \xrightarrow{a} A' \quad P \xrightarrow{p} P'}{A \mid P \xrightarrow{ap} A' \mid P'}}{(A \mid P) \dagger (B \mid Q) \xrightarrow{ap} (A' \mid P') \dagger (B \mid Q)}$$

leading to $\mathcal{N}(A', B, P', Q)$

Case BQ:

$$\frac{\dfrac{B \xrightarrow{b} B'}{A \dagger B \xrightarrow{b} B'} \quad \dfrac{Q \xrightarrow{q} Q'}{P \dagger Q \xrightarrow{q} Q'}}{(A \dagger B) \mid (P \dagger Q) \xrightarrow{bq} B' \mid Q'} \qquad \frac{\dfrac{B \xrightarrow{b} B' \quad Q \xrightarrow{q} Q'}{B \mid Q \xrightarrow{bq} B' \mid Q'}}{(A \mid P) \dagger (B \mid Q) \xrightarrow{bq} B' \mid Q'}$$

leading to $\mathcal{N}(A, B', P, Q')$

□

The role of the synchronisation condition in the proof

The synchronisation condition rules out the awkward cases in the bisimulation where one side cannot match a move by the other. If the components in the left hand agent are allowed to do independent displacements, or if one component can do a normal action and synchronise with a displace action by the other, we get into trouble. The condition is refined; it rules out just these cases and no

others. We could say our specification of the contexts in which the equivalence holds is as weak as possible.

Case B[Q]: Independent Displacement, leading to agents differing in form:

$$\frac{\dfrac{B \xrightarrow{r} B'}{A \dagger B \xrightarrow{r} B'}}{(A \dagger B) \mid (P \dagger Q) \xrightarrow{r} B' \mid (P \dagger Q)} \qquad \frac{\dfrac{B \xrightarrow{r} B'}{B \mid Q \xrightarrow{r} B' \mid Q}}{(A \mid P) \dagger (B \mid Q) \xrightarrow{r} B' \mid Q}$$

leading to $\mathcal{MISMATCH}$

Case AQ[BP]: Normal combining with Displace, leading to agents different in form; actions also differ:

$$\frac{\dfrac{A \xrightarrow{a} A'}{A \dagger B \xrightarrow{a} A' \dagger B} \quad \dfrac{Q \xrightarrow{q} Q'}{P \dagger Q \xrightarrow{q} Q'}}{(A \dagger B) \mid (P \dagger Q) \xrightarrow{aq} (A' \dagger B) \mid Q'} \qquad \frac{\dfrac{Q \xrightarrow{q} Q'}{B \mid Q \xrightarrow{q} B \mid Q'}}{(A \mid P) \dagger (B \mid Q) \xrightarrow{q} B \mid Q'}$$

leading to $\mathcal{MISMATCH}$

□

Consider other conditions under which the theorem could hold. For instance, could constraints on sorts rule out Case AQ? We might then be able to have an ordinary bisimulation, and avoid context dependent bisimulations. It is not obvious how to do this in MCCS, but in a two sorted calculus we could have all displace actions belong to one sort and normal actions to the other, with products between different sorts undefined. But such conditions can only rule out synchronisation, not enforce it. So Case B would still lead naturally to enforced synchronisation as a condition, and Case AQ would then get ruled out automatically. Thus we have nothing to gain in this case from a two-sorted calculus.

We now give two corollaries of the SDT.

COROLLARY 3.2.7. *In any context* $C[\,]$ *that synchronises* B *and* F, $C[(A \dagger B) \mid F] \sim C[A \dagger (B \mid F)]$

PROOF. $F \sim \mathbf{0} \dagger F$. □

So if the displace actions are all triggered by a demon, we can put the demon in parallel with the exception agent. Note that the synchronisation condition implies that the demon does not synchronise or communicate with the normal

agent. Further, if the exception agent does immediate restarts, we get a system that restarts autonomously. Put $p = 1$ in the following corollary.

COROLLARY 3.2.8. *Let* $P \Leftarrow A \dagger pf.P$, $F \Leftarrow f^{-1}.F$, *and* $f, f^{-1} \notin A$. *Then* $(P \mid F) \setminus\setminus \{f\} \sim S$, *where* $S \Leftarrow A \dagger p.S$. □

Asymmetry in the statement of the SDT

The synchronisation condition of the SDT is not only intensional, it is also asymmetric, since the contexts under which the equivalence holds are specified in terms of the lhs. Thus given a context and the lhs, we can easily see whether we can replace the lhs by the rhs, but if we start with a context and the rhs, we cannot immediately see if the theorem is applicable.

Predictably, then, we will always use the theorem left to right.

The lhs also has more of the flavour of an implementation than the rhs, since the intensionality in the condition applies only to it. This is borne out by our applications, where the rhs is usually a simplification (specification) of the lhs. Given the directionality of application, this seems to make the SDT a "bottom-up theorem"; in so far as an equivalence can be either top-down or bottom-up.

Since in our applications of the SDT we always use one particular form of context, we could consider a form of the theorem that started with it and specified the acceptable set. We would then avoid specifying the acceptable inferences, and our justification of the proof method could be less long-winded. Some earlier versions of the theorem can with hindsight be interpreted as attempts in this direction. But those versions still equated two agents with the same forms as in the present SDT, and the end result was that the agents were specified very much more tightly; for example, with only one given displace action permitted. Thus no feeling of top-down-ness resulted automatically because we started with the context. A further price was that the notion of synchronisation remained ill defined. The theorems were symmetric, but this was not particularly valuable.

A more basic asymmetry in the SDT is that we cannot in general guess at the lhs starting from the rhs, because $\mid$ is symmetric: from $(A \mid P) \dagger (B \mid Q)$ we cannot construct the lhs, because we cannot tell whether A goes with B or with Q. There is more information in the lhs, which is another reason for considering it the implementation. If it were not for this loss of information, the earlier asymmetry would only be an inconvenience in that given the rhs and a context, we would have to construct the lhs before we could check the context for suitability.

As it turns out, the symmetry of | leads to a curious result for $\dagger$. Suppose that we start with the rhs of the SDT, and foolishly try to construct the lhs anyway. For $\dagger$, we can't go wrong!

PROPOSITION 3.2.9. *In a context that synchronises the displace actions of* $A \dagger B$ *and* $P \dagger Q$, $(A \dagger B) \mid (P \dagger Q) \sim (A \dagger Q) \mid (P \dagger B)$.

PROOF. A context that synchronises the displace actions of the lhs also synchronises the displace actions of the rhs. Both are then equivalent to $(A \mid P) \dagger (B \mid Q)$, by the SDT. □

Which says that it doesn't matter if the programmer attaches exception agents to normal agents wrongly! Is this a disaster? No. The result is true only under the synchronisation condition, and then only because with $\dagger$ the exception agents don't interact at all with their normal wards. We just have a set of normal agents, all of which will be wiped out simultaneously and be replaced by the set of exception agents.

This curiosity does not hold for the audit and checkpoint operators, because there we have a linkage between normal and exception agents. Technically, the attempt to run the SDT backwards is also worse with those operators because, as we shall see, there is greater loss of information from left to right.

3.2.4 The semaphore example

Here are the definitions again:

$P \Leftarrow pa.vb.P$ — *the perfect user*

$S \Leftarrow p^{-1}.v^{-1}.S$ — *the semaphore*

$Q \Leftarrow P \dagger f.Q$ — *the faulty user*

$T \Leftarrow S \dagger g.T$ — *the faulty semaphore*

$F \Leftarrow f^{-1}g^{-1}.F$ — *the demon*

In the context $([\,] \mid F) \backslash\backslash \{f, g\}$, Q and T are synchronised on displace actions. Let

$A_N \Leftarrow (P \mid S) \backslash\backslash \{p, v\}$ — *the perfect system*

$A \Leftarrow (Q \mid T \mid F) \backslash\backslash \{f, g, p, v\}$ — *the fault tolerant system*

Then

$A_N \sim a.b.A_N$

$A \sim ((Q \mid T) \backslash\backslash \{p, v\} \mid F) \backslash\backslash \{f, g\}$ — *moving* $\backslash\backslash \{p, v\}$ *over* F

and

$(Q \mid T) \backslash\backslash \{p, v\} \sim ((P \mid S) \dagger fg.(Q \mid T)) \backslash\backslash \{p, v\}$ — *by SDT*

$\sim A_N \dagger fg.(Q \mid T) \backslash\backslash \{p, v\}$

distributing restriction over $\dagger$*; SDT eq*

So

$$\begin{aligned} A &\sim ((Q \mid T) \setminus\setminus \{p,v\} \mid F) \setminus\setminus \{f,g\} \\ &\sim ((A_N \dagger fg.(Q \mid T) \setminus\setminus \{p,v\}) \mid F) \setminus\setminus \{f,g\} && \textit{SDT in context} \\ &\sim A_N \dagger 1.A && \textit{by Corollary 3.2.8} \end{aligned}$$

Thus the faulty system A has a simple relationship to the original system $(P \mid S) \setminus\setminus \{p,v\}$, but they are not observationally equivalent, for Q might fail after an a, and get backed up to before it, thus producing two consecutive a's. We have to be able to back up to different points to be able to avoid this. (Note that we will not get two b's. The asymmetry is because of the restart state.) If we put $a = 1$, we get $A \approx A_N$.

But we can prove that the system recovers from faults without deadlocking, and that two users look like one in this context. The demon must now cause both to fail at the same time as the semaphore. (Obviously, we would like to exempt the user outside its critical region from this community failure! But more of this later).

Let

$$\begin{aligned} F_2 &\Leftarrow f^{-2}g^{-1}.F_2 && \textit{fails semaphore and both users} \\ B_N &\Leftarrow (P \mid P \mid S) \setminus\setminus \{p,v\} && \textit{two user perfect system} \\ B &\Leftarrow (Q \mid Q \mid T \mid F_2) \setminus\setminus \{f,g,p,v\} && \textit{two user fault-tolerant system} \end{aligned}$$

Then

$$\begin{aligned} B_N &\sim a.b.B_N \\ &\sim A_N && \textit{as expected for the fault-free users} \end{aligned}$$

and

$$\begin{aligned} (Q \mid Q \mid T) \setminus\setminus \{p,v\} &\sim (Q \mid ((P \mid S) \dagger fg.(Q \mid T))) \setminus\setminus \{p,v\} && \textit{by SDT} \\ &\sim \big((P \mid P \mid S) \dagger f^2g.(Q \mid Q \mid T)\big) \setminus\setminus \{p,v\} && \textit{by SDT} \\ &\sim B_N \dagger f^2g.(Q \mid Q \mid T) \setminus\setminus \{p,v\} \\ & && \textit{distributing restriction over } \dagger\textit{; SDT eq.} \end{aligned}$$

So

$$\begin{aligned} B &\sim ((Q \mid Q \mid T) \setminus\setminus \{p,v\} \mid F_2) \setminus\setminus \{f,g\} \\ &\sim \big((B_N \dagger f^2g.(Q \mid Q \mid T) \setminus\setminus \{p,v\}) \mid F_2\big) \setminus\setminus \{f,g\} && \textit{SDT in context} \\ &\sim B_N \dagger 1.B && \textit{by Corollary 3.2.8} \\ &\sim A && \textit{defined by the same equation as } B \end{aligned}$$

Thus the two-user fault-tolerant system is congruent to the single-user fault tolerant system.

3.2.5 Would a restart operator do better?

Consider how the previous example would look without the new operator, particularly in CCS with interleavings and protocols. We appear in comparison to have done better, but our modest little success has come at a price: MCCS, $\dagger$, and a strong constraint, the synchronisation condition. Worse, all three seem linked. It is time to take stock.

The move to MCCS from CCS has ruled out the protocol problem by fiat, as we saw in Chapter 1, but because it is not a "natural" part of the problem, we don't feel too guilty about this. Both MCCS and $\dagger$ might be useful elsewhere as well. It is the SDT and the synchronisation condition that need comment.

The SDT is as significant for the conditions under which it holds, and the general framework necessary (MCCS, some form of synchronisation, a simple version of context-dependent bisimulation), as for its statement. In fact the design decisions underlying the three checkpoint operators are interwoven with the SDT, and both operators and theorem need synchronisation; since SCCS and ASCCS are unsuitable, this leads to MCCS. While it seems reasonable to suppose that MCCS is independently applicable, the same cannot be said of the other components of the package, particularly the more complicated audit and checkpoint operators. The whole lot should thus be seen together as a suggested analysis of restartable systems.

Can we do better with a different operator? We are using $\dagger$ in only a limited way, to restart. Suppose we define a restart operator directly. Would its more restricted behaviour give better theorems?

DEFINITION 3.2.10. *The operational semantics of the binary restart operator,* ***rest****.*

$$\textit{Normal}\quad \frac{E \xrightarrow{a} E'}{E\mathbf{rest}F \xrightarrow{a} E'\mathbf{rest}F}$$

$$\textit{Restart}\quad \frac{F \xrightarrow{a} F'}{E\mathbf{rest}F \xrightarrow{a} F'\mathbf{rest}F}$$

□

We can use this operator to represent faulty versions etc. very much as we used $\dagger$. Most of the properties of the $\dagger$ carry over to **rest**, some with minor modifications. The SDT carries over as well, with the same condition. Thus we cannot do any better with this more structured operator.

In fact any attempt to analyse exception handling along these lines leads naturally to the SDT condition. If we want an operator ρ to have the property

$$\rho(N_1, E_1) \mid \rho(N_2, E_2) \sim \rho((N_1 \mid N_2), (E_1 \mid E_2))$$

where the semantics of ρ involve a suspension of normal processing upon the exception event, we see that the rhs cannot suspend normal processing in just one component. Which leads us to ask why we want such a theorem in the first place. Our long list of inequalities shows that other ways of analysing the structure are not likely to be very successful, at least with $\dagger$ or **rest**. The sort of distributivity we have in the SDT is the best we have done so far towards a simple analysis of distributed restartable systems. Thus we seem to be stuck with synchronised displacements. The semaphore example shows that they are not unnatural.

We conclude this section by mentioning some related work. We have already mentioned CSP's interrupt operator and the disable operator of LOTOS. CSP also has a restart operator which can be translated as $restart_f(P) \sim P \dagger f.P$. However, this translation suggests that there is a family of operators, one for each f. In fact, CSP uses a distinguished action instead of any f, and so has only one restart operator.

[HH86] solve a problem closely related to the FTS problem of [Pra84]. Differences in framework and formulation, as also the difference in the equivalence relation used, make the connection between the two papers non-obvious. For our purposes, we note only that [HH86] also use a synchronised displacement condition, thus strengthening our confidence in its naturalness, if not inevitability.

3.2.6 The displaced environment theorem

Here we use parameterised bisimulation, [Lar86], a little less trivially, but still very simply. Readers who wish to can skip this section without loss of continuity.

First we finish our elementary introduction to Larsen's theory by pointing out how grossly we have simplified it. The parameterised bisimulation of [Lar86] is far more powerful than our simplified version above. The subscript is in general an *environment* (we simplify immediately again, and think of it as just a CCS agent), and the set of actions it can immediately perform is our acceptable set above. But on the next step, the set of acceptable actions might change: to show $P \sim_O Q$ we have to show for each a such that $O \xrightarrow{a} O'$ and $P \xrightarrow{a} P'$ that we have a Q' such that $Q \xrightarrow{a} Q'$ and $P' \sim_{O'} Q'$, and also of course that P can match Q. If $\mathbf{O}$ is the family of O and all its derivatives, we need an $\mathbf{O}$-indexed family of bisimulations, not just one parameterised bisimulation.

DEFINITION 3.2.11. *Let* $\mathbf{O}$ *be a family of agents closed under* $\rightarrow$. *Then an* $\mathbf{O}$*-parameterised bisimulation,* $\mathcal{R}$*, is an* $\mathbf{O}$*-indexed family of binary relations,* $\mathcal{R}_O \subseteq \mathbf{P} \times \mathbf{P}$ *for* $O \in \mathbf{O}$ *such that whenever* $P\mathcal{R}_O Q$ *the following holds:*
$\forall a \in Act$ *if* $O \xrightarrow{a} O'$, *then*
(i) if $P \xrightarrow{a} P'$ *then, for some* Q', $Q \xrightarrow{a} Q'$ *and* $P'\mathcal{R}_{O'}Q'$,
(ii) if $Q \xrightarrow{a} Q'$ *then, for some* P', $P \xrightarrow{a} P'$ *and* $P'\mathcal{R}_{O'}Q'$
P and Q are said to be equivalent in an environment $O \in \mathbf{O}$ *iff* $\exists$ *an* $\mathbf{O}$*-parameterised bisimulation* $\mathcal{R}$ *such that* $P\mathcal{R}_O Q$. *Then we write* $P \sim_O Q$.
□

$\sim_O$ is an equivalence relation for all O. Any two agents P and Q are related by $P \sim_{\mathbf{0}} Q$, because the environment $\mathbf{0}$ does not care about any actions at all, and the relation $\sim_U$, where U is the universal agent that can do every action at every stage, is the same as $\sim$, because now the environment cares about all actions. The more possibilities the environment has, the stronger the parameterised bisimulation. $P \sim Q$ implies $P \sim_O Q$ for all O.

Thus our special case can be thought of as dealing with environments of the form $O \Leftarrow \sum_{a \in \mathcal{A}} a.O$ where the sum is over the set of acceptable actions. Thus the environment does not change as the matching agents evolve, and our family of environment agents is a singleton set. The form of our environment agents shows that they can each be characterised completely by their sets of acceptable actions, which justifies the notation we adopted briefly a little while ago.

We will only use strong parameterised bisimulation. For completeness, we mention that we have merely touched on a small area of [Lar86]. Among the many topics he deals with, are, for example, extension of the theory to weak bisimulation, a general study of contexts as environment transformers, and modal properties of parameterised bisimulation and of contexts.

We have already seen some examples of parameterised bisimulations. The semaphore example gives us another instance, this time not characterisable by a set of actions.

$P \Leftarrow pa.vb.P$ *the perfect user*
$S \Leftarrow p^{-1}.v^{-1}.S$ *the semaphore*

Here $P \mid P \not\sim P$, but $(P \mid P \mid S) \backslash\backslash \{p, v\} \sim (P \mid S) \backslash\backslash \{p, v\}$, and of course $P \sim_{\mathbf{0}} P \mid P$. What other environments equate the two?

One suitable environment is P itself: $P \mid P \sim_P P$. But this is silly; if we had a more detailed specification of P, we would certainly not want to restrict consideration to such a limited environment. We expect that the semaphore

does not care about actions other than p or v; for these, it enforces its discipline. The following environment meets this intuition:

$N \Leftarrow \Sigma_r\, r.N + \Sigma_x\, x.N'$

$N' \Leftarrow \Sigma_r\, r.N' + \Sigma_y\, y.N$

The sum over r is over all actions that have no p or v particles, x is over all actions with a single p particle, and y is over all actions with a single v particle. As the reader can check, we have $P \mid P \sim_N P$. This parameterised bisimulation is in fact the strongest we can get, and is what is predicted for the context $([\,] \mid S) \backslash\backslash \{p, v\}$ by Larsen's theory.

Now we have already proved that two faulty users Q are congruent to one in the context of a faulty semaphore T if the failures are synchronised, though we did not at that stage separate out the semaphore: we proved that $(Q \mid T) \sim (Q \mid Q \mid T)$ in the context of suitable demons.

We would like to do the proof over again. We ask if we can prove $Q \mid Q \sim Q$ in the context of T (and the demons). If we consider the demon context as outermost, enclosing the semaphore context, which encloses the faulty users, we see that $Q \mid Q$ and Q are in a context that has normal and exception states, and that a "displace" in the context (i.e. transition to exception state) is synchronised with the displace actions of the users. We also know that $P \mid P \sim P$ in the environment N of a normal semaphore. Can we complete the proof by dealing with just the exception agents in their exceptional context? Yes, because of the following theorem. Note that because we haven't said formally how a context induces environments, this theorem should be taken as an outline of how such an argument might be made formally.

THEOREM 3.2.12. *Let*

$P \Leftarrow P_N \dagger P_E$

$Q \Leftarrow Q_N \dagger Q_E$

Let $C[\,]$ be a context that synchronises the displace actions of P and Q, and induces the environment $O \Leftarrow N \dagger E$, where E accepts only the displace actions of either P or Q, and N accepts no displace actions. Suppose $P_N \sim_N Q_N$ and $P_E \sim_E Q_E$. Then $C[P] \sim_O C[Q]$.

PROOF. The parameterised bisimulation is as follows: For every agent O of the form $N \dagger E$, $\mathcal{R}_O$ consists of all pairs $\langle P, Q \rangle$ where P is of the form $P_N \dagger P_E$, Q is of the form $Q_N \dagger Q_E$, and $P_N \sim_N Q_N$ and $P_E \sim_E Q_E$. Add in the pairs $\sim_O$ for all $O \in \mathcal{O}$, where $\mathcal{O}$ is the family of all derivatives of O.

We outline the proof.

Start from $P \sim_O Q$. If O does a normal action, $N \xrightarrow{a} N'$, P can only do a normal action, because all displaces are synchronised. If $P_N \xrightarrow{a} P'_N$, we have because $P_N \sim_N Q_N$, $Q_N \xrightarrow{a} Q'_N$, such that $P'_N \sim_{N'} Q'_N$. We reach the pair $\langle P', Q' \rangle$ in $\mathcal{R}_{O'}$, where P' is $P'_N \dagger P_E$, Q' is $Q'_N \dagger Q_E$ and O' is $N' \dagger E$.

If O does a displace action, so does P, and Q can match this because $P_E \sim_E Q_E$. We reach the pair $\langle P'_E, Q'_E \rangle$, which is in $\sim_{E'}$. □

Here are the definitions again for the semaphore example. We've included a new faulty "double" user Q_2 that needs an f^2 to fail.

$P \Leftarrow pa.vb.P$	*the perfect user*
$S \Leftarrow p^{-1}.v^{-1}.S$	*the semaphore*
$Q \Leftarrow P \dagger f.Q$	*a faulty user*
$Q_2 \Leftarrow P \dagger f^2.Q$	*the faulty "double" user*
$T \Leftarrow S \dagger g.T$	*the faulty semaphore*
$F \Leftarrow f^{-1}g^{-1}.F$	*the demon*
$F_2 \Leftarrow f^{-2}g^{-1}.F_2$	*the double demon*

to prove prove $Q \mid Q$ and Q_2 congruent in the context of T and F_2.

We proved earlier that a two-user fault tolerant system is congruent to a single-user one. We had $(Q \mid Q \mid T \mid F_2) \backslash\backslash \{p, v, f, g\} \sim (Q \mid T \mid F) \backslash\backslash \{p, v, f, g\}$. We can show easily that $(Q_2 \mid T \mid F_2) \backslash\backslash \{p, v, f, g\}$ is also equal to these two.

Now we want to do the proof over again, showing that $Q \mid Q \sim_O Q_2$ where O is the environment corresponding to the context $([\,] \mid T \mid F_2) \backslash\backslash \{p, v, f, g\}$. Call this context $\mathcal{C}$. To be fussy, $\mathcal{C}$ is really the context of the semaphore enclosed in a context about which we only know that it synchronises the displace actions of the semaphore and the users.

We outline the proof informally. The environment corresponding to $\mathcal{C}$ is $O \Leftarrow N \dagger \sum_e e.O$, where N is the environment of the perfect semaphore. The second term (call it E) is summed over all actions e that have precisely two f particles in them. The demon synchronises the displace action of the semaphore with any such action. The context then looks exactly as it did at the start.

We have $P \mid P \sim_N P$. We have to show $f.Q \mid f.Q \sim_E f^2.Q_2$. The left hand side is forced to synchronise (E is not interested in actions with just one f), and we finish up in the same pair of the bisimulation as we started.

Now apply the above theorem.

3.3 The Audit operator

We want to generalise the $\dagger$ operator so that the exception agent can keep track of the progress of the normal one, and thus restart it at different states. In our semaphore example, if the errant user could be restarted inside the critical region or outside depending on where the failure occurred, we might get observational equivalence between the fault-free and faulty systems.

With the displace operator, the exception agent cannot change state to keep track of normal processing, because any action by it immediately aborts the normal agent. So the new operator must allow the exception agent to perform actions that are not displacements. There seems little reason to allow the exception agent to change state by itself, except for displace actions as usual. Thus the new actions by the exception agent will always be in synchronisation with the normal agent. We could even allow communication between the normal and exception agents; this is in fact what happens with the checkpoint operator we define later. But it is quite complicated and is unnecessary for our immediate goals. The exception agent needs only to eavesdrop on some actions, the *significant* ones that require a change of state on restart.

For example, most modern text editors have significant commands that change the file being edited, such as insert and delete, and several insignificant commands that do not, such as scrolling, viewing, helping, etc. The exception handler should now be thought of as an audit trail mechanism ("journal file" and "log file" are other commonly used terms) that records significant commands. It does not save the state of the edited file; should failure occur, recovery consists of re-running all the commands in the journal file.

We propose a variant of the displace operator to program such applications. Insignificant actions of the application do not need to be recorded in the journal file, so the exception agent ignores them; i.e. they become normal actions as seen by the operator.

Significant actions, or *audited actions* as we shall call them, are handled by the *audit* rule for the new operator. The simplest kind of auditing would be for both agents to do the entire action. This would be like the parallel operator in CSP, or like various other synchronisation and conjunction operators in LOTOS and SCCS. This is a little too restrictive for most purposes. In text editors, deleting a line usually causes the display to change as well. One action is audited and the other normal, but it is natural to synchronise the two, to prevent others from intervening.

The exception agent acts alone for displacements, precisely as with the dis-

place operator. The resulting state will not in general be that it would have been had a displace occurred earlier; more commands would have gone into the log file.

DEFINITION 3.3.1. *The operational semantics of the audit operator.* $\Cup\!\!\!\!+_{\mathcal{C}}$ *is a binary operator* $\mathbf{P}^M \times \mathbf{P}^M \to \mathbf{P}^M$ *that is parameterised by a set* $\mathcal{C}$ *of particles, called the audit set.*

$$\textit{Normal} \quad \frac{A \xrightarrow{a} A' \quad a \notin \mathcal{C}}{A \Cup\!\!\!\!+_{\mathcal{C}} B \xrightarrow{a} A' \Cup\!\!\!\!+_{\mathcal{C}} B}$$

$$\textit{Displace} \quad \frac{B \xrightarrow{r} B' \quad r \notin \mathcal{C}}{A \Cup\!\!\!\!+_{\mathcal{C}} B \xrightarrow{r} B'}$$

$$\textit{Audit} \quad \frac{A \xrightarrow{a} A' \quad B \xrightarrow{b} B' \quad a\mathcal{C}b \quad b \in \mathcal{C}}{A \Cup\!\!\!\!+_{\mathcal{C}} B \xrightarrow{a} A' \Cup\!\!\!\!+_{\mathcal{C}} B'}$$

□

The normal and displace rules differ from those for the displace operator. They now insist that there should be no particles from the audit set $\mathcal{C}$. The reason for this will become clear shortly.

In the audit rule, the resulting action is what the normal agent does, thus giving an external appearance of normalcy. This action has at least one particle from the audit set, and possibly some not in it. The exception agent keeps track by doing an action consisting of all the particles from the audit set. These last two statements are what the side conditions express, as we shall now see.

To define the predicate $a\mathcal{C}b$, we begin by defining the auxiliary relation *equality over* $\mathcal{C}$. Consider two actions a and b expressed as products of non-opposing particles. Informally, we would like to say that they are equal over $\mathcal{C}$ if the parts of the two products that use particles from $\mathcal{C}$ are equal. Thus if we define these parts of the products as $a_{\mathcal{C}}$ and $b_{\mathcal{C}}$, where $_{\mathcal{C}}$ is a unary operator called *projection on* $\mathcal{C}$, we can define equality over $\mathcal{C}$ conveniently as $a_{\mathcal{C}} = b_{\mathcal{C}}$, rather than define a series of parameterised equality relations.

DEFINITION 3.3.2. *Let* $\mathcal{C}$ *be a set of particles, and* $a = \lambda_1^{z_1} \ldots \lambda_n^{z_n}$ *be an action where the* λ_i *are all distinct, and the* z_i *are all non-zero integers. Suppose that* $\lambda_1 \ldots \lambda_i \in \mathcal{C}$ *and that* $\lambda_{i+1} \ldots \lambda_n \notin \mathcal{C}$. *Then the projection of* a *onto* $\mathcal{C}$, $a_{\mathcal{C}}$, *is defined to be* $\lambda_1^{z_1} \ldots \lambda_i^{z_i}$. □

Thus the projection of a on $\mathcal{C}$ is the product of all particles of a that are in $\mathcal{C}$. Our use of the term "projection" is roughly the same as in geometry or linear algebra, but the analogy is incomplete and dangerous because our sorts

are based on submonoids instead of subgroups. Thus the projection of a^{-1} on Part(a) is 1, not a^{-1}, which would have been the result with sorts based on subgroups. For another example, note that $(ab)_C = a_C b_C$ is *false*: if a is particulate, $b = a^{-1}$, and $C = \{a\}$, the left hand side will be 1 whereas the right hand side will be a.

DEFINITION 3.3.3. *Two actions a and b are said to be equal over a set of particles C if $a_C = b_C$. If further, these projections are not 1, we say a and b are checkpoint equal over C and write aCb. That is, we denote "$a_C = b_C$ and $a \not\leqslant C$" by aCb.* □

Checkpoint equality is the sole side condition of the checkpoint rule for the checkpoint operator. We have chosen to separate it out, and have two side conditions for the audit rule to facilitate comparison.

The first thing to notice is that the checkpoint equality condition insists that every significant particle in the normal agent's action is to be tracked by the exception agent. The side condition in the displace rule enables us to look at any action of the exception agent and say unambiguously whether it is a displace or part of an audit. The same reason motivates the side condition in the normal rule—all actions of the normal agent are separated unambiguously into either significant or insignificant. We now see that the side condition on the normal rule prevents any leaking away of significance.

We decide that 1 cannot be a significant action. In the editor application, autonomous actions by the editor (garbage collection, for example) should not alter the file! This explains why the audit rule insists that there should be a visible particle from the audit set. It follows that the exception agent cannot do a 1 as its contribution to an audit action. These decisions also mesh well with the SDT; see the discussion following the theorem. That the visibility condition and equality of projection are presented together as a single predicate, checkpoint equality, is just an abbreviation.

Since we are keeping audits and other actions mutually exclusive for both normal and exception agents, and since we don't want to limit ourselves to rigid agents, 1 is allowed as a normal or displace action, as with the displace operator. $1 \not\leqslant C$ for all C.

PROPOSITION 3.3.4. $A \dagger_{\emptyset} B \sim A \dagger B$

PROOF. All actions of A are acceptable as normal actions, because any a vacuously satisfies the condition $a \not\leqslant \emptyset$. Similarly, any action of B can be a displace. There are no audit particles, so the audit rule never applies. □

Obviously, we have to program to avoid having actions of A or B that are not allowed to contribute via any of the three rules. Similarly, we could put in as many particles into $\mathcal{C}$ as we like, but including particles that are not in $\mathcal{B}$ will not affect either displace or audit actions; it will only restrict some normal actions. Similarly including particles not in $\mathcal{A}$ will merely restrict B's displacements. A more sensible way to program these cases is to use the restriction operator! It pays to keep $\mathcal{C}$ as small as possible, because it will then be easier to check that the conditions for the SDT apply.

So far the discussion applies equally to the checkpoint operator. The first consequence of the second side condition is that the resulting action is just what the normal agent did. Matters are considerably messier with the checkpoint operator, and we have sometimes to prove action equality in bisimulation proofs.

Another consequence of the second side condition is that the normal and exception agents cannot communicate. With the checkpoint operator, the normal agent can actually pass values to the exception agent, for example, some indication of its current state. In the audit rule, the exception agent can only keep track by stringing along and making its own estimates as to where the normal agent would have got to. With deterministic agents, we can program this very well, because the exception agent knows all the significant actions that have happened.

A third consequence, that we shall sometimes use in proofs, is that a in that rule can be written in the form bx, where $x \not\preceq \mathcal{C}$. Thus the two conditions together are equivalent to "a can be written as a product of non-opposed actions b and x, where $x \not\preceq \mathcal{C}$, $b \in \mathcal{C}$, and $b \neq 1$." (Remember that $b \in \mathcal{C}$ means that all the particles of b are in $\mathcal{C}$.)

In the next two sections, we briefly examine substitutive and equational properties of $\dagger_{\mathcal{C}}$ before studying synchronised agents. Desirable properties which fail to hold for $\dagger$ of course also fail for the more general $\dagger_{\mathcal{C}}$, so we refer the reader back to Section 3.1 for these.

We shall often say "audit operator", or "$\dagger_{\mathcal{C}}$", to mean the family $\dagger_{\mathcal{C}}$ indexed by $\mathcal{C}$. This informal usage parallels that of "action prefix". In this sense, the audit operator is a generalisation of displace, because one instance of the family, $A \dagger_{\emptyset} B$, is the same as $\dagger$.

3.3.1 Substitutive properties of the Audit operator

PROPOSITION 3.3.5. *$P \sim Q \Longrightarrow P \dagger_{\mathcal{C}} R \sim Q \dagger_{\mathcal{C}} R$ and $R \dagger_{\mathcal{C}} P \sim R \dagger_{\mathcal{C}} Q$.* □

As for $\dagger$, $\approx$ is substitutive on the left but not on the right of $\dagger_{\mathcal{C}}$.

PROPOSITION 3.3.6. $P \approx Q \Longrightarrow P \between_{\mathcal{C}} R \approx Q \between_{\mathcal{C}} R$.

PROOF. By bisimulation. Remember that 1 is acceptable as a normal action. For the audit case, if $P \xrightarrow{a} P'$, there are Q_1, Q_2, Q' such that $Q \xrightarrow{1^k} Q_1 \xrightarrow{a} Q_2 \xrightarrow{1^l} Q'$ and $P' \approx Q'$. To match $\dfrac{P \xrightarrow{a} P' \quad R \xrightarrow{b} R' \quad aCb \quad b \in \mathcal{C}}{P \between_{\mathcal{C}} R \xrightarrow{a} P' \between_{\mathcal{C}} R'}$, the rhs does $\dfrac{Q \xrightarrow{1^k} Q_1 \quad 1^k \notin \mathcal{C}}{Q \between_{\mathcal{C}} R \xrightarrow{1^k} Q_1 \between_{\mathcal{C}} R}$, then $\dfrac{Q_1 \xrightarrow{a} Q_2 \quad R \xrightarrow{b} R' \quad aCb \quad b \in \mathcal{C}}{Q_1 \between_{\mathcal{C}} R \xrightarrow{a} Q_2 \between_{\mathcal{C}} R'}$ and $\dfrac{Q_2 \xrightarrow{1^l} Q' \quad 1^l \notin \mathcal{C}}{Q_2 \between_{\mathcal{C}} R' \xrightarrow{1^l} Q' \between_{\mathcal{C}} R'}$. □

$\between_{\mathcal{C}}$ does not preserve even $\approx^{cM}$ on the right. If $Q \approx^{cM} S$, consider $P \between_{\mathcal{C}} Q$ and $P \between_{\mathcal{C}} S$. To match an audit action of Q's, S might begin by doing 1's. But these will be displacements, and will abort P! For the same reason, $\between_{\mathcal{C}}$ fails to preserve $\approx^{cA}$. As we shall see in Chapter 5, this means that $\between_{\mathcal{C}}$ is not derivable from either MCCS or ASCCS.

3.3.2 Equational properties of audit

Restriction distributes unconditionally over $\between_{\mathcal{C}}$ because the normal and exception agent do not communicate; i.e. the actions they do as their contributions to an audit action (the only time they act together), have no opposing particles.

PROPOSITION 3.3.7. $(P \between_{\mathcal{C}} Q) \backslash \mathcal{N} \sim (P \backslash \mathcal{N}) \between_{\mathcal{C}} (Q \backslash \mathcal{N})$

PROOF. By bisimulation. Normal and displace actions are as for $\between$. For audit, if $P \xrightarrow{p} P'$, both sides need $p \notin \mathcal{N}$. □

Morphism distributes over $\between_{\mathcal{C}}$ if it is a *relabelling*, i.e., if it is an isomorphism. We use the notation $\phi(\mathcal{C})$ to mean $\{b \mid b = \phi a, a \in \mathcal{C}\}$.

LEMMA 3.3.8. $a \lessdot \mathcal{C} \Longleftrightarrow \phi(a) \lessdot \phi(\mathcal{C})$ *and* $aCb \Longleftrightarrow \phi(a)\,\phi(\mathcal{C})\,\phi(b)$. □

PROPOSITION 3.3.9. *If* ϕ *is an isomorphism,* $(P \between_{\mathcal{C}} Q)[\phi] \sim (P[\phi]) \between_{\phi(\mathcal{C})} (Q[\phi])$. □

$\between_{\mathcal{C}}$ is obviously not commutative, but is it associative? We shall see that $(P \between_{\mathcal{C}} Q) \between_{\mathcal{C}} R \sim P \between_{\mathcal{C}} (Q \between_{\mathcal{C}} R)$ holds, but that $(P \between_{\mathcal{C}} Q) \between_{\mathcal{D}} R \sim P \between_{\mathcal{C}} (Q \between_{\mathcal{D}} R)$ is not true in general. It is instructive to investigate the conditions under which it is, for we use the same kind of argument as we need for the SDT, and

motivate several lemmas about particles and checkpoint equality that we need for that proof.

We quote the one we need for the proof below:

Particles 3. If $\mathcal{P} \cap \mathcal{D} \subseteq \mathcal{C}$ and $P \xrightarrow{p}$ and $p \notin \mathcal{C}$ then $p \notin \mathcal{D}$.

Remember that we use $\mathcal{P}$ for the sort of an agent P.

PROPOSITION 3.3.10. *If $\mathcal{P} \cap \mathcal{D} \subseteq \mathcal{C}$ and $\mathcal{R} \cap \mathcal{C} \subseteq \mathcal{D}$ then $(P \upharpoonleft_{\mathcal{C}} Q) \upharpoonleft_{\mathcal{D}} R \sim P \upharpoonleft_{\mathcal{C}} (Q \upharpoonleft_{\mathcal{D}} R)$.*

PROOF. The condition is needed because if $\mathcal{D}$ has particles of $\mathcal{P}$ not in $\mathcal{C}$, i.e., the outer agent audits particles not audited by the inner, the lhs will stop normal actions of P permitted on the right. If $\mathcal{C}$ has particles of $\mathcal{R}$ not in $\mathcal{D}$, i.e., the inner agent protects itself against displace actions permitted by the outer, R can do some displace actions on the lhs not permitted on the rhs.

The first condition is needed for audit actions by P and Q. If $\mathcal{D}$ is too big, the lhs might stop an audit by $P \upharpoonleft_{\mathcal{C}} Q$ because of the non-$\mathcal{C}$ particles in P's action; Q's contribution has to pass through $\mathcal{D}$ on both sides, so is no problem.

No conditions are needed for audit actions by Q and R; this is because the result looks like a normal action by Q, which has to pass through both $\mathcal{C}$ and $\mathcal{D}$ on both sides.

More subtly, the conditions also ensure that P and R cannot do an audit together, without a contribution from Q. If this were allowed, P's contribution on the lhs would be a normal action to $P \upharpoonleft_{\mathcal{C}} Q$, whereas R's contribution on the rhs would be a displace to $Q \upharpoonleft_{\mathcal{D}} R$, wiping out Q. The condition prevent this disaster by ensuring that a normal action by P (filtered through $\mathcal{C}$) can have no particles in $\mathcal{D}$, and similarly for R filtered through $\mathcal{D}$.

For both sides, an inference can be labelled unambiguously by saying which of P, Q or R contributes. For example, PR can only mean a normal action by $P \upharpoonleft_{\mathcal{C}} Q$ contributing to an audit action by $(P \upharpoonleft_{\mathcal{C}} Q) \upharpoonleft_{\mathcal{D}} R$, and can only be matched by $P \upharpoonleft_{\mathcal{C}} (Q \upharpoonleft_{\mathcal{D}} R)$ by its PR inference—any other inference would use premises not used on the lhs. The table below lists all the cases, giving for each the additional assumptions on side-conditions needed by the lhs and rhs. The reader will find it easier to follow the table in conjunction with the bisimulation, also given below.

An arrow $\rightarrow$ labels the case where we consider how the rhs might match the lhs; $\leftarrow$ the other way.

$P \rightarrow$ Unconditionally.

$\leftarrow$ Need $P \xrightarrow{p} P'$ and $p \notin \mathcal{C} \Longrightarrow p \notin \mathcal{D}$. See Particles 3.

Q $\leftrightarrow$ Unconditionally.

R $\rightarrow$ Need $R \xrightarrow{r} R'$ and $r \notin \mathcal{D} \Longrightarrow r \notin \mathcal{C}$. See Particles 3.

$\leftarrow$ Matches the rhs unconditionally.

PQ $\rightarrow$ Need $q \notin \mathcal{D}$. Every particle in q is in p, which has no particles in $\mathcal{D}$.

$\leftarrow$ Need $p \notin \mathcal{D}$. p can be written qx where $q \notin \mathcal{D}$ and $x \notin \mathcal{C}$. By Particles 3, we have $x \notin \mathcal{D}$.

QR $\leftrightarrow$ Unconditionally.

PR $\rightarrow$ The lhs inference is $\dfrac{\dfrac{P \xrightarrow{p} P' \quad p \notin \mathcal{C}}{P \mathbin{\natural_{\mathcal{C}}} Q \xrightarrow{p} P' \mathbin{\natural_{\mathcal{C}}} Q} \quad R \xrightarrow{r} R' \quad p\mathcal{D}r \quad r \in \mathcal{D}}{(P \mathbin{\natural_{\mathcal{C}}} Q) \mathbin{\natural_{\mathcal{D}}} R \xrightarrow{p} (P' \mathbin{\natural_{\mathcal{C}}} Q) \mathbin{\natural_{\mathcal{D}}} R'}$. If Q is not to act, any inference on the rhs will lose it. Fortunately, the PR inference on the lhs is impossible because by Particles 3, $p \notin \mathcal{C}$ and $P \xrightarrow{p} P'$ imply $p \notin \mathcal{D}$. Then we cannot have $p\mathcal{D}r$.

$\leftarrow$ The rhs inference is $\dfrac{P \xrightarrow{p} P' \quad \dfrac{R \xrightarrow{r} R' \quad r \notin \mathcal{D}}{Q \mathbin{\natural_{\mathcal{D}}} R \xrightarrow{r} R'} \quad p\mathcal{C}r \quad r \in \mathcal{C}}{P \mathbin{\natural_{\mathcal{C}}} (Q \mathbin{\natural_{\mathcal{D}}} R) \xrightarrow{p} P' \mathbin{\natural_{\mathcal{C}}} R'}$. No inference on the lhs can get rid of Q alone. Fortunately, the PR inference on the rhs is impossible because by Particles 3, $r \notin \mathcal{D}$ and $R \xrightarrow{r} R'$ imply $r \notin \mathcal{C}$. Then we cannot have $p\mathcal{C}r$.

PQR $\rightarrow$ Need $q\mathcal{D}r$. We have $p\mathcal{C}q$ and $p\mathcal{D}r$. Every particle of P that is in $\mathcal{D}$ is in $\mathcal{C}$, by Particles 3. Every particle of q is in p.

$\leftarrow$ Need $p\mathcal{D}r$ from $p\mathcal{C}q$ and $q\mathcal{D}r$. We can write p as qx and q as ry, where $x \notin \mathcal{C}$ and $y \notin \mathcal{D}$. By Particles 3, $x \notin \mathcal{D}$.

The bisimulation below lists all but the case ruled out.

$$\mathcal{N}(P,Q,R) \qquad \langle\, (P \mathbin{\natural_{\mathcal{C}}} Q) \mathbin{\natural_{\mathcal{D}}} R, \quad P \mathbin{\natural_{\mathcal{C}}} (Q \mathbin{\natural_{\mathcal{D}}} R) \,\rangle$$

Case P:

$$\frac{\dfrac{P \xrightarrow{p} P' \quad p \notin \mathcal{C}}{P \mathbin{\natural_{\mathcal{C}}} Q \xrightarrow{p} P' \mathbin{\natural_{\mathcal{C}}} Q} \quad p \notin \mathcal{D}}{(P \mathbin{\natural_{\mathcal{C}}} Q) \mathbin{\natural_{\mathcal{D}}} R \xrightarrow{p} (P' \mathbin{\natural_{\mathcal{C}}} Q) \mathbin{\natural_{\mathcal{D}}} R} \qquad \frac{P \xrightarrow{p} P' \quad p \notin \mathcal{C}}{P \mathbin{\natural_{\mathcal{C}}} (Q \mathbin{\natural_{\mathcal{D}}} R) \xrightarrow{p} P' \mathbin{\natural_{\mathcal{C}}} (Q \mathbin{\natural_{\mathcal{D}}} R)}$$

leading to $\mathcal{N}(P',Q,R)$

Case Q:

$$\frac{Q \xrightarrow{q} Q' \quad q \notin \mathcal{C}}{\dfrac{P \between_{\mathcal{C}} Q \xrightarrow{q} Q' \quad q \notin \mathcal{D}}{(P \between_{\mathcal{C}} Q) \between_{\mathcal{D}} R \xrightarrow{q} Q' \between_{\mathcal{D}} R}} \qquad \frac{Q \xrightarrow{q} Q' \quad q \notin \mathcal{D}}{\dfrac{Q \between_{\mathcal{D}} R \xrightarrow{q} Q' \between_{\mathcal{D}} R \quad q \notin \mathcal{C}}{P \between_{\mathcal{C}} (Q \between_{\mathcal{D}} R) \xrightarrow{q} Q' \between_{\mathcal{D}} R}}$$

leading to $\mathcal{I}$

Case R:

$$\frac{R \xrightarrow{r} R' \quad r \notin \mathcal{D}}{(P \between_{\mathcal{C}} Q) \between_{\mathcal{D}} R \xrightarrow{r} R'} \qquad \frac{\dfrac{R \xrightarrow{r} R' \quad r \notin \mathcal{D}}{Q \between_{\mathcal{D}} R \xrightarrow{r} R' \quad r \notin \mathcal{C}}}{P \between_{\mathcal{C}} (Q \between_{\mathcal{D}} R) \xrightarrow{r} R'}$$

leading to $\mathcal{I}$

Case PQ:

$$\frac{P \xrightarrow{p} P' \quad Q \xrightarrow{q} Q' \quad p \mathcal{C} q \quad q \in \mathcal{C}}{\dfrac{P \between_{\mathcal{C}} Q \xrightarrow{p} P' \between_{\mathcal{C}} Q' \qquad p \notin \mathcal{D}}{(P \between_{\mathcal{C}} Q) \between_{\mathcal{D}} R \xrightarrow{p} (P' \between_{\mathcal{C}} Q') \between_{\mathcal{D}} R}}$$

$$\frac{P \xrightarrow{p} P' \quad \dfrac{Q \xrightarrow{q} Q' \quad q \notin \mathcal{D}}{Q \between_{\mathcal{D}} R \xrightarrow{q} Q' \between_{\mathcal{D}} R} \quad p \mathcal{C} q \quad q \in \mathcal{C}}{P \between_{\mathcal{C}} (Q \between_{\mathcal{D}} R) \xrightarrow{p} P' \between_{\mathcal{C}} (Q' \between_{\mathcal{D}} R)}$$

leading to $\mathcal{N}(P', Q', R)$

Case QR:

$$\dfrac{\dfrac{Q \xrightarrow{q} Q' \quad q \notin \mathcal{C}}{P \between_{\mathcal{C}} Q \xrightarrow{q} Q'} \qquad R \xrightarrow{r} R' \quad q \mathcal{D} r \quad r \in \mathcal{D}}{(P \between_{\mathcal{C}} Q) \between_{\mathcal{D}} R \xrightarrow{q} Q' \between_{\mathcal{D}} R'}$$

$$\dfrac{Q \xrightarrow{q} Q' \quad R \xrightarrow{r} R' \quad q \mathcal{D} r \quad r \in \mathcal{D}}{Q \between_{\mathcal{D}} R \xrightarrow{q} Q' \between_{\mathcal{D}} R' \qquad q \notin \mathcal{C}}$$

$$P \between_{\mathcal{C}} (Q \between_{\mathcal{D}} R) \xrightarrow{q} Q' \between_{\mathcal{D}} R'$$

leading to $\mathcal{I}$

Case PQR:

$$\dfrac{\dfrac{P \xrightarrow{p} P' \quad Q \xrightarrow{q} Q' \quad p \mathcal{C} q \quad q \in \mathcal{C}}{P \between_{\mathcal{C}} Q \xrightarrow{p} P' \between_{\mathcal{C}} Q'} \qquad R \xrightarrow{r} R' \quad p \mathcal{D} r \quad r \in \mathcal{D}}{(P \between_{\mathcal{D}} Q) \between_{\mathcal{D}} R \xrightarrow{p} (P' \between_{\mathcal{D}} Q') \between_{\mathcal{D}} R'}$$

$$\dfrac{P \xrightarrow{p} P' \qquad \dfrac{Q \xrightarrow{q} Q' \quad R \xrightarrow{r} R' \quad q \mathcal{D} r \quad r \in \mathcal{D}}{Q \between_{\mathcal{D}} R \xrightarrow{q} Q' \between_{\mathcal{D}} R'} \quad p \mathcal{C} q \quad q \in \mathcal{C}}{P \between_{\mathcal{C}} (Q \between_{\mathcal{C}} R) \xrightarrow{p} P' \between_{\mathcal{C}} (Q' \between_{\mathcal{C}} R')}$$

leading to $\mathcal{N}(P', Q', R')$

□

COROLLARY 3.3.11. $(P \between_{\mathcal{C}} Q) \between_{\mathcal{C}} R \sim P \between_{\mathcal{C}} (Q \between_{\mathcal{C}} R)$. □

Thus we can write $P \between_{\mathcal{C}} Q \between_{\mathcal{C}} R$ without ambiguity. $(\mathbf{P}/\sim, \between_{\mathcal{C}}, \mathbf{0})$ is not a monoid, however, because $\mathbf{0}$ is an identity for $\between_{\mathcal{C}}$ only in the following practically useless case.

PROPOSITION 3.3.12. $P \between_{\mathcal{C}} \mathbf{0} \sim P \iff \mathcal{P} \cap \mathcal{C} = \emptyset \iff \mathbf{0} \between_{\mathcal{C}} P \sim P$. □

As for $\between$, most absorptive and distributive laws do not hold for $\between_{\mathcal{C}}$, but the following exception for $\between$ survives.

PROPOSITION 3.3.13. $(Q + R) \between_{\mathcal{C}} P \sim (Q \between_{\mathcal{C}} P) + (R \between_{\mathcal{C}} P)$

PROOF. By bisimulation. Normal and displace actions as for $\between$. Audit actions can occur in two ways, with either Q or R contributing, but it still

doesn't matter whether the choice between the two is made within the normal component of one agent, or between two agents. □

But we no longer have the following: $(A \lhd\!\!\rhd_{\mathcal{C}} B) + B \sim A \lhd\!\!\rhd_{\mathcal{C}} B$, for now the behaviour of B as an exception agent is curbed by $\mathcal{C}$. A more sensible viewpoint is that B in the audited agent is programmed to update journal files, say, but outside such a context this becomes an independent activity.

3.3.3 Particles and checkpoint equality

We now come to some lemmas used mostly in the proofs of the two SDTs to follow, as well as in the proofs of associativity for $\lhd\!\!\rhd_{\mathcal{C}}$ and $\lhd\!\!\rhd_{\mathcal{C}}$, grouped together under headings "Particles", "Checkpoint Equality", etc. They are in the general setting of actions a, b, p, q and sets of particles $\mathcal{C}$ and $\mathcal{R}$ such that $a, a^{-1}, b, b^{-1} \notin \mathcal{R}$ and $p, p^{-1}, q, q^{-1} \notin \mathcal{C}$, and the subset of these required for each proposition are written out in the heading.

The proofs are simple but tedious and we have to be on guard against overlooking cases. We give outlines of only the less obvious cases.

LEMMA 3.3.14. ***Particles***
Suppose $a, a^{-1} \notin \mathcal{R}$ *and* $p, p^{-1} \notin \mathcal{C}$. *Let* $\mathcal{U} = \mathcal{C} \cup \mathcal{R}$. *Then*

1. $a \notin \mathcal{C} \iff a \notin \mathcal{U}$
2. $a \notin \mathcal{C} \wedge p \notin \mathcal{R} \iff ap \notin \mathcal{U}$
3. *If* $\mathcal{P} \cap \mathcal{D} \subseteq \mathcal{C}$ *and* $P \xrightarrow{p}$ *and* $p \notin \mathcal{C}$ *then* $p \notin \mathcal{D}$.

PROOF. 1 is a special case of 2; put $p = 1$.
2.($\Longrightarrow$) We have $\mathrm{Part}(ap) \subset \mathrm{Part}(a) \cup \mathrm{Part}(p)$ and the conditions assure us that neither $\mathrm{Part}(a)$ nor $\mathrm{Part}(b)$ has any intersection with either $\mathcal{C}$ or $\mathcal{R}$.
($\Longleftarrow$) $ap \notin \mathcal{C}$ is immediate; from this we get the result because $p^{-1} \notin \mathcal{C}$ means p cannot cancel out particles in $\mathcal{C}$.
3. Immediate. □

LEMMA 3.3.15. ***Projections***
Suppose $a, a^{-1} \notin \mathcal{R}$ *and* $p, p^{-1} \notin \mathcal{C}$. *Let* $\mathcal{U} = \mathcal{C} \cup \mathcal{R}$. *Then*

1. $a_{\mathcal{C}} = a_{\mathcal{U}}$
2. $a_{\mathcal{C}} = (ap)_{\mathcal{C}}$
3. $p \notin \mathcal{R} \iff a_{\mathcal{C}} = (ap)_{\mathcal{U}}$

PROOF. *2.* Note that p can neither contribute particles in $\mathcal{C}$ nor cancel out any particles in it.
3. As for *2*, the $\mathcal{C}$ particles match up. Similarly, a can neither contribute $\mathcal{R}$ particles nor cancel them, so any of these have to come from p. □

LEMMA 3.3.16. ***Checkpoint Equality***
Suppose $a, a^{-1}, b, b^{-1} \notin \mathcal{R}$ *and* $p, p^{-1}, q, q^{-1} \notin \mathcal{C}$. *Let* $\mathcal{U} = \mathcal{C} \cup \mathcal{R}$. *Then*

1. $a\mathcal{C}b \iff a\mathcal{U}b$
2. $a\mathcal{C}b \wedge p \notin \mathcal{R} \iff ap\mathcal{U}b$
3. $a\mathcal{C}b \wedge p\mathcal{R}q \iff ap\mathcal{U}bq$
4. $\neg a\mathcal{U}q$
5. *If* $p \notin \mathcal{D}$ *and* p *and* x *have no opposing particles, then* $x\mathcal{D}y \iff px\mathcal{D}y$.

PROOF. *1.* $a, b \notin \mathcal{R}$ implies $a_{\mathcal{C}} = a_{\mathcal{U}}$ and $b_{\mathcal{C}} = b_{\mathcal{U}}$. The result follows.
2. ($\Longrightarrow$) $(ap)_{\mathcal{U}} = a_{\mathcal{C}}$. ($\Longleftarrow$) All the particles in $\mathcal{C}$ come from a and none of these can be cancelled out by p. Any particles in $\mathcal{R}$ have to be from p and none of these can be cancelled out by a. But b has no $\mathcal{R}$ particles.
3. Similar
4. If $a\mathcal{U}q$ then $a_{\mathcal{C}} = a_{\mathcal{U}} = q_{\mathcal{R}} = q_{\mathcal{U}}$. But no particle from a can be equal to a particle from $\mathcal{R}$.
5. $(px)_{\mathcal{D}} = x_{\mathcal{D}}$. □

3.3.4 The Synchronised Displacement Theorem

We begin with a comment about sorts. The *sorting* of agents is described in [Mil83], and we follow the same scheme, with the added rule that $\mathrm{Sort}(P +\!\!|_{\mathcal{C}}\, Q) = \mathrm{Sort}(P) \cup \mathrm{Sort}(Q)$. This does not necessarily produce the smallest sort for the agent $P +\!\!|_{\mathcal{C}}\, Q$, of course, but the reader can readily prove that it does produce a sort.

Before we look at the SDT, consider how we synchronise displace actions in practice. Let $D \Leftarrow A +\!\!|_{\mathcal{C}}\, B$ and $S \Leftarrow P +\!\!|_{\mathcal{R}}\, Q$. Suppose $\mathcal{F}$ and $\mathcal{G}$ are sets of particles such that of the four sorts $\mathcal{A}$, $\mathcal{B}$, $\mathcal{P}$ and $\mathcal{Q}$, $\mathcal{F}$ has a non-null intersection only with $\mathcal{B}$, and $\mathcal{G}$ has a non-null intersection only with $\mathcal{Q}$. The idea of course is to get the displace actions to synchronise by ensuring that the context synchronises $\mathcal{F}$-actions with $\mathcal{G}$-actions. The only new detail is that here we ensure $\forall b$ such that $B' \xrightarrow{b} B''$, $b \notin \mathcal{C} \Rightarrow b \lessdot \mathcal{F}$, where B' is any *derivative of* B. This ensures that any possible displace action by B, even after some audit actions, carries an $\mathcal{F}$ particle. Similarly, ensure that any displace action by any derivative Q' of Q carries a $\mathcal{G}$ particle.

THEOREM 3.3.17. ***The Synchronised Displacement Theorem***
Let $D \Leftarrow A +\!\!|_{\mathcal{C}}\, B$ *and* $S \Leftarrow P +\!\!|_{\mathcal{R}}\, Q$. *Suppose* $\mathcal{C} \cap \mathcal{S} = \mathcal{C} \cap \overline{\mathcal{S}} = \mathcal{R} \cap \mathcal{D} = \mathcal{R} \cap \overline{\mathcal{D}} = \emptyset$. *Then if* $\mathcal{X}[\,]$ *is a context that synchronises the displace actions of* D *and* S, $\mathcal{X}[(A +\!\!|_{\mathcal{C}}\, B) \mid (P +\!\!|_{\mathcal{R}}\, Q)] \sim \mathcal{X}[(A \mid P) +\!\!|_{\mathcal{C} \cup \mathcal{R}}\, (B \mid Q)]$.

PROOF. We write $\mathcal{U} = \mathcal{C} \cup \mathcal{R}$.

We shall refer to the condition on the sorts as "the sort condition".

As in the proof of associativity, the sort condition enables us to derive side-conditions from each other, and is used in almost all the cases of the bisimulation. For instance, A in the lhs can produce a normal action if it has no particles in $\mathcal{C}$; it doesn't have to meet any condition involving $\mathcal{R}$. It does in the rhs; but the sort condition says there will be no interference from $\mathcal{R}$.

The sort condition also says that D can neither do any actions with audit particles of S in them, nor complement such particles. All audit particles from $A \mid P$ will thus be seen by $B \mid Q$, and vice-versa, and so the rhs can match audit actions by the lhs.

For both sides, an inference can be labelled unambiguously by saying which of A, B, P, or Q contributes. For example, AQ can only mean a normal action by $A \between_{\mathcal{C}} B$ synchronising with a displace action by $P \between_{\mathcal{R}} Q$, and can only be matched by $(A \mid P) \between_{\mathcal{U}} (B \mid Q)$ by *its* AQ inference, an audit action—any other inference would use premises not used on the lhs. The synchronisation condition rules out the lhs AQ, so it is essential that the rhs AQ be ruled out as well. This is done using the sort condition; see below.

Awkward cases on the lhs are always ruled out by the synchronisation condition. This rules out displacements by just one of D or S (Case B below), and also displacements of D in synchrony with normal or audit actions by S (Cases AQ and ABQ below). The sort condition is needed for the rhs, alone for case AQ, and in conjunction with the synchronisation condition for B and ABQ. The two conditions are thus intertwined.

The table below lists all the cases, giving for each the additional assumptions on side-conditions needed by the lhs and rhs. Symmetric cases are indicated in brackets. We also give the interpretation on the lhs, such as the one for AQ above. The reader might find it easier to follow the table in conjunction with the bisimulation, also given below.

Note that the sort condition expands to $a, a^{-1}, b, b^{-1} \notin \mathcal{R}$ and $p, p^{-1}, q, q^{-1} \notin \mathcal{C}$, where a is any action by A, b any action by B, p any action by P and q any action by Q. This means that the conditions of Lemmas 3.3.14, and 3.3.16 (Particles and Checkpoint Equality) are all satisfied.

A[P]. Normal action by one. $\leftrightarrow$ $a \notin \mathcal{C} \iff a \notin \mathcal{U}$. See Particles 1.

B[Q]. Displace action by one.

$\rightarrow$ The lhs inference $\dfrac{\dfrac{B \xrightarrow{r} B' \quad r \notin \mathcal{C}}{A \between_{\mathcal{C}} B \xrightarrow{r} B'}}{(A \between_{\mathcal{C}} B) \mid (P \between_{\mathcal{R}} Q) \xrightarrow{r} B' \mid (P \between_{\mathcal{R}} Q)}$ is ruled out by the synchronisation condition.

$\leftarrow$ The rhs inference $\dfrac{\dfrac{B \xrightarrow{b} B'}{B \mid Q \xrightarrow{b} B' \mid Q} \quad b \notin \mathcal{U}}{(A \mid P) \between_{\mathcal{U}} (B \mid Q) \xrightarrow{b} B' \mid Q}$ is ruled out thus: by Particles 1, $b \notin \mathcal{U}$ implies $b \notin \mathcal{C}$. Then apply the synchronisation condition.

AP. Normal actions by both. $\leftrightarrow$ By Particles 2, $ap \notin \mathcal{U}$ iff $a \notin \mathcal{C} \wedge p \notin \mathcal{R}$.

BQ. Displace actions by both. $\leftrightarrow$ As for Case AP.

AB[PQ]. Audit action by one. $\leftrightarrow$ By Checkpoint Equality 1, $a\mathcal{C}b$ iff $a\mathcal{U}b$. Since $b \notin \mathcal{R}$, $b \in \mathcal{C}$ iff $b \in \mathcal{U}$.

AQ[BP]. Normal action synchronised with displace action.

$\rightarrow$ The lhs inference $\dfrac{\dfrac{A \xrightarrow{a} A' \quad a \notin \mathcal{C}}{A \between_{\mathcal{C}} B \xrightarrow{a} A' \between_{\mathcal{C}} B} \quad \dfrac{Q \xrightarrow{q} Q' \quad q \notin \mathcal{R}}{P \between_{\mathcal{R}} Q \xrightarrow{q} Q'}}{(A \between_{\mathcal{C}} B) \mid (P \between_{\mathcal{R}} Q) \xrightarrow{aq} (A' \between_{\mathcal{C}} B) \mid Q'}$ is ruled out by the synchronisation condition.

$\leftarrow$ Rhs $\dfrac{\dfrac{A \xrightarrow{a} A'}{A \mid P \xrightarrow{a} A' \mid P} \quad \dfrac{Q \xrightarrow{q} Q'}{B \mid Q \xrightarrow{q} B \mid Q' \quad a\mathcal{U}q \quad q \in \mathcal{U}}}{(A \mid P) \between_{\mathcal{U}} (B \mid Q) \xrightarrow{a} (A' \mid P) \between_{\mathcal{U}} (B \mid Q')}$ is ruled out because the side condition can never be satisfied: see Checkpoint Equality 4.

APB[APQ]. Normal action synchronised with audit action.

$\leftrightarrow$ By Checkpoint Equality 2, $a\mathcal{C} \wedge p \notin \mathcal{R}$ iff $ap\mathcal{U}b$. Since $b \notin \mathcal{R}$, $b \in \mathcal{C}$ iff $b \in \mathcal{U}$.

ABQ[PBQ]. Audit synchronous with displace.

$\rightarrow$ The inference $\dfrac{\dfrac{A \xrightarrow{a} A' \quad B \xrightarrow{b} B' \quad a\mathcal{C}b \quad b \in \mathcal{C}}{A \between_{\mathcal{C}} B \xrightarrow{a} A' \between_{\mathcal{C}} B'} \quad \dfrac{Q \xrightarrow{q} Q' \quad q \notin \mathcal{R}}{P \between_{\mathcal{R}} Q \xrightarrow{q} Q'}}{(A \between_{\mathcal{C}} B) \mid (P \between_{\mathcal{R}} Q) \xrightarrow{aq} (A' \between_{\mathcal{C}} B') \mid Q'}$ is ruled out by the synchronisation condition.

← The inference $\dfrac{\dfrac{A \xrightarrow{a} A'}{A \mid P \xrightarrow{a} A' \mid P} \quad \dfrac{B \xrightarrow{b} B' \quad Q \xrightarrow{q} Q'}{B \mid Q \xrightarrow{bq} B' \mid Q'} \quad a \mathcal{U} bq \;\; bq \in \mathcal{U}}{A \mid P \between_{\mathcal{U}} B \mid Q \xrightarrow{a} A' \mid P \between_{\mathcal{U}} B' \mid Q'}$ is ruled out thus: Since $a \notin \mathcal{R}$ and $q \notin \mathcal{C}$, $a \mathcal{U} bq$ implies $q \notin \mathcal{R}$ and $a \mathcal{C} b$. This last implies $b \lessdot \mathcal{C}$. Thus the whole derivation is finally ruled out by the synchronisation condition.

APBQ. Audit synchronous with audit.

↔ By Checkpoint Equality 3, $a \mathcal{C} b \wedge p \mathcal{R} q$ iff $ap \mathcal{U} bq$. Because $b \notin \mathcal{R}$ and $q \notin \mathcal{C}$, $b \in \mathcal{C} \wedge q \in \mathcal{R}$ iff $bq \in \mathcal{U}$.

The bisimulation below shows the six cases remaining.

$$\mathcal{N}(A,B,P,Q) \qquad \langle\, (A \between_{\mathcal{C}} B) \mid (P \between_{\mathcal{R}} Q),\quad (A \mid P) \between_{\mathcal{U}} (B \mid Q) \,\rangle$$

Case A:

$$\frac{\dfrac{\dfrac{A \xrightarrow{a} A' \quad a \notin \mathcal{C}}{A \between_{\mathcal{C}} B \xrightarrow{a} A' \between_{\mathcal{C}} B}}{(A \between_{\mathcal{C}} B) \mid (P \between_{\mathcal{R}} Q) \xrightarrow{a} (A' \between_{\mathcal{C}} B) \mid (P \between_{\mathcal{R}} Q)} \qquad \dfrac{\dfrac{A \xrightarrow{a} A'}{A \mid P \xrightarrow{a} A' \mid P} \quad a \notin \mathcal{U}}{(A \mid P) \between_{\mathcal{U}} (B \mid Q) \xrightarrow{a} (A' \mid P) \between_{\mathcal{U}} (B \mid Q)}}{\text{leading to } \mathcal{N}(A',B,P,Q)}$$

Case AP:

$$\frac{\dfrac{\dfrac{A \xrightarrow{a} A' \quad a \notin \mathcal{C}}{A \between_{\mathcal{C}} B \xrightarrow{a} A' \between_{\mathcal{C}} B} \quad \dfrac{P \xrightarrow{p} P' \quad p \notin \mathcal{R}}{P \between_{\mathcal{R}} Q \xrightarrow{p} P' \between_{\mathcal{R}} Q}}{(A \between_{\mathcal{C}} B) \mid (P \between_{\mathcal{R}} Q) \xrightarrow{ap} (A' \between_{\mathcal{C}} B) \mid (P' \between_{\mathcal{R}} Q)} \qquad \dfrac{\dfrac{A \xrightarrow{a} A' \quad P \xrightarrow{p} P'}{A \mid P \xrightarrow{ap} A' \mid P'} \quad ap \notin \mathcal{U}}{(A \mid P) \between_{\mathcal{U}} (B \mid Q) \xrightarrow{ap} (A' \mid P') \between_{\mathcal{U}} (B \mid Q)}}{\text{leading to } \mathcal{N}(A',B,P',Q)}$$

Case BQ:

$$\dfrac{\dfrac{B \xrightarrow{b} B' \quad b \notin \mathcal{C}}{A \between B \xrightarrow{b} B'} \quad \dfrac{Q \xrightarrow{q} Q' \quad q \notin \mathcal{R}}{P \between Q \xrightarrow{q} Q'}}{(A \between B) \mid (P \between Q) \xrightarrow{bq} B' \mid Q'} \qquad \dfrac{\dfrac{B \xrightarrow{b} B' \quad Q \xrightarrow{q} Q'}{B \mid Q \xrightarrow{bq} B' \mid Q' \quad bq \notin \mathcal{U}}}{(A \mid P) \between (B \mid Q) \xrightarrow{bq} B' \mid Q'}$$

leading to $\mathcal{I}$

Case AB:

$$\dfrac{\dfrac{A \xrightarrow{a} A' \quad B \xrightarrow{b} B' \quad a\mathcal{C}b \quad b \in \mathcal{C}}{A \between B \xrightarrow{a} A' \between B'}}{(A \between B) \mid (P \between Q) \xrightarrow{a} (A' \between B') \mid (P \between Q)}$$

$$\dfrac{\dfrac{A \xrightarrow{a} A'}{A \mid P \xrightarrow{a} A' \mid P} \quad \dfrac{B \xrightarrow{b} B'}{B \mid Q \xrightarrow{b} B' \mid Q} \quad a\mathcal{U}b \quad b \in \mathcal{U}}{(A \mid P) \between (B \mid Q) \xrightarrow{a} (A' \mid P) \between (B' \mid Q)}$$

leading to $\mathcal{N}(A', B', P, Q)$

Case APB:

$$\dfrac{\dfrac{A \xrightarrow{a} A' \quad B \xrightarrow{b} B' \quad a\mathcal{C}b \quad b \in \mathcal{C}}{A \between B \xrightarrow{a} A' \between B'} \quad \dfrac{P \xrightarrow{p} P' \quad p \notin \mathcal{R}}{P \between Q \xrightarrow{p} P' \between Q}}{(A \between B) \mid (P \between Q) \xrightarrow{ap} (A' \between B') \mid (P' \between Q)}$$

$$\dfrac{\dfrac{A \xrightarrow{a} A' \quad P \xrightarrow{p} P'}{A \mid P \xrightarrow{ap} A' \mid P'} \quad \dfrac{B \xrightarrow{b} B'}{B \mid Q \xrightarrow{b} B' \mid Q} \quad ap\mathcal{U}b \quad b \in \mathcal{U}}{(A \mid P) \between (B \mid Q) \xrightarrow{ap} (A' \mid P') \between (B' \mid Q)}$$

leading to $\mathcal{N}(A', B', P', Q)$

Case APBQ:

$$\frac{\dfrac{A \xrightarrow{a} A' \quad B \xrightarrow{b} B' \quad a\mathcal{C}b \quad b \in \mathcal{C}}{A \dagger_{\mathcal{C}} B \xrightarrow{a} A' \dagger_{\mathcal{C}} B'} \qquad \dfrac{P \xrightarrow{p} P' \quad Q \xrightarrow{q} Q' \quad p\mathcal{R}q \quad q \in \mathcal{R}}{P \dagger_{\mathcal{R}} Q \xrightarrow{p} P' \dagger_{\mathcal{R}} Q'}}{(A \dagger_{\mathcal{R}} B) \mid (P \dagger_{\mathcal{R}} Q) \xrightarrow{ap} (A' \dagger_{\mathcal{R}} B') \mid (P' \dagger_{\mathcal{R}} Q')}$$

$$\frac{\dfrac{A \xrightarrow{a} A' \quad P \xrightarrow{p} P'}{A \mid P \xrightarrow{ap} A' \mid P'} \quad \dfrac{B \xrightarrow{b} B' \quad Q \xrightarrow{q} Q'}{B \mid Q \xrightarrow{bq} B' \mid Q'} \quad ap\mathcal{U}bq \quad bq \in \mathcal{U}}{(A \mid P) \dagger_{\mathcal{U}} (B \mid Q) \xrightarrow{ap} (A' \mid P') \dagger_{\mathcal{U}} (B' \mid Q')}$$

leading to $\mathcal{N}(A', B', P', Q')$

□

Discussion

We see that the corresponding theorem for $\dagger$, Theorem 3.2.6, is a special case of this one; there, there are no audit particles, so the sort condition is vacuously satisfied, and the synchronisation condition is the sole condition in that theorem. Again since there are no audit actions it is easier to program this condition there than here; now we must ensure that the synchronising particles are carried by all displace actions in all derivatives of the exception agent. We shall see in the examples in the next chapter that this is quite simple.

If we had the simpler condition "B and Q are synchronised", Case ABQ would be permitted; we have seen that this is impossible to patch up without making fundamental changes in the theorem. It would also have the unfortunate effect of ruling out case AB, i.e. audit action by one component of the system, and we don't want to lose a practically useful case. Thus the condition of synchronised *displacements* seems quite deeply embedded in our whole set-up.

Comparison with the awkward cases for $\dagger$ makes two interesting points. In Case B, where one of D or S does a displace alone, the difficulty in both theorems is that the inference on the other side leads to a system of agents that cannot be matched. In Case AQ for the earlier theorem we had both this structural difficulty and the fact that the action was different. Here it is conceivable that by changing the audit rule to produce ab instead of a, we can get a matching action. But the structural difficulty remains, and can thus be seen as the more basic. The other point is that the new theorem needs the sort condition as well as the synchronisation condition to rule out the rhs inferences. So the sort condition is needed not just for the new inferences.

Case AQ also shows that it is important not to allow the exception agent to do a 1 as its contribution to an audit. If we did, the rhs could do an a by A, and a 1 by Q, the result being an audit. The lhs in matching this must displace P as a result of Q's action.

If we see the SDT as a tool to separate out normal and exception handling in a system rather than to decompose a system into its components, i.e., if we expect to use the theorem left-to-right rather than right-to-left, it is easy to see the synchronisation condition as a restriction on the separate components, and the sort condition as a discipline that curbs unwanted inferences on the rhs. Compare the condition that permits restriction to distribute over —. $(A \mid B)\backslash\mathcal{N}$ has more derivatives than $(A\backslash\mathcal{N}) \mid (B\backslash\mathcal{N})$, and the condition $\mathcal{A} \cap \overline{\mathcal{B}} \cap \mathcal{N} = \overline{\mathcal{A}} \cap \mathcal{B} \cap \mathcal{N} = \emptyset$ rules out precisely these additional possibilities. There is some truth in this assignment of roles to the two conditions, but we only have to look at the *cases that work* to realise that condition and theorem are much more intricately interwoven: every case in the bisimulation invokes the sort condition to assure us of a match!

Lastly, a further comment about applying the SDT. For audit actions, the sort condition means that any agent communicating with the system $D \mid S$ can always tell which of D or S produced the various audit particles.

3.3.5 Difficulties in assessing operators

The proofs about $\frac{1}{2}$ are characterised by excruciating case analyses. So matters are quite delicate. Casually redesigning operators can easily lead to errors in such propositions, and is an indication that we need to develop notation to encapsulate some of the logic here. Checkpoint Equality is an attempt at such a notation.

First, some technicalities. We saw that audit does not preserve weak congruence on the right. This can be remedied by including a fourth rule, *settle*, that allows the exception agent to do 1's without displacing the normal agent. This would imply that displaces will always have to be visible, no bad thing in itself. We have not carried this out because we do not need to substitute on the right of $\frac{1}{2}$ very often, and agree just to be careful when we do this, rather than have our case analyses blow up still further.

There is quite a jump from displace to audit. What about simpler intermediate operators? The γ-conjunction operator from [Mil83] and similar looking operators in LOTOS are good examples. In our terms, the simplest obvious audit would be for both agents to do the entire action. The audit rule would then be exactly the **join** rule we mentioned earlier. We have already said we

would like normal particles to be allowed to synchronise with audits (eg. delete a line and change the display). The technical reason we do not want to enforce a **join** like audit is that the SDT won't hold if we do. $(A \,{}^{\mathcal{C}}\!\!\upharpoonright\!\!\upharpoonright_{\mathcal{C}}\, B) \mid (P \,{}^{\mathcal{R}}\!\!\upharpoonright\!\!\upharpoonright_{\mathcal{R}}\, Q)$ will be able to do a normal action by the first agent and an audit by the second, but $(A \mid P) \,{}^{\mathcal{U}}\!\!\upharpoonright\!\!\upharpoonright_{\mathcal{U}}\, (B \mid Q)$ will not have an audit action to which A, P and Q contribute. One way to still have the SDT then is to insist on synchronised audits as well!

Similarly, if we restrict the projection on $\mathcal{C}$ in the definition above to a single particle, λ, we get into trouble because the first of our two agents here will be able to do audits by both components simultaneously, whereas the second agent will be forced to do audits in sequence.

This is also a good place to say how calculus design gets into the game. Some troublesome cases are caused by agents acting independently; these we cure by synchronising, as we do for displacements. But the two cases in the previous paragraph arose because of synchrony. There is no way to explicitly say two actions should not synchronise, in any of SCCS, ASCCS or MCCS. Hence the attraction of a two sorted calculus where we could rule out certain combinations, and then have for a condition to the SDT that audits should be one kind of action, and normal another.

Another idea that comes immediately to mind is machine assistance. The only problem with this is that it takes a lot of time to develop a system that is flexible enough to let you play with different kinds of operator definitions; almost any system is likely to increase your resistance to exploring if you know you will have to change the system or live without it for the new operator you have in mind. It is also not clear that the investment in such a system is worthwhile until some such proofs have been carried out by hand ... There is no doubt some golden mean, but the author has not found it. Early versions of some of the proofs here were checked by the Prolog system referred to in Chapter 1, but not the current versions.

But we are not entirely without intuition! And the only justification we offer for our design choices is that they are consistent with a reasonable interpretation of the examples we have in mind.

Applications of the audit operator are postponed to the next chapter. We end with a theorem that extends the displaced environment theorem to the audit operator. One of the interesting things about it is that it does not extend further to the checkpoint operator. We shall discuss that fact later.

THEOREM 3.3.18. *Let*

$P \Leftarrow P_N \between_{\mathcal{C}} P_E$

$Q \Leftarrow Q_N \between_{\mathcal{C}} Q_E$

Let $\mathcal{C}[\ ]$ be a context that synchronises the displace actions of P and Q, and induces the environment $O \Leftarrow N \between_{\mathcal{C}} E$, where E accepts only the displace actions of either P or Q, and N accepts no displace actions. Suppose $P_N \sim_N Q_N$ and $P_E \sim_E Q_E$. Then $\mathcal{C}[P] \sim_O \mathcal{C}[Q]$.

PROOF. If O does a displace action, P and Q have to as well; rest follows as before. A normal action of P can only be of interest to O as a normal action, because the presence or absence of particles from $\mathcal{C}$ labels an action unambiguously as either normal or audit. That leaves audit actions of P. Q has to match those that are of interest to O; when it does both Q_N and Q_E have their contributions uniquely decided by the side conditions. Q_N has to do all of the action, and Q_E, it's projection on $\mathcal{C}$. The rest is routine. □

4

FTS Revisited, and the Checkpoint Operator

In this chapter, we study an example of a restartable system that we will later informally relate to the FTS example of the Appendix. We will not translate the latter into our new formalism; we will directly model an informal description of the "log file method" of error recovery using MCCS and our new operators, and then compare the two versions.

We repeat the caveat from Chapter 1: we shall continue to use terms like "fault tolerant system" and "restartable system" interchangeably, meaning by both the very restricted kind of system defined formally here. Related terms like "failure" and "error recovery" also refer always only to the specific interpretations we place on them. "FTS" means the particular example we develop here, and its older version in the Appendix.

4.1 Restartable systems

Our restartable systems are all fallible versions of *rigid*, *strongly determinate* agents (the formal definitions follow). The infallible versions, or *perfect agents* as we shall call them, thus have extremely simple behaviour.

The fallible versions Q of perfect agents P will be of the form $Q_N \dagger Q_E$, where Q_N has a simple relation to P. Failure is represented by a displacement. Thus the fallible agent works correctly until failure, when it is effectively reset to its starting state—we will not use the displace to do anything else. This seems a reasonable way to model *fail-stop* [SS81] processes, and as we saw in

earlier chapters, the $\dashv\vdash$ operator was invented to exhibit this structure clearly.

The basic idea is to augment a fallible strongly determinate process with *stable storage*, a fault free backup agent that records all significant actions in a log file. The audit operator was designed to do just this. Upon failure, the fallible system can be restarted in its initial state, and fed back all the actions in the log file. This completes recovery, and processing continues as usual until the next failure. An obvious efficiency concern is to be able to dump the current state upon a checkpoint command, so that future failures do not take us all the way back to the start state. Later in the chapter, we develop the checkpoint operator, and show how to use it to achieve this.

Strongly determinate agents are necessary for this method to work; otherwise the restarted computation might be different from the original one.

Where the significant action is an output from the fallible agent, the backup doesn't have to record what action was done, just the fact that one was done. If we think of actions as messages, the content of the message need not be recorded—it will be generated again. What the backup agent must do is to absorb the action, otherwise the environment will receive multiple copies.

The definition below is actually a proposition in [Mil80], but in fact the property it expresses is used there in applications as the defining property of a strongly determinate agent.

DEFINITION 4.1.1. *An agent P is strongly determinate iff*
(i) $P \xrightarrow{a} P'$ *and* $P \xrightarrow{a} P''$ *implies* $P' \sim P''$;
(ii) $P \xrightarrow{a} P'$ *implies* P' *is strongly determinate.* □

We can therefore characterise a derivative of a strongly determinate agent P upto strong congruence thus: if $P \xrightarrow{a} P'$ we can write $P(a)$ instead of P'.

Rigidity is primarily a technical necessity: invisible actions cannot be audited. A suggestion for further work is to see if an audit operator with a settle rule, i.e. one that allowed the exception agent to do 1's without causing displacements, can handle non-rigid perfect agents. The only real difficulty with such a programme is that it would entail even worse case analysis than we already have with the present operator.

DEFINITION 4.1.2. *An agent is said to be stable iff it has no 1 actions. An agent is said to be rigid iff all its derivatives are stable.* □

4.1.1 Restartable systems without input

We begin with a proposition about restartable agents that accept no input. We have not distinguished between input and output actions so far, and for a

good reason—we have been using only a pure calculus without value passing. With value passing, input actions can be distinguished from output, because the accepted value in an input can affect the behaviour of the agent: consider $a(v).P(v)$. Without value passing, the difference between input and output is purely conventional; in CCS, one usually calls a an input, and a^{-1} an output. But this convention is not meaningful for SCCS, ASCCS or MCCS: there is no obvious way to extend it to a^2b^{-1}.

We shall continue to use only pure MCCS, so an action by itself is neither input nor output. But we can regard an agent as accepting input during an external choice. $a.A + a.B$ decides autonomously which branch it will take, but $a.A + b.B$ in a context $([\,] \mid E) \backslash\backslash \{a, b\}$ has a choice only if E offers it both a^{-1} and b^{-1}. If E offers only one of them, the choice is said to be made externally. If we define "input" as accepting information from the environment, then external choice is input. Now, $+$ cannot be read automatically as "external choice" even in strongly determinate agents, to which we restrict ourselves. However, such an agent can be put in a context where the choice is made externally, and thus every $+$ is potentially an external choice.

We see that the only way to ensure that a determinate agent accepts no information is to insist that it should have no $+$'s. We shall remove this restriction in the next subsection.

Thus the distinction we are making is actually about the use of $+$, but we shall not label it as such. The input-output distinction is a natural one, as we saw informally in our message passing interpretation, and the link to $+$ is specific to pure CCS.

In the proposition below, P_0 is a strongly determinate agent without input. So the tree representing P_0 is a chain. Let P' and P be points along it, $P_0 \rightarrow \cdots \rightarrow P' \rightarrow \cdots \rightarrow P$. We will therefore use a special case of the notation $P(a)$ that we introduced a little while ago. Since there is only one derivative for every derivative of P_0, we merely write $P(\_)$, showing that we don't need to know the action to know what the derivative is. Also, we can write a_P for the sole action that is possible for P. We use just $\mathcal{P}$ rather than $\mathcal{P}_0$ to stand for the sort of P_0.

The perfect agent is thus defined by a set of equations like this one:

$$P \Leftarrow a_P.P(\_)$$

a_P can be regarded as a function from states to actions, and is really what the perfect agent computes.

Now $P \dashv f.P$ is a stuttering version of P. It makes some progress, does

an f and starts over again. We always assume $f \notin \mathcal{P}$. The f's are external manifestations that stuttering is taking place, and will be absorbed by a demon in applications. In fact, the demon is usually regarded as triggering the stuttering. But $P \dagger f.P$ is visibly different from P even if the f's are absorbed. We could think of the stuttering as arising from the inability of the exception agent to keep track of the normal agent's evolution.

The following agent stutters silently, and its external behaviour is the same as that of P, except that it also does an arbitrary number of f's interleaved with the actions of P. We call such agents restartable versions of P.

$$QD(P', P) \Leftarrow QD_N(P', P) \dagger_{\mathcal{P}} QD_E(P)$$
$$QD_N(P', P) \Leftarrow \textbf{if } P' = P \textbf{ then } a_P.QD(P(\_), P(\_)) \textbf{ else } 1.QD(P'(\_), P)$$
$$QD_E(P) \Leftarrow f.QD(P_0, P) + \Sigma_{a \neq 1}\, a.QD_E(P(\_))$$

$QD(P', P)$ is a restartable version of P; i.e., it represents the external behaviour of a fallible agent plus backup agent[1]. $QD(P, P)$ is the normal state of the restartable system; here it can immediately do the sole action a_P of P, the "external state" seen by the observer. Since all actions of P are in the audit set, and P is rigid, all actions of the system during normal processing are actually audits. (This system has in fact no normal actions except during recovery). Thus when failure occurs, the exception agent knows how far the system has got. Notice that the exception agent needs to know the behaviour of the perfect agent to the extent of being able to compute $P(\_)$, but not to the extent of being able to compute a_P. It is prepared to do anything the normal agent does.

Upon failure we get $QD(P_0, P)$; the "internal state" reverts to the start state. We regard P_0 as fixed for QD, and so avoid a third parameter. The only effect P_0 has is that the longer the chain from P_0 to P, the further back we go upon failure, and the longer the recovery.

The action f represents the occurrence of failure as well as the initiation of *recovery*, during which the agent is represented $QD(P', P)$, with $P' \neq P$. During recovery, invisible moves replace the original actions along the chain—this explains the term $1.QD(P'(\_), P)$. Recovery lasts until QD gets to the furthest point it had previously reached, i.e., until it evolves to $QD(P, P)$. More failures can of course occur during recovery, wiping out that attempt at recovery, and starting a fresh one. Once QD has successfully completed a

[1]In this section, whenever we use the a for an action, $a \notin \mathcal{P}$ holds

recovery, it returns to *normal execution* and continues along the chain until the next failure. Thus QD repeatedly starts down the chain, making varying amounts of progress before it fails and starts over again. We show below that the visible behaviour of $QD(P,P)$ is that of P, apart from the f's.

We abbreviate an agent $G \Leftarrow f.G$ by f^*.

PROPOSITION 4.1.3. *Let P_0 be a strongly determinate, rigid agent expressed without using $+$. Let P' be a derivative of P_0, and P a derivative of P'. Let*

$$QD(P',P) \Leftarrow QD_N(P',P) \between_{\mathcal{P}} QD_E(P)$$
$$QD_N(P',P) \Leftarrow \textbf{if } P' = P \textbf{ then } a_P.QD(P(\_),P(\_)) \textbf{ else } 1.QD(P'(\_),P)$$
$$QD_E(P) \Leftarrow f.QD(P_0,P) + \Sigma_{a\neq 1}\, a.QD_E(P(\_))$$

Then $QD(P',P) \approx P \mid f^*$.

PROOF. The (weak) bisimulation consists of all pairs $\langle QD(P',P), P \mid f^* \rangle$ where P is a derivative of P' and both are derivatives of P_0.

$$\mathcal{N}(P',P) \qquad \langle\, QD(P',P),\quad P \mid f^* \,\rangle$$

Case Recovery, $P' \neq P$. (Normal rule for lhs). $\longrightarrow$*:*

$$QD(P',P) \xrightarrow{1} QD(P'(\_),P) \qquad\qquad P \mid f^* \stackrel{\varepsilon}{\Longrightarrow} P \mid f^*$$
leading to $\mathcal{N}(P'(1),P)$

Case Normal, $P' = P$. (Audit rule for lhs). $\longleftrightarrow$*:*

$$QD(P,P) \xrightarrow{a} QD(P(\_),P(\_)) \qquad\qquad P \mid f^* \xrightarrow{a} P(\_) \mid f^*$$
leading to $\mathcal{N}(P(\_),P(\_))$

Case Failure (displace) $\longleftrightarrow$*:*

$$QD(P',P) \xrightarrow{f} QD(P_0,P) \qquad\qquad P \mid f^* \xrightarrow{f} P \mid f^*$$
leading to $\mathcal{N}(P_0,P)$

□

The proof shows that failure freezes the "external state" P of the restartable system, and that all states (P',P) corresponding to the same external state are equivalent. The only visible behaviour a failed system is capable of is that of the frozen external state, though repeated failures and the resulting infinite stuttering might prevent it from ever completing recovery. However, recovery is always possible.

The proposition also shows in the simplest way why observational equiva-

lence is particularly suited to our model of restartable systems. It is a strong equivalence, so we are of course happy to be able to make a strong statement. Yet the equivalence has a blind spot that we need: it ignores infinite chatter. Testing equivalence would not equate the above two systems.

The above proposition captures the spirit of all the systems we will discuss in this chapter, though we will usually not be able to apply it directly. The following form of the proposition is very much more representation oriented, but reflects what the FTS looks like.

With every agent on the chain that is the derivation tree of P_0, we can associate a number: the number of moves that separate it from P_0. If a derivative is m moves away from P_0, we can call it $P(m)$. Thus P_0 is $P(0)$. Then instead of $a_{P(m)}$ we can write a_m. The program takes advantage of the representation to use $m'' = m - m'$ as a (smaller) counter that represents how much work is left during recovery. At this abstract level, we don't see any representation of the internal state, but it can of course be computed from the information available to QD_N.

COROLLARY 4.1.4. *Let $P(0)$ be a strongly determinate, rigid agent expressed without using $+$. Let $P(m)$ be the derivative of $P(0)$ separated from it by m moves. Let $\mathcal{P}$ be the sort of $P(0)$. Let*

$$QD(m'', m) \Leftarrow QD_N(m'', m) \between_{\mathcal{P}} QD_E(m)$$
$$QD_N(m'', m) \Leftarrow \textbf{if } m'' = 0 \textbf{ then } a_m.QD_N(0, m+1) \textbf{ else } 1.QD_N(m''-1, m)$$
$$QD_E(m) \Leftarrow f.QD(m, m) + \Sigma_{a \neq 1}\, a.QD_E(m+1)$$

where $0 \leq m'' \leq m$. Then $QD(m', m) \approx P(m) \mid f^*$.

PROOF. In the proposition, put $P(m - m'')$ for P', $P(m)$ for P, $P(m+1)$ for $P(\_)$, $P(0)$ for P_0, and a_m for a_P. □

Another proof would of course be directly by bisimulation. This would also prove that all derivatives $QD(i, j)$ of $QD(m'', m)$ where $0 \leq m'' \leq m$, have the property that $0 \leq i \leq j$, a fact worth proving because the definitions are not guaranteed to make any sense otherwise. A similar remark and proof apply to the earlier version with states instead of counters.

It is easy to show by induction on m'' that $QD_N(m'', m) \sim 1^{m''}.QD_N(0, m)$, which says how many 1's the recovery will take.

4.1.2 Restartable systems with input

We now permit $+$ in the perfect agent P, but it is still strongly determinate and rigid. It could therefore be subject to external choice, and we say that

it can accept input. P is now represented by a tree, in general not a chain. Our notation $P(\_)$ is no longer valid, and we revert to $P(a)$ to mean the a-derivative of P. If P does not have an a action, $P(a)$ is of course undefined. We can no longer refer to derivatives of P by a single number. Lastly, we cannot write a_P to uniquely identify an action, since P might have a choice. We let $v(P)$ represent the set of actions P can do (the *initial* actions of P as they are sometimes known). Note that $v(P)$ is a subset of the sort of P.

The perfect agent is thus defined by a set of equations like this one:

$$P \Leftarrow \Sigma_{a \in v(P)}\, a.P(a)$$

Each state of the agent offers a choice to the environment of a set of actions which it is prepared to do. This set is determined by v, which is a function from states to sets of actions. The environment might either force an external choice on P or offer it an internal choice, by being prepared to accept more than one of the actions in $v(P)$.

Contrast P with the perfect agent of the last subsection, which can be thought of as a special case of this one: there, $v(P)$ is always a singleton set. In the light of our earlier discussion, that agent can be seen as accepting no information from the environment, whereas P here does.

$v(P)$ is the real computation that P performs. If $v(P)$ is a singleton set, $P(q)$ can be thought of as a "pure output state". If $v(P) = Act$, the state can be thought of as a "pure input state". Any state where $v(P) \neq Act$ provides information to the user: if the environment restricts P's actions to a set that does not intersect $v(P)$, P deadlocks. It follows that if $P(q)$ never offers a restricted set of actions, i.e., if it were prepared to do any action at any stage, it would be a useless agent. $U(q) \Leftarrow \Sigma_{a \in Act}\, a.U(q{:}a)$ is just an information sink from which the user can learn nothing.

We now define a restartable version QS of P. The problem in programming QS is that during recovery it has to know which branch to take. We solve this by using as parameters, not just the external and internal states, but the entire state histories that lead to these states. We still write just $\mathcal{P}$ rather than $\mathcal{P}_0$ to mean the sort of P_0.

$$QS(q'', q) \Leftarrow QS_N(q'', q) \,\overline{+}_{\mathcal{P}}\, QS_E(q)$$

$$QS_N(q'', q) \Leftarrow \textbf{if } q'' = \varepsilon \textbf{ then } \Sigma_{a \in v(P)}\, a.QS_N(\varepsilon, q{:}P(a)) \textbf{ else } 1.QS_N(tl(q''), q)$$

$$QS_E(q) \Leftarrow f.QS(q, q) + \Sigma_{a \neq 1}\, a.QS_E(q{:}P(a))$$

Here q and q'' are sequences of agents (paths along the tree of P_0), and q

represents P in the sense that the last agent in q is P. The fixed starting state is represented by ε, the empty sequence.

Again, Q_N does not have any explicit representation of the internal state during recovery, though it can be computed. Consider the equation $q'@q'' = q$ where @ means list-append. When there are q', q'' such that the equation is true, we say that q' is an *initial subsequence* of q and that q'' is a *final subsequence* of q. We shall show that the above agents preserve the property that q'' is a final subsequence of q. Then there is always a q' so that $q'@q'' = q$, and we interpret the last element of q' as the internal state. Thus the sequence representing the internal state is just an initial subsequence of that representing the external. It is instructive that at this abstract level, QS_N does not need to represent the internal state; the behaviour of the whole system is after all that of the external state.

Under normal execution, both QS_N and QS_E build up q. QS_N needs it in order to compute $v(P)$, and the QS_E to keep track; as usual, there are only audits, no normal actions. $q{:}P$ appends the agent P to the sequence q. We also write P as an abbreviation for the last element of q.

Upon failure, the exception agent starts off the normal agent in the start state (i.e. $q' = \varepsilon$). During recovery, the (recovering) normal agent effectively accumulates elements from q in q'. ($tl(q)$ is the sequence consisting of all but the first element of q). As we go forward along q, we reconstruct the path that lead from P_0 to P. q'' says how much is left of q that hasn't yet been moved into q'. q' is always a leading subsequence of q, so this process brings q' closer and closer to q. Recovery is over when $q' = q$. We can show by induction on $|q''|$ that $QS_N(q'', q) \sim 1^{|q''|}.\Sigma_{a \in v(P)}\, a.QS_N(\varepsilon, q{:}P(a))$. Here $|q|$ is the length of q.

PROPOSITION 4.1.5. *Let P_0 be a strongly determinate, rigid agent. Let q be a sequence of derivatives of P_0 and let q'' be a final subsequence of q. Let P be the last element of q. Let*

$$QS(q'', q) \Leftarrow QS_N(q'', q) \between_P QS_E(q)$$
$$QS_N(q'', q) \Leftarrow \textbf{if } q'' = \varepsilon \textbf{ then } \Sigma_{a \in v(P)}\, a.QS_N(\varepsilon, q{:}P(a)) \textbf{ else } 1.QS_N(tl(q''), q)$$
$$QS_E(q) \Leftarrow f.QS(q, q) + \Sigma_{a \neq 1}\, a.QS_E(q{:}P(a))$$

Then $QS(q'', q) \approx P \mid f^*$.

PROOF. The (weak) bisimulation consists of all pairs $\langle QS(q'', q), P \mid f^* \rangle$ where q'' is a final subsequence of q.

$$\mathcal{N}(q'', q) \qquad \langle\, QS(q'', q),\quad P \mid f^* \,\rangle$$

Case Recovery, $q'' \neq \varepsilon$. *(Normal rule for lhs).* $\longrightarrow$*:*

$$QS(q'', q) \xrightarrow{1} QS(tl(q''), q) \qquad P \mid f^* \stackrel{\varepsilon}{\Longrightarrow} P \mid f^*$$
leading to $\mathcal{N}(tl(q''), q)$

Case Normal, $q'' = \varepsilon$. *(Audit rule for lhs).* $\longleftrightarrow$*:*

$$QS(\varepsilon, q) \xrightarrow{a} QS(\varepsilon, q{:}P(a)) \qquad P \mid f^* \xrightarrow{a} P(a) \mid f^*$$
leading to $\mathcal{N}(\varepsilon, q{:}P(a))$

Case Failure (displace) $\longleftrightarrow$*:*

$$QS(q'', q) \xrightarrow{f} QS(q, q) \qquad P \mid f^* \xrightarrow{f} P \mid f^*$$
leading to $\mathcal{N}(q, q)$

□

COROLLARY 4.1.6. ***On Deterministic Histories***
If q'' *is a final subsequence of* q*, then* t'' *is a final subsequence of* t *for all derivatives* $QS(t'', t)$ *of* $QS(q'', q)$*. Also, if* $q'@q'' = q$ *and* $q'' \neq \varepsilon$ *then* $q'{:}\,hd(q'')$ *is an initial subsequence of* q*.*

PROOF. The first part is by looking at the lhs of the bisimulation above. The second is merely a property of sequences. ($hd(q)$ is the first element of q). □

There is very little difference between the proofs for systems without and with input, though it is obvious that recovery in the latter case is more expensive to implement, since the backup agent (which we have not yet separated from the fallible one) has to store sequences of states instead of single states.

We will consider implementations of QD and QS in the next section; this will bring us down to a concrete level of description corresponding to that we use in the Appendix. We will consider first a restartable system (FTS) without input, since, in our peculiar terminology, output is a special case of input!

4.2 FTS with no input

We begin by giving the perfect agent an explicit definition:

$$P(m) \Leftarrow a_{v(m)}.P(m+1)$$

This is a special case of the perfect agent in Corollary 4.1.4: P above is strongly determinate, rigid, and has no $+$'s used in it. We think of P as communicating along a set of channels $a_1, a_2, \ldots$ and v is effectively a function from states to channels; it incorporates the real computation that a user wants from P.

We will now show that a restartable version of this perfect agent can be built up from a simpler faulty agent Q:

$$Q(m') \Leftarrow P(m')[c_i/a_i] \dotplus f.Q(0)$$

and a *backup agent* D', to be described. Q renames the a's to c's, these being hidden communications with the backup system which does the actual a's on Q's behalf; our notation for this relabelling is informal but unambiguous. The set of c's and the set of c^{-1}'s are both disjoint from the set of a's. Apart from the renaming, Q is just a fail stop version of P.

During recovery, Q will repeat outputs that it had already done previously. Thus the backup system consists of just a *duplication suppressor*, D', an agent that filters out all output during recovery, but permits it during normal execution, translating c's to a's.

$$\begin{aligned} D'(m'', m) \Leftarrow \textbf{if } m'' = 0 \textbf{ then } & \textstyle\sum_i c_i^{-1} a_i . D'(0, m+1) \\ \textbf{else } & \textstyle\sum_i c_i^{-1} . D'(m''-1, m) \\ & + g . D'(m, m) \end{aligned}$$

Here m'' and m are counters, local and external, by which D' keeps track of recovery and normal execution. m, which never decreases, is the number of a's done, and m'' is the number of silent moves needed to complete recovery. Normal execution is restored when m'' drops to 0. The sum over i spans all the labels P might output. We shall synchronise g with the f in $Q(m')$.

The abstract version $QD(m'', m)$ of the previous section (see Corollary 4.1.4) is here modelled by $(Q(m') \mid D'(m'', m)) \setminus\setminus \{c_i\}$. We have at the start $0 \leq m'' \leq m$ and $m' + m'' = m$. We shall show that these properties are preserved by the agent. Note that D' can work with any Q that communicates along c's, wanting to get them translated to a's. The restriction ensures that none of Q's actions go untranslated.

Failure is modelled as being caused by a demon, $F \Leftarrow f^{-1} g^{-1} . F$.

Now we see intuitively that the behaviour of $(Q(m') \mid D'(m'', m) \mid F) \setminus\setminus \{c, f, g\}$ is observationally the same as that of $P(m)$. As long as no failure occurs, there is no difference between the two. Failure results in $(Q(0) \mid D'(m, m))$ and starts recovery, when Q's actions no longer result in externally visible a's. Once recovery is complete we get back to normal execution. All of this is exactly as in the last proposition, and we only have to establish a connection.

The connection is easily made using the SDT. We first show that D' can be written using $\dotplus$. We could of course have written it so to start with, but we wish to establish the connection with the original version in CCS, of which

our first definition here is an adaptation to take advantage of MCCS. Also, the original version is simpler to start with, and seems obvious as a backup agent to do the task at hand. The structure in the $\lhd$ version is part of the analysis of this naive implementation of a backup agent.

LEMMA 4.2.1. *Let*

$$D(m'',m) \Leftarrow D_N(m'') \lhd_{\{a_i\}} D_E(m)$$
$$D_N(m'') \Leftarrow \textbf{if } m''=0 \textbf{ then } \Sigma_i c_i^{-1}a_i.D_N(0) \textbf{ else } \Sigma_i c_i^{-1}.D_N(m''-1)$$
$$D_E(m) \Leftarrow g.D(m,m) + \Sigma_i a_i.D_E(m+1)$$

and

$$D'(m'',m) \Leftarrow \textbf{if } m''=0 \textbf{ then } \Sigma_i c_i^{-1}a_i.D'(0,m+1) \textbf{ else } \Sigma_i c_i^{-1}.D'(m''-1,m) + g.D'(m,m)$$

Then $D(m'',m) \sim D'(m'',m)$.

PROOF. By the following bisimulation.

$\mathcal{N}(m'',m) \qquad \langle D(m'',m),\ D'(m'',m)\rangle$

Case Normal, $m'' > 0$*:*

$D(m'',m) \xrightarrow{c_i^{-1}} D(m''-1,m) \qquad D'(m'',m) \xrightarrow{c_i^{-1}} D'(m''-1,m)$ *leading to* $\mathcal{N}(m''-1,m)$

Case Audit, $m''=0$*:*

$D(0,m) \xrightarrow{c_i^{-1}a_i} D_N(0) \lhd_{\{a_i\}} D_E(m+1) \qquad D'(0,m) \xrightarrow{c_i^{-1}a_i} D'(0,m+1)$ *leading to* $\mathcal{N}(0,m+1)$

Case Displace:

$D(m'',m) \xrightarrow{g} D(m,m) \qquad D'(m'',m) \xrightarrow{g} D'(m,m)$ *leading to* $\mathcal{N}(m,m)$

□

This shows us the advantage of using the new operators; if we choose our representation carefully, m'' here, we get a clear separation between normal and exception processing. Henceforth we will use only the version $D_N(m'') \lhd_{\{a_i\}} D_E(m)$, where D_N and D_E are defined as in the lemma.

PROPOSITION 4.2.2. ***FTS with no input***
Let

$$P(m) \Leftarrow a_{v(m)}.P(m+1) \qquad \text{perfect agent}$$
$$Q(m') \Leftarrow P(m')[c_i/a_i] \leftthreetimes f.Q(0) \qquad \text{fallible agent}$$
$$F \Leftarrow f^{-1}g^{-1}.F \qquad \text{demon}$$

backup agent (duplication suppressor)

$$D(m'',m) \Leftarrow D_N(m'') \leftthreetimes_{\{a_i\}} D_E(m)$$
$$D_N(m'') \Leftarrow \textbf{if } m''=0 \textbf{ then } \Sigma_i\, c_i^{-1}a_i.D_N(0) \textbf{ else } \Sigma_i\, c_i^{-1}.D_N(m''-1)$$
$$D_E(m) \Leftarrow g.D(m,m) + \Sigma_i\, a_i.D_E(m+1)$$

Then $(Q(m') \mid D(m'',m) \mid F) \backslash\backslash \{c_i, f, g\} \approx P(m)$ *where* $0 \leq m' \leq m$ *and* $m'' = m - m'$.

PROOF. We introduce the following abbreviations.

$$QD(m'',m) \Leftarrow (Q(m') \mid D(m'',m)) \backslash\backslash \{c_i\}$$
$$QD_N(m'',m) \Leftarrow (P(m')[c_i/a_i] \| D_N(m'',m)) \backslash\backslash \{c_i\}$$
$$QD_E(m) \Leftarrow (f.Q(0) \mid D_E(m)) \backslash\backslash \{c_i\}$$

where we assume $0 \leq m'' \leq m$ and $m' + m'' = m$ in the first two. As the abbreviations suggest, we will relate $QD(m',m)$ here to the more abstract version of Corollary 4.1.4.

$QD_N(m'',m)$ can be expanded

$$QD_N(m'',m) \sim (P(m')[c_i/a_i] \textbf{ if } m''=0 \textbf{ then } \Sigma_i\, c_i^{-1}a_i.D_N(0) \textbf{ else } \Sigma_i\, c_i^{-1}.D_N(m''-1)) \backslash\backslash \{c_i\}$$

The **then** part expands to $a_{v(m)}.(P(m+1)[c_i/a_i] \| D_N(0)) \backslash\backslash \{c_i\}$; the **else** part to $1.(P(m'+1)[c_i/a_i] \| D_N(m''-1)) \backslash\backslash \{c_i\}$. It is easy to see by induction on m'' that

$$QD_N(m'',m) \sim \textbf{if } m''=0 \textbf{ then } a_{v(m)}.QD_N(0,m+1) \textbf{ else } 1.QD_N(m''-1,m)$$

since the conditions on the parameters of QD_N are preserved. QD_N is exactly like the abstract normal part except that the action here is more explicitly written $a_{v(m)}$ instead of just a_m. The exception part will be like the abstract one only in a context that synchronises the displace actions of Q and D; we also need the SDT to say this is all we have to prove. But we have such a context.

Write $\mathcal{C}[\,]$ for $([\,] \mid F) \backslash\backslash \{f, g\}$. $\{f\}$-actions and $\{g\}$-actions are synchronised in this context. That is, the displace actions of Q and D are synchronised.

Also, $Q(m')$ has no a^{-1} or a actions. We then have,

$$\begin{aligned}
&\mathcal{C}[QD(m'',m)]\\
\sim&\mathcal{C}[(Q(m') \mid D(m'',m)) \backslash\backslash \{c_i\}]\\
\sim&\mathcal{C}\left[\left((P(m')[c_i/a_i] + f.Q(0)) \mid \left(D_N(m'',m) \leftthreetimes_{\{a_i\}} D_E(m)\right)\right) \backslash\backslash \{c_i\}\right]\\
\sim&\mathcal{C}\left[QD_N(m'',m) \leftthreetimes_{\{a_i\}} QD_E(m)\right]\\
&\qquad\qquad \textit{by SDT, and distributing } \backslash\backslash\{c_i\} \textit{ over } \leftthreetimes_{\{a_i\}}\\
\sim&\mathcal{C}\left[QD_N(m'',m) \leftthreetimes_{\{a_i\}} (fg.QD(m,m) + \Sigma_i\, a_i.QD_E(m+1))\right]\\
&\qquad\qquad f \textit{ and } g \textit{ also synchronised in expanding } QD_E
\end{aligned}$$

Now comparing the quantity in the context with the restartable agent of Corollary 4.1.4, we get

$\mathcal{C}[QD(m'',m)] \approx \mathcal{C}[P(m) \mid (fg)^*]$ *$\mathcal{C}$ respects $\approx$; it has no $+$*

Write out the context, and expand the rhs.

Then $(Q(m') \mid D(m'',m) \mid F) \backslash\backslash \{c,f,g\} \approx P(m) \mid 1^* \approx P(m)$. □

Thus a single application of the SDT allows us to separate normal and exception processing. The context is also needed to tame the expansion in the exception agent after applying the SDT, but it is not doing anything new here, it is only insisting once again that $\{f\}$- and $\{g\}$-actions should be synchronised. At this stage, after we apply the SDT, they have lost their structural interpretation as displace actions of the component parts. The expansion clarifies the picture: fg is the displace action of the restartable agent.

Note that the context is not synchronising "failures", it is synchronising displace actions, one of which is a failure, and the other a detection of failure and initiation of recovery. The SDT replaces two exception agents by a single one. Exception agents (D_E) should not be confused with backup agents (D). In an application where we plan to replace several backup agents by a single one, the SDT demands that all failure detections should be synchronised, and hence all failures also.

4.3 FTS with input

4.3.1 Definition of the perfect agent

We begin by giving the perfect agent an explicit definition.

$P(q) \Leftarrow \Sigma_{i \in v(q)}\, a_i.P(q:a_i)$

We think of it as communicating along a set of channels $a_1, a_2, \ldots$.

$P(q)$ is a special case of the perfect agent in Proposition 4.1.5. It is strongly determinate and rigid. The representation here builds on the fact that if $P(q)$

has the move $P(q) \xrightarrow{a} P'$, we can characterise P' as $(P(q))(a)$. Indeed this is what our definition says, except that we abbreviate $(P(q))(a)$ to $P(q{:}\,a)$. $P(q)$ itself is an abbreviation for "$P(\varepsilon)$ having done the actions in q one by one". The q of this representation is not a sequence of states, but a trace of actions. If we take $P(\varepsilon)$ to be fixed, corresponding to P_0 of the abstract proposition, q completely characterises any of its derivatives, and so we write just $v(q)$ instead of $v(P(q))$.

We shall assume that $v(\varepsilon)$ and therefore $P(\varepsilon)$ are always defined. But just as $P(a)$ is defined only if P has an a action, $v(q)$ and $P(q)$ are not defined for every q. Suppose $v(q) = \{1,2\}$. Then $P(q{:}\,a_3)$ is undefined. Notice that $q{:}\,a_3$ is not a trace of P, and that $q{:}\,a_1$ and $q{:}\,a_2$ are traces. ε is a trace of any agent. $v(q)$, and hence $P(q)$, are always defined if q is a trace of P. Finally, note that $v(q) = \emptyset$ just means that the computation terminates with q. Then q itself is a trace of P, but extensions of it are not.

LEMMA 4.3.1. ***The Trace Lemma***
With the agent definition above, if q is a trace of P, so is any initial subsequence q' of q. If $q{:}\,a_i$ is a trace of P, then $i \in v(q)$. □

The faulty agent Q is similar to the previous one.

$$Q(q') \Leftarrow P(q')[c_i/a_i] \dotplus f.Q(\varepsilon)$$

Note that the parameter to Q is a trace of P, not a trace of Q.

4.3.2 Stable storage

During recovery, the backup agent now has to feed back to Q all the information it consumed during normal processing. This it does by maintaining its own copy of the sequence q of inputs. Since q in fact characterises the state of the perfect agent completely, this is equivalent to maintaining a state history just as QS does in Proposition 4.1.5. We shall refer to this backup agent as the *stable storage*.

$$S(q'', q) \Leftarrow S_N(q'') \dotplus_{\{a_i\}} S_E(q)$$

$$S_N(q'') \Leftarrow \textbf{if } q'' = \varepsilon \textbf{ then } \Sigma_i\, c_i^{-1} a_i.S_N(\varepsilon)$$
$$\textbf{else } hd(q'').S_N(tl(q''))[c_i^{-1}/a_i]$$

$$S_E(q) \Leftarrow g.S(q,q) + \Sigma_i\, a_i.S_E(q{:}\,a_i)$$

Note that during normal execution, the **then** clause of S_N, any action by Q is acceptable. The restriction to $v(q)$ is known only to Q, and S can work with any Q that communicates along c's, wanting to get them translated to a's.

It can be shown by bisimulation that $S(q'', q) \sim S'(q'', q)$, defined below. S'

is the MCCS version of the stable storage, of (the spirit behind) the original CCS version.

$$S'(q'', q) \Leftarrow \textbf{if } q'' = \varepsilon \textbf{ then } \Sigma_i c_i^{-1} a_i.S'(\varepsilon, q{:}a_i)$$
$$\textbf{else } hd(q'').S'(tl(q''), q)[c_i^{-1}/a_i]$$
$$+ g.S(q, q)$$

The relation between S and S' is the same as that between D and D'. We shall only use S, not S'. S' itself is so much more succinct than the original CCS version (see the Appendix) that even working without the new operators or the SDT, one might get a manageable proof. The gain is largely in using synchronous actions whenever we are not interested in intermediate states that would result from interleaving; we are also using only the pure calculus now.

Duplication suppressors simulate stable storage

It is instructive to compare stable storage with the duplication suppressor:

$$D(m'', m) \Leftarrow D_N(m'') \upharpoonright_{\{a_i\}} D_E(m)$$
$$D_N(m'') \Leftarrow \textbf{if } m'' = 0 \textbf{ then } \Sigma_i c_i^{-1} a_i.D_N(0)$$
$$\textbf{else } \Sigma_i c_i^{-1}.D_N(m'' - 1)$$
$$D_E(m) \Leftarrow g.D(m, m) + \Sigma_i a_i.D_E(m + 1)$$

If in S we replace the sequences q, q'' by their lengths, and call them m, m'', we almost get D. Only the **else** clause of S_N causes trouble. This is the part that deals with recovery, and feeds back to Q the exact input it got previously. D does not know what actions were done during normal execution, and is prepared to accept any action by its recovering ward.

Formally, the relation between the two is that $D(|q''|, |q|)$ *simulates* $S(q'', q)$.

DEFINITION 4.3.2. *A simulation $\mathcal{R}$ is a binary relation on $\mathbf{P} \times \mathbf{P}$ such that whenever $P\mathcal{R}Q$ and $a \in Act$ then:*

$P \xrightarrow{a} P'$ implies $\exists Q'$ such that $Q \xrightarrow{a} Q'$ and $P'\mathcal{R}Q'$.

Q is said to simulate P iff there is a simulation $\mathcal{R}$ with $P\mathcal{R}Q$. □

Note which agent simulates which! Every agent simulates **0**. A simulation is thus "half a bisimulation". But note that if P simulates Q and Q simulates P, it does *not* follow that $P \sim Q$. Consider

$$P \Leftarrow a.\mathbf{0} + a.b.\mathbf{0} \text{ and } Q \Leftarrow a.b.\mathbf{0}$$

Then $\{\langle P, Q\rangle, \langle b.\mathbf{0}, b.\mathbf{0}\rangle, \langle \mathbf{0}, \mathbf{0}\rangle, \langle \mathbf{0}, b.\mathbf{0}\rangle\}$ is a simulation that shows that Q simulates P, and $\{\langle Q, P\rangle, \langle b.\mathbf{0}, b.\mathbf{0}\rangle, \langle \mathbf{0}, \mathbf{0}\rangle\}$ is a simulation that shows that P simulates Q. But $P \not\sim Q$, because P can do an a and reach **0**, whereas Q cannot.

The simulation between $S(q'', q)$ and $D(m'', m)$ is $\{\langle S(q'', q), D(|q''|, |q|)\rangle\}$. D can afford to be more tolerant than S because the wards of D can do only one action at each step. D in charge of an agent with input will allow it any input it likes—it will convert every $+$ into internal choice, and it will do this during recovery as well. Thus the $Q(0)$ might take a different path down the tree each time, and $(Q(q') \mid D(|q''|, |q|) \mid F) \setminus\setminus \{f, g\}$ only simulates $P(q)$; it is not equivalent to it. This attempt an FTS could behave just like the perfect agent, but it need not. Since even Q simulates P, $Q \mid D$ is not much of an improvement.

Note that since output only agents are special cases of agents with input, S put in charge of one would work as a backup.

4.3.3 Proof of equivalence

This is very like the previous one, except in one detail. We know that stable storage differs from the duplication suppressor in that during recovery, the former is prepared to offer only one action at each stage to its ward. How do we know these values will be acceptable to Q? That they should be seems intuitively obvious; if they are, it is easy to see that Q will trace the same path down the tree.

PROPOSITION 4.3.3. ***FTS with input***
Let

$$P(q) \Leftarrow \Sigma_{i \in v(q)}\, a_i.P(q{:}\, a_i) \qquad \text{perfect agent}$$
$$Q(q') \Leftarrow P(q')\,[c_i/a_i] \dagger f.Q(\varepsilon) \qquad \text{fallible agent}$$
$$F \Leftarrow f^{-1}g^{-1}.F \qquad \text{demon}$$
$$\text{backup agent (stable storage)}$$
$$S(q'', q) \Leftarrow S_N(q'') \dagger_{\{a_i\}} S_E(q)$$
$$S_N(q'') \Leftarrow \textbf{if } q'' = \varepsilon \textbf{ then } \Sigma_i\, c_i^{-1}a_i.S_N(\varepsilon)$$
$$\textbf{else } hd(q'').S_N(tl(q''))\,[c_i^{-1}/a_i]$$
$$S_E(q) \Leftarrow g.S(q, q) + \Sigma_i\, a_i.S_E(q{:}\, a_i)$$

Then $(Q(q') \mid S(q'', q) \mid F) \setminus\setminus \{c_i, f, g\} \approx P(q)$ *where* q *is a trace of* $P(\varepsilon)$, *and* $q' @ q'' = q$.

PROOF. We shall only do one bit, the expansion of

$$QS_N(q'', q) \Leftarrow (S_N(q'') \mid P(q')\,[c_i/a_i]) \setminus\setminus \{c_i\} \qquad \text{where } q' @ q'' = q$$

Then

$$
\begin{aligned}
& QS_N(q'', q) \\
\sim\ & \big(\big(\textbf{if } q'' = \varepsilon \textbf{ then } \textstyle\sum_i c_i^{-1} a_i . S_N(\varepsilon) \\
& \quad \textbf{else } hd(q'') . S_N(tl(q''))\,[c_i^{-1}/a_i]\big) \,|\, P(q')\,[c_i/a_i]\big) \setminus\setminus \{c_i\} \\
\sim\ & \textbf{if } q'' = \varepsilon \textbf{ then } \textstyle\sum_{i \in v(q)} a_i . (P(q{:}\,a_i)\,[c_i/a_i] \,||\, S_N(\varepsilon)) \setminus\setminus \{c_i\} \\
& \textbf{else } \big(\textstyle\sum_{i \in v(q')} c_i^{-1} . P(q'{:}\,a_i)\,[c_i/a_i]\,||\ hd(q'') . S_N(tl(q''))\,[c_i^{-1}/a_i]\big) \setminus\setminus \{c_i\}
\end{aligned}
$$

The **then** part does an action $\sum_{i \in v(q)} a_i$, which is the normal action for $P(q)$. For the **else** part, suppose $hd(q'') = a_i$. Now, $q' @ q'' = q$, so $tl(q'')$ is a final subsequence of q. So $q'{:}\,hd(q'')$ is an initial subsequence of q, and therefore itself a trace. By the Trace Lemma, if $hd(q'')$ is a_i, then $i \in v(q')$. Thus $S_N(q'')$'s offer of a_i will be accepted by the context $(P(q')\,[c_i/a_i] || [\,]) \setminus\setminus \{c_i\}$ and a 1 results.

We have shown the first actions of $QS_N(q'', q)$ are what we expect. Now we have to show that the resulting agents have respectively the form $QS_N(\varepsilon, q{:}\,a_i)$ and $QS_N(tl(q''), q)$.

For the **then** case, note that if $q'' = \varepsilon$ then $(q'{:}\,a_i) @ q'' = q{:}\,a_i$. For the **else** we need $(q'{:}\,hd(q'')) @ tl(q'') = q$, which follows directly from $q' @ q'' = q$ and $q'' \neq \varepsilon$. Then it is easy to see by induction on $|q''|$ that

$$
\begin{aligned}
QS_N(q'', q) \sim & \textbf{if } q'' = \varepsilon \textbf{ then } \textstyle\sum_{i \in v(q)} a_i . QS_N(\varepsilon, q{:}\,a_i) \\
& \textbf{else } 1 . QS_N(tl(q''), q)
\end{aligned}
$$

which is exactly like the abstract normal part of Proposition 4.1.5 except that the sequences here are of actions rather than of states. The rest of the proof is analogous to the proof for FTS with no input. □

4.4 Further variations of the FTS

We shall not deal with these in detail. A version with both input and output can be dealt with by programming a version that ignores the difference between the two, and records output actions also on the sequence of inputs. This can then be handled as in the previous section. But this is not really satisfactory, for it is of course of interest to study an FTS with two kinds of backup agents. These proofs too proceed along the same lines as the previous ones. Here is an outline of one for input and output.

PROPOSITION 4.4.1. ***FTS with input and output***
Let

$$P(q,m) \Leftarrow \textbf{if } \sigma(q,m) \textbf{ then } \Sigma_{i\in v(q,m)}\, a_i.P(q{:}\, a_i, m) \textbf{ else } b_{j(q,m)}.P(q,m+1) \qquad \text{perfect agent}$$

$$Q(q',m') \Leftarrow P(q',m')[c_i/a_i, d_i/b_i] \leftthreetimes f.Q(\varepsilon,0) \qquad \text{fallible agent}$$

$$F \Leftarrow f^{-1}g^{-1}h^{-1}.F \qquad \text{demon}$$

stable storage

$$S(q'',q) \Leftarrow S_N(q'') \leftthreetimes_{\{a_i\}} S_E(q)$$

$$S_N(q'') \Leftarrow \textbf{if } q''=\varepsilon \textbf{ then } \Sigma_i\, c_i^{-1}a_i.S_N(\varepsilon) \textbf{ else } hd(q'').S_N(tl(q''))[c_i^{-1}/a_i]$$

$$S_E(q) \Leftarrow g.S(q,q) + \Sigma_i\, a_i.S_E(q{:}\, a_i)$$

duplication suppressor

$$D(m'',m) \Leftarrow D_N(m'') \leftthreetimes_{\{b_i\}} D_E(m)$$

$$D_N(m'') \Leftarrow \textbf{if } m''=0 \textbf{ then } \Sigma_i\, d_i^{-1}b_i.D_N(0) \textbf{ else } \Sigma_i\, d_i^{-1}.D_N(m''-1)$$

$$D_E(m) \Leftarrow g.D(m,m) + \Sigma_i\, b_i.D_E(m+1)$$

Then $(Q(q',m') \mid S(q'',q) \mid D(m'',m) \mid F) \setminus\setminus \{c_i,d_i,f,g,h\} \approx P(q,m)$ *where* $P(q',m')$ *and* $P(q,m)$ *are both derivatives of* $P(\varepsilon,0)$, *and* $P(q,m)$ *is a derivative of* $P(q',m')$. $q'@q'' = q$ *and* $m'+m''=m$.

PROOF. In the context $\mathcal{C}[\,] = ([\,] \mid F) \setminus\setminus \{f,g,h\}$, the displace actions of Q, S and D are all synchronised, and none of them can do actions that either have particles in or cancel particles in an audit set of one of the others. We can thus apply the SDT twice and distribute restriction to get

$$(Q(q',m') \mid S(q'',q) \mid D(m'',m) \mid F) \setminus\setminus \{c_i,d_i,f,g,h\}$$
$$\sim \mathcal{C}\left[N(q',q'',q,m',m'',m) \leftthreetimes_{\{a_i,b_i\}} E(q,m)\right]$$

where

$$N(q',q'',q,m',m'',m) \Leftarrow (Q_N(q',m') \mid S_N(q'',q) \mid D_N(m'',m)) \setminus\setminus \{c_i,d_i\}$$
$$E(q,m) \Leftarrow (f.Q(\varepsilon,0) \mid S_E(q) \mid D_E(m)) \setminus\setminus \{c_i,d_i\}$$

The expansion of N has four cases instead of two, and the argument about acceptability of past history and preservation of form is a little more tricky, but otherwise there are no new ideas required. The expansion of E poses no problems.

The rest of the proof is easiest done by brute force bisimulation, which again is tolerable because the worst agent is N, with four cases. □

The above proof is an obvious area for further work. Preliminary investigations suggest the following will work. Any deterministic agent with two sets of disjoint channels, of which it chooses one or the other at every stage, can be transformed into an agent with just one set of channels and therefore one sequence as history instead of a pair. This should ease both the details in the proof of acceptability of past history, as well as allow us to transform the final four case N into a two case agent with the exact form of QS_N of the last section, thus allowing us to avoid a brute force bisimulation.

There remains one issue, that of checkpointing. We deal with this in the following pages, but first we compare the current state of the FTS proof with the Appendix version.

4.5 Discussion

The first thing to note about the FTS proofs in this chapter, is that the difficult parts did not involve any feature peculiar to CCS, but were about data structures, induction and the like. The SDT is applied once, separates out the normal and exception processing, and the rest is similar to reasoning about any other sort of program.

The proofs so far have been structured, and each bit has been a small step. Each captures some concept or property; for example, the only weak bisimulation that actually appears is at the last stage, where we refer to the abstract versions first dealt with, and even this has only to be carried out once. Subsequently we just substitute formally similar expressions. All the other equivalences are strong bisimulations.

Even a hard to prove fact such as that the sequence of past inputs is acceptable on recovery, is broken up: in the concrete stages, we actually proved it only for the normal processing; the fact that the whole agent has a similar property was done at a more abstract level.

What is the price we have paid for this structure? The technical apparatus (MCCS, new operators, an awareness of context dependent bisimulation) is a little more elaborate than just CCS. But this is a price we only pay once; and we might argue that the tools have justified their existence.

So a more serious question is what parts of the problem we have not solved satisfactorily compared to the old version. The first simplification here is that we are using a pure calculus. The second is that we may have failed to model features that some might consider necessary. The key to the reduction in complexity is the ability to synchronise actions whenever we need to, as for the SDT, and also when we are just not interested in the intermediate states.

But states we termed "uninteresting" may not be so to everyone. All we can say is that the MCCS model is a reasonable representation of an informal description of the problem. The author's own dissatisfaction with the current state of affairs is only about the brute force still necessary for the last extensions of the problem.

4.6 The Checkpoint operator

This is a generalisation of the audit operator that allows the normal and exception agents to communicate during an audit; the new action is called a checkpoint.

DEFINITION 4.6.1. *The operational semantics of the checkpoint operator.* $\leftthreetimes_{\mathcal{C}}$ *is a binary operator* $\mathbf{P}^M \times \mathbf{P}^M \to \mathbf{P}^M$ *that is parameterised by a set* $\mathcal{C}$ *of particles, the checkpoint set.*

Normal
$$\frac{A \xrightarrow{a} A' \quad a \notin \mathcal{C}}{A \leftthreetimes_{\mathcal{C}} B \xrightarrow{a} A' \leftthreetimes_{\mathcal{C}} B}$$

Displace
$$\frac{B \xrightarrow{r} B' \quad r \notin \mathcal{C}}{A \leftthreetimes_{\mathcal{C}} B \xrightarrow{r} B'}$$

Checkpoint
$$\frac{A \xrightarrow{a} A' \quad B \xrightarrow{b} B' \quad a\mathcal{C}b}{A \leftthreetimes_{\mathcal{C}} B \xrightarrow{a(b/\mathcal{C})} A' \leftthreetimes_{\mathcal{C}} B'}$$

□

The side conditions on the first two rules are as for the audit operator. The side condition on the last is a generalisation of the one for the audit operator: here we do not have the restriction $b \in \mathcal{C}$. Thus the exception agent can contribute to a checkpoint action particles other than those in the *checkpoint set* $\mathcal{C}$.

All of the discussion following the definition of the audit operator applies, except for the parts dealing with the effect of the second side condition $b \in \mathcal{C}$ for the audit rule.

Communication between normal and exception agents as part of a checkpoint action does not mean that a 1 might result: communication between A and B above is only possible as part of an (externally visible) checkpoint action. This is because of our definition of $a\mathcal{C}b$; a and b have equal non-zero projections on $\mathcal{C}$. This projection cannot be cancelled out. (The actions cannot carry the cancelling particles!) Note also that A and B cannot communicate via a particle in $\mathcal{C}$.

The action performed as a result of a checkpoint inference is called a checkpoint product.

Checkpoint products

We begin by defining an ancillary quantity, the quotient of a by $\mathcal{C}$. It is the product of the particles of a not in $\mathcal{C}$.

DEFINITION 4.6.2. *Let $\mathcal{C}$ be a set of particles, and $a = \lambda_1^{z_1} \ldots \lambda_n^{z_n}$ be an action where the λ_i are all distinct, and the z_i are all non-zero integers. Suppose that $\lambda_1 \ldots \lambda_i \in \mathcal{C}$ and that $\lambda_{i+1} \ldots \lambda_n \notin \mathcal{C}$. The quotient of a by $\mathcal{C}$, denoted by $(a/\mathcal{C})$. is defined to be $\lambda_{i+1}^{z_{i+1}} \ldots \lambda_n^{z_n}$.* □

A subscripted notation would in some ways have been better, because our choice suggests $a(b/\mathcal{C}) = (ab)/\mathcal{C}$ and even $a(b/\mathcal{C}) = (a/\mathcal{C})b$, *both of which are false.* (Take a and b to be particulate, and $\mathcal{C}$ to be $\{a\}$.) To avoid such mistakes, we have to use parentheses and usually treat $(a/\mathcal{C})$ as an indivisible quantity. Despite these drawbacks, we stay with the division notation because subscripts on actions, which frequently appear on top of arrows, are messy and hard to read.

The submonoid problem crops up here as well. $(ab)/\mathcal{C} \neq (a/\mathcal{C})(b/\mathcal{C})$ in general; take a to be particulate, $b = a^{-1}$, and $\mathcal{C} = \{a\}$, the left hand side will be 1 whereas the right hand side will be b.

Now we can define the *checkpoint product* of a and b over $\mathcal{C}$. It need only be defined if a and b are checkpoint-equal. Its main property is that it suppresses the duplication of particles from $\mathcal{C}$ that would result in an ordinary ab. The quantity $a(b/\mathcal{C})$ is a candidate, because it has the latter property if a and b are equal over $\mathcal{C}$, as the following proposition assures us. However, it is defined even otherwise, so we will have to treat it as an extension of checkpoint product. Note that, while duplication is suppressed, $a(b/\mathcal{C})$ retains particles in $\mathcal{C}$, which is of course essential.

PROPOSITION 4.6.3. *If $a_{\mathcal{C}} = b_{\mathcal{C}}$ then $a(b/\mathcal{C}) = (a/\mathcal{C})b$. Also, if $a\mathcal{C}b$ then $a(b/\mathcal{C}) \lessdot \mathcal{C}$.*

PROOF. Expand the two sides; $a_{\mathcal{C}}(a/\mathcal{C})(b/\mathcal{C}) = (a/\mathcal{C})b_{\mathcal{C}}(b/\mathcal{C})$. □

DEFINITION 4.6.4. *The checkpoint product of a and b over $\mathcal{C}$ is $a(b/\mathcal{C})$.* □

The proposition above says that if $a_{\mathcal{C}} = b_{\mathcal{C}}$, the quantity $a(b/\mathcal{C})$ is symmetric between a and b. It follows that under this condition, and therefore also if $a\mathcal{C}b$, the checkpoint product is commutative, which we certainly want.

Our definition settles for the extension instead of the less defined function we had in mind. This is no problem. We merely note that the expression $a(b/C)$, always defined, has the properties of checkpoint product of a and b over C if aCb. So we can use it as the derived checkpoint action in $\dfrac{A \xrightarrow{a} A' \quad B \xrightarrow{b} B' \quad aCb}{A \between B \xrightarrow{a(b/C)} A' \between B'}$. Otherwise we treat it exactly as would any product and quotient, using the proposition above, if needed and applicable, in algebraic simplification.

Another advantage is that we have not introduced further notation. We could have introduced a quantity $a \times_C b$, defined only if $a_C = b_C$. This would have been awkward to handle, for we would have had to check for the condition before we used the expression. Nor would it have saved us the parentheses, because we will often need, say, $p \times (q \times_C r)$ where q and r are equal over C, but p is not equal over C to these two. Indeed in this case our use of ordinary products is positively preferable: it makes it easy to see that $pq(r/C) = qp(r/C)$, for example.

LEMMA 4.6.5. ***Quotients***
Suppose $a, a^{-1} \notin \mathcal{R}$ *and* $p, p^{-1} \notin C$. *Let* $\mathcal{U} = C \cup \mathcal{R}$. *Then*

1. $a/\mathcal{R} = a$
2. $a/\mathcal{U} = a/C$
3. $(a/C)(p/\mathcal{R}) = (ap)/\mathcal{U}$
4. $aCb \Longrightarrow (a(b/C))/C = (a/C)(b/C)$

PROOF. *1* and *2* are special cases of *3*, but they are easy to prove independently, and we use them in the proof of *3*. Consider two subcases. Either a and p have some complementary particles or they don't. If they don't, it is easy to see the result from *1* and *2*. If they do, note that the complementary particles cannot belong to either C or $\mathcal{R}$. Hence the cancelling out takes place on both sides. *4*. The result is obviously true if a and (b/C) have no complementary particles, or such particles in a don't belong to C. The remaining case is interesting. Example: a is particulate, $C = \{a\}$ and $b = a^{-1}$. This would produce 1 on the left and a^{-1} on the right. But such a case cannot arise, because we have aCb. Therefore any particles in a which belong to C also occur in b. We cannot have both these particles and their complements in b. □

4.6.1 Properties of Checkpoint

We shall often say "checkpoint operator", or "$\between$", to mean the family $\between$ indexed by C. This informal usage parallels that of "action prefix". In this sense,

the checkpoint operator is a generalisation of displace, because one instance of the family, $A \dagger\!\!\!\!\dagger_{\emptyset} B$, is the same as $\dagger$.

The relation between the checkpoint and audit operators is a little more difficult to show. The more general operator can simulate the special case.

PROPOSITION 4.6.6. *$A \dagger\!\!\!\!\dagger_{\mathcal{C}} B$ simulates $A \dagger\!\!\!\!\dagger_{\mathcal{C}} B$.*

PROOF. The simulation is $\mathcal{R} \cup \sim$ where $\mathcal{R}$ consists of all pairs of the form $< A \dagger\!\!\!\!\dagger_{\mathcal{C}} B, A \dagger\!\!\!\!\dagger_{\mathcal{C}} B >$. Normal actions by $A \dagger\!\!\!\!\dagger_{\mathcal{C}} B$ can be simulated by $A \dagger\!\!\!\!\dagger_{\mathcal{C}} B$ to reach a new pair in $\mathcal{R}$. Displace actions by $A \dagger\!\!\!\!\dagger_{\mathcal{C}} B$ can be simulated by $A \dagger\!\!\!\!\dagger_{\mathcal{C}} B$ to reach $\sim$. Lastly the inference $\dfrac{P \xrightarrow{a} P' \quad Q \xrightarrow{b} Q' \quad aCb \quad b \in \mathcal{C}}{P \dagger\!\!\!\!\dagger_{\mathcal{C}} Q \xrightarrow{a} P' \dagger\!\!\!\!\dagger_{\mathcal{C}} Q'}$ can be matched by the inference $\dfrac{P \xrightarrow{a} P' \quad Q \xrightarrow{b} Q' \quad aCb}{P \dagger\!\!\!\!\dagger_{\mathcal{C}} Q \xrightarrow{a(b/\mathcal{C})} P' \dagger\!\!\!\!\dagger_{\mathcal{C}} Q'}$ because the side condition is automatically satisfied; we reach a new pair in $\mathcal{R}$. Because $b \in \mathcal{C}$ from the lhs, $a\,(b/\mathcal{C}) = a$. □

The checkpoint operator shares many properties with the audit operator. Desirable properties which fail to hold for either $\dagger$ or for $\dagger\!\!\!\!\dagger_{\mathcal{C}}$ of course also fail for the more general $\dagger\!\!\!\!\dagger_{\mathcal{C}}$, so we refer the reader back to the relevant sections for these.

PROPOSITION 4.6.7. *$P \sim Q \Longrightarrow P \dagger\!\!\!\!\dagger_{\mathcal{C}} R \sim Q \dagger\!\!\!\!\dagger_{\mathcal{C}} R$ and $R \dagger\!\!\!\!\dagger_{\mathcal{C}} P \sim R \dagger\!\!\!\!\dagger_{\mathcal{C}} Q$.* □

Like $\dagger$, $\dagger\!\!\!\!\dagger_{\mathcal{C}}$ is substitutive on the left but not on the right of $\approx$.

PROPOSITION 4.6.8. *$P \approx Q \Longrightarrow P \dagger\!\!\!\!\dagger_{\mathcal{C}} R \approx Q \dagger\!\!\!\!\dagger_{\mathcal{C}} R$.* □

$\dagger\!\!\!\!\dagger_{\mathcal{C}}$ does not preserve either $\approx^{cM}$ or $\approx^{cA}$ on the right, because $\dagger\!\!\!\!\dagger_{\mathcal{C}}$ does not.

$\dagger\!\!\!\!\dagger_{\mathcal{C}}$ is obviously not commutative. We shall postpone associativity till after we have looked at restriction.

The displace and audit operators allow restriction to distribute unconditionally. Restriction $\backslash\mathcal{N}$ distributes over $\dagger\!\!\!\!\dagger_{\mathcal{C}}$ under the condition that P and Q in $P \dagger\!\!\!\!\dagger_{\mathcal{C}} Q$ cannot communicate via a member of $\mathcal{N}$. Here for the first time we pay a price for allowing P and Q to communicate during a checkpoint action. However, the condition is a familiar one from CCS and SCCS, and not too much of a burden. We can weaken it a bit because communication cannot occur via particles in $\mathcal{C}$ as we saw above. The resulting condition is still too strong, because the complementary particles are a problem only if they occur as part of checkpoint actions. Unfortunately, this is awkward to state concisely.

PROPOSITION 4.6.9. *If* $\mathcal{P} \cap \overline{\mathcal{Q}} \cap (\mathcal{N} - \mathcal{C}) = \overline{\mathcal{P}} \cap \mathcal{Q} \cap (\mathcal{N} - \mathcal{C}) = \emptyset$ *then* $(P \dagger_{\mathcal{C}} Q)\backslash\mathcal{N} \sim (P\backslash\mathcal{N}) \dagger_{\mathcal{C}} (Q\backslash\mathcal{N})$

PROOF. By bisimulation. Normal and displace actions are as for $\dagger$. For checkpoint, we need $a(b/\mathcal{C}) \not\leq \mathcal{N} \iff a \not\leq \mathcal{N} \wedge b \not\leq \mathcal{N}$. $\Longleftarrow$ is always true, and $\Longrightarrow$ true under the theorem's condition. □

As for $\dagger$, most absorptive and distributive laws do not hold, but the following exception for $\dagger$ and $\dagger_{\mathcal{C}}$ still survives.

PROPOSITION 4.6.10. $(Q + R) \dagger_{\mathcal{C}} P \sim (Q \dagger_{\mathcal{C}} P) + (R \dagger_{\mathcal{C}} P)$ □

We now look at associativity. Here again the possibility of communication complicates matters. A checkpoint action of the form $a(b/\mathcal{C})$ might have no particles in $\mathcal{D}$ even if a or b do so, because these particles can be cancelled out in the checkpoint product. We did not have to consider this for the audit operator because there the final action was equal to the normal action. Another complication in this proof is that for one case, we even have to prove that the actions on both sides are the same.

PROPOSITION 4.6.11. *If* $\mathcal{P} \cap \mathcal{D} \subseteq \mathcal{C}$ *and* $\mathcal{R} \cap \mathcal{C} \subseteq \mathcal{D}$, *and* P *and* Q *cannot communicate via* $\mathcal{D}$, *and* Q *and* R *cannot communicate via* $\mathcal{C}$, *then* $(P \dagger_{\mathcal{C}} Q) \dagger_{\mathcal{D}} R \sim P \dagger_{\mathcal{C}} (Q \dagger_{\mathcal{D}} R)$.

PROOF. The reader might want to look over the proof for $\dagger_{\mathcal{C}}$ again. Here we point out only the differences in the proofs.

Of the 7 possible cases, P, Q, R, PQ, QR, PR and PQR, the first three (permitted) and PR (ruled out) are exactly the same as in the earlier proof. Here is the table of case analysis followed by the bisimulation for the remaining three cases.

PQ $\rightarrow$ Need $q \not\leq \mathcal{D}$. We have $p(q/\mathcal{C}) \not\leq \mathcal{D}$. Therefore $(p/\mathcal{C})q \not\leq \mathcal{D}$. Since P and Q cannot communicate via $\mathcal{D}$, we get $q \not\leq \mathcal{D}$.

$\leftarrow$ Need $p(q/\mathcal{C}) \not\leq \mathcal{D}$. By Particles 3, $(p/\mathcal{C}) \not\leq \mathcal{D}$. We have $q \not\leq \mathcal{D}$. So $(p/\mathcal{C})q \not\leq \mathcal{D}$ which implies result.

QR $\leftarrow$ Need $q(r/\mathcal{D}) \not\leq \mathcal{C}$. By Particles 3, $(r/\mathcal{D}) \not\leq \mathcal{C}$, and we have $q \not\leq \mathcal{C}$. Result follows.

$\rightarrow$ Need $q \not\leq \mathcal{C}$. By Particles 3, $(r/\mathcal{D}) \not\leq \mathcal{C}$, and we have $q(r/\mathcal{D}) \not\leq \mathcal{C}$. Because Q and R cannot communicate, result follows.

PQR First we have to show that the actions are equal. By Particles 3, $(r/\mathcal{D}) \not\leq \mathcal{C}$. Then $(r/\mathcal{D}) = (r/\mathcal{D})/\mathcal{C}$. The result follows by Quotients 4.

→ Need $q\mathcal{D}r$ and $p\mathcal{C}q(r/\mathcal{D})$. We have $p\mathcal{C}q$. By Particles 3 $(r/\mathcal{D}) \notin \mathcal{C}$; also R and Q do not communicate via $\mathcal{C}$. So $p\mathcal{C}q(r/\mathcal{D})$. We also have $p(q/\mathcal{C})\mathcal{D}r$, which can be written $(p/\mathcal{C})q\mathcal{D}r$. By Particles 3, $(p/\mathcal{C}) \notin \mathcal{D}$; also P and Q do not communicate via $\mathcal{D}$. So $q\mathcal{D}r$.

← Need $p(q/\mathcal{C})\mathcal{D}r$. From $q\mathcal{D}r$ and $(p/\mathcal{C}) \notin \mathcal{D}$ and because P and Q do not communicate via $\mathcal{D}$. We also need $p\mathcal{C}q$. Follows similarly from $p\mathcal{C}q(r/\mathcal{D})$.

The bisimulation below lists these three cases. PR is ruled out and the P, Q and R are exactly as for $\Vdash_{\mathcal{C}}$.

$$\mathcal{N}(P,Q,R) \qquad \langle\, (P \Vdash_{\mathcal{C}} Q) \Vdash_{\mathcal{D}} R, \quad P \Vdash_{\mathcal{C}} (Q \Vdash_{\mathcal{D}} R)\,\rangle$$

Case PQ:

$$\frac{\dfrac{P \xrightarrow{p} P' \quad Q \xrightarrow{q} Q' \quad p\mathcal{C}q}{P \Vdash_{\mathcal{C}} Q \xrightarrow{p(q/\mathcal{C})} P' \Vdash_{\mathcal{C}} Q' \quad p(q/\mathcal{C}) \notin \mathcal{D}}}{(P \Vdash_{\mathcal{C}} Q) \Vdash_{\mathcal{D}} R \xrightarrow{p(q/\mathcal{C})} (P' \Vdash_{\mathcal{C}} Q') \Vdash_{\mathcal{D}} R}$$

$$\frac{P \xrightarrow{p} P' \quad \dfrac{Q \xrightarrow{q} Q' \quad q \notin \mathcal{D}}{Q \Vdash_{\mathcal{D}} R \xrightarrow{q} Q' \Vdash_{\mathcal{D}} R} \quad p\mathcal{C}q}{P \Vdash_{\mathcal{C}} (Q \Vdash_{\mathcal{D}} R) \xrightarrow{p(q/\mathcal{C})} P' \Vdash_{\mathcal{C}} (Q' \Vdash_{\mathcal{D}} R)}$$

leading to $\mathcal{N}(P',Q',R)$

Case QR:

$$\frac{Q \xrightarrow{q} Q' \quad q \notin \mathcal{C}}{\dfrac{P \Vdash_{\mathcal{C}} Q \xrightarrow{q} Q' \quad R \xrightarrow{r} R' \quad q\mathcal{D}r}{(P \Vdash_{\mathcal{C}} Q) \Vdash_{\mathcal{D}} R \xrightarrow{q(r/\mathcal{D})} Q' \Vdash_{\mathcal{D}} R'}}$$

$$\frac{\dfrac{Q \xrightarrow{q} Q' \quad R \xrightarrow{r} R' \quad q\mathcal{D}r}{Q \Vdash_{\mathcal{D}} R \xrightarrow{q(r/\mathcal{D})} Q' \Vdash_{\mathcal{D}} R' \quad q(r/\mathcal{D}) \notin \mathcal{C}}}{P \Vdash_{\mathcal{C}} (Q \Vdash_{\mathcal{D}} R) \xrightarrow{q(r/\mathcal{D})} Q' \Vdash_{\mathcal{D}} R'}$$

leading to $\mathcal{I}$

Case PQR:

$$\dfrac{\dfrac{P \xrightarrow{p} P' \quad Q \xrightarrow{q} Q' \quad p\mathcal{C}q}{P \leftthreetimes Q \xrightarrow{p(q/C)} P' \leftthreetimes Q' \quad R \xrightarrow{r} R' \quad p\,(q/C)\,\mathcal{D}r}}{(P \leftthreetimes Q) \leftthreetimes R \xrightarrow{p(q/C)(r/\mathcal{D})} (P' \leftthreetimes Q') \leftthreetimes R'}$$

$$\dfrac{\dfrac{Q \xrightarrow{q} Q' \quad R \xrightarrow{r} R' \quad q\mathcal{D}r}{P \xrightarrow{p} P' \quad Q \leftthreetimes R \xrightarrow{q(r/\mathcal{D})} Q' \leftthreetimes R' \quad p\mathcal{C}q\,(r/\mathcal{D})}}{P \leftthreetimes (Q \leftthreetimes R) \xrightarrow{p(q(r/\mathcal{D})/\mathcal{C})} P' \leftthreetimes (Q' \leftthreetimes R')}$$

leading to $\mathcal{N}(P', Q', R')$

□

COROLLARY 4.6.12. $(P \leftthreetimes Q) \leftthreetimes R \sim P \leftthreetimes (Q \leftthreetimes R)$

PROOF. This is still unconditional! The conditions on the intersections of the checkpoint sets with the sorts of the agents are automatically satisfied, as for $\leftthreetimes$, and the constraints on communication are automatically satisfied because the normal and exception agents cannot communicate via the checkpoint set. □

Thus we can write $P \leftthreetimes Q \leftthreetimes R$ without ambiguity.

4.6.2 The Synchronised Displacement Theorem

This is similar to that for the audit operator. We point out the differences.

THEOREM 4.6.13. ***The Synchronised Displacement Theorem***
Let $D \Leftarrow A \leftthreetimes B$ *and* $S \Leftarrow P \leftthreetimes Q$. *Suppose* $\mathcal{C} \cap \mathcal{S} = \mathcal{C} \cap \overline{\mathcal{S}} = \mathcal{R} \cap \mathcal{D} = \mathcal{R} \cap \overline{\mathcal{D}} = \emptyset$. *Then if* $\mathcal{X}[\,]$ *is a context that synchronises the displace actions of* D *and* S, $\mathcal{X}[(A \leftthreetimes B) \mid (P \leftthreetimes Q)] \sim \mathcal{X}[(A \mid P) \leftthreetimes_{\mathcal{C} \cup \mathcal{R}} (B \mid Q)]$.

PROOF. We write $\mathcal{U} = \mathcal{C} \cup \mathcal{R}$. The general ideas and even the cases are exactly as for the audit operator. Cases A, AP and BQ are identical to those in the first proof, as are the ruled out cases B, AQ and ABQ. That leaves AB, APB and $APBQ$.

In the bisimulation below, the case headings show that the side conditions on the two sides imply each other, and that the actions done by the two sides are equal. Note that the sort condition can be expanded to $a, a^{-1}, b, b^{-1} \notin \mathcal{R}$ and $p, p^{-1}, q, q^{-1} \notin \mathcal{C}$, where a is any action by A, b any action by B, p any action by P and q any action by Q. This means that the conditions of Lemmas 3.3.14, 3.3.15, 4.6.5 and 3.3.16 are all satisfied. (Particles, Checkpoint Equality and Quotients).

$$\mathcal{N}(A,B,P,Q) \qquad \langle\, (A \leftthreetimes_{\vec{\mathcal{U}}} B) \mid (P \leftthreetimes_{\vec{\mathcal{R}}} Q),\quad (A \mid P) \leftthreetimes_{\vec{\mathcal{U}}} (B \mid Q) \,\rangle$$

Case AB. $a\mathcal{C}b \iff a\mathcal{U}b$ *from Checkpoint Equality 1 and* $b/\mathcal{C} \cup \mathcal{R} = b/\mathcal{C}$ *from Quotients 2:*

$$\dfrac{\dfrac{A \xrightarrow{a} A' \quad B \xrightarrow{b} B' \quad a\mathcal{C}b}{A \leftthreetimes_{\vec{\mathcal{C}}} B \xrightarrow{a(b/\mathcal{C})} A' \leftthreetimes_{\vec{\mathcal{C}}} B'}}{(A \leftthreetimes_{\vec{\mathcal{C}}} B) \mid (P \leftthreetimes_{\vec{\mathcal{R}}} Q) \xrightarrow{a(b/\mathcal{C})} (A' \leftthreetimes_{\vec{\mathcal{C}}} B') \mid (P \leftthreetimes_{\vec{\mathcal{R}}} Q)}$$

$$\dfrac{\dfrac{A \xrightarrow{a} A'}{A \mid P \xrightarrow{a} A' \mid P} \quad \dfrac{B \xrightarrow{b} B'}{B \mid Q \xrightarrow{b} B' \mid Q} \quad a\mathcal{U}b}{(A \mid P) \leftthreetimes_{\vec{\mathcal{U}}} (B \mid Q) \xrightarrow{a(b/\mathcal{U})} (A' \mid P) \leftthreetimes_{\vec{\mathcal{U}}} (B' \mid Q)}$$

leading to $\mathcal{N}(A', B', P, Q)$

Case APB. $a\mathcal{C}b \wedge p \notin \mathcal{R} \iff (ap)\mathcal{U}b$ *from Checkpoint Equality 2, and* $b/\mathcal{U} = b/\mathcal{C}$ *from Quotients 2:*

$$\dfrac{\dfrac{A \xrightarrow{a} A' \quad B \xrightarrow{b} B' \quad a\mathcal{C}b}{A \leftthreetimes_{\vec{\mathcal{C}}} B \xrightarrow{a(b/\mathcal{C})} A' \leftthreetimes_{\vec{\mathcal{C}}} B'} \quad \dfrac{P \xrightarrow{p} P' \quad p \notin \mathcal{R}}{P \leftthreetimes_{\vec{\mathcal{R}}} Q \xrightarrow{p} P' \leftthreetimes_{\vec{\mathcal{R}}} Q}}{(A \leftthreetimes_{\vec{\mathcal{C}}} B) \mid (P \leftthreetimes_{\vec{\mathcal{R}}} Q) \xrightarrow{a(b/\mathcal{C})p} (A' \leftthreetimes_{\vec{\mathcal{C}}} B') \mid (P' \leftthreetimes_{\vec{\mathcal{R}}} Q)}$$

$$\dfrac{\dfrac{A \xrightarrow{a} A' \quad P \xrightarrow{p} P'}{A \mid P \xrightarrow{ap} A' \mid P'} \quad \dfrac{B \xrightarrow{b} B'}{B \mid Q \xrightarrow{b} B' \mid Q} \quad ap\mathcal{U}b}{(A \mid P) \leftthreetimes_{\vec{\mathcal{U}}} (B \mid Q) \xrightarrow{ap(b/\mathcal{U})} (A' \mid P') \leftthreetimes_{\vec{\mathcal{U}}} (B' \mid Q)}$$

leading to $\mathcal{N}(A', B', P', Q)$

Case $APBQ$. $a\mathcal{C}b \wedge p\mathcal{R}q \iff ap\mathcal{U}bq$ *from Checkpoint Equality 3 and* $(b/\mathcal{C})(q/\mathcal{R}) = (bq)/\mathcal{U}$ *from Quotients 3:*

$$\dfrac{\dfrac{A \xrightarrow{a} A' \quad B \xrightarrow{b} B' \quad a\mathcal{C}b}{A \leftthreetimes_{\vec{\mathcal{C}}} B \xrightarrow{a(b/\mathcal{C})} A' \leftthreetimes_{\vec{\mathcal{C}}} B'} \quad \dfrac{P \xrightarrow{p} P' \quad Q \xrightarrow{q} Q' \quad p\mathcal{R}q}{P \leftthreetimes_{\vec{\mathcal{R}}} Q \xrightarrow{p(q/\mathcal{R})} P' \leftthreetimes_{\vec{\mathcal{R}}} Q'}}{(A \leftthreetimes_{\vec{\mathcal{R}}} B) \mid (P \leftthreetimes_{\vec{\mathcal{R}}} Q) \xrightarrow{a(b/\mathcal{C})p(q/\mathcal{R})} (A' \leftthreetimes_{\vec{\mathcal{R}}} B') \mid (P' \leftthreetimes_{\vec{\mathcal{R}}} Q')}$$

$$\dfrac{\dfrac{A \xrightarrow{a} A' \quad P \xrightarrow{p} P'}{A \mid P \xrightarrow{ap} A' \mid P'} \quad \dfrac{B \xrightarrow{b} B' \quad Q \xrightarrow{q} Q'}{B \mid Q \xrightarrow{bq} B' \mid Q'} \quad ap\mathcal{U}bq}{(A \mid P) \leftthreetimes_{\vec{\mathcal{U}}} (B \mid Q) \xrightarrow{ap(bq/\mathcal{U})} (A' \mid P') \leftthreetimes_{\vec{\mathcal{U}}} (B' \mid Q')}$$

leading to $\mathcal{N}(A', B', P', Q')$

□

The forms of the two theorems are exactly alike, and this suggests that checkpoint is in some sense a more stable point in the design space, and that we had stopped rather artificially short with audit. Indeed, in their present forms, the audit was invented as a simplification of the checkpoint operator.

However, the proof of the SDT needs even more finicky attention to particles and side-conditions, and we have to prove action equality sometimes. Such considerations also affect the user who wishes to carry out proofs. A few theorems, notably the distribution of restriction over $+\!\!\!\!\!\!\subset$ are also not as easy to use as the unconditional versions for $+\!\!\!\!\!\!\subset$, though these constraints are no worse than what we are used to from CCS.

The main motivation was to carry out as much work as possible with as simple an operator as possible, and so the audit is interesting as the simplest of its kind that does satisfy the SDT.

The displaced environment theorem for the checkpoint operator does not hold. If we look at the proof for the audit, we see that in the last step we say that given an action a and an audit set C, the contribution by both normal and exception agents is determined. This is not so with the checkpoint, as the particles of a not in C could come from either agent. Now if $P_N +\!\!\!\!\!\!\subset P_E$ does an action a, and it is of interest because the environment $O_N +\!\!\!\!\!\!\subset O_E$ also does an a, the particular b that P_N does may not be of interest to O_N. The conditions of the theorem now do not let us conclude that Q can produce a checkpoint action a.

Thus the unique factoring of the audit action has some effect, at least.

A final comment concerns context dependent bisimulation. Consider the following agents.

$A \Leftarrow ax.\mathbf{0}$

$B \Leftarrow bx.\mathbf{0}$

$Q \Leftarrow a^{-1}.Q + b^{-1}.Q$

Then $(A \mid Q) \backslash\backslash \{a, b\} \sim (B \mid Q) \backslash\backslash \{a, b\}$, which can be read as "in a context that does not care about a'a and b's, A and B are equal". For example a or b might be a bit telling us whether the x is upper or lower case, and if we are printing on an old-fashioned printer that prints only upper case, we have a model of the equivalence induced by the printer—larger than that induced by a modern one. Now the only environment under which A and B are equal is $\mathbf{0}$, which is prepared to equate any pair of agents. This may be regarded as

disappointing, but it is consistent with Larsen's theory, for the environment does not "interact identically" with the two agents A and B. This is related to why the displaced environment does not hold for the checkpoint operator. In both cases, we are using the structure we have on the action set, and this is not considered in [Lar86]. It remains to be seen whether this is a fruitful area for future work.

4.6.3 Application of the Checkpoint operator

Checkpoints are accepted by the restartable system, and cause it to throw away all history prior to the present state. No future failure will ever take it further back than the present state. Since in our programs we are representing the current state itself by a trace, the only visible effect here is that of resetting the state to which the normal agent is taken back upon failure.

The normal agent has to pass its current state to the exception agent, and we do it simply by using a value passing version of MCCS.

PROPOSITION 4.6.14. *Let*

$$\begin{aligned}
P(q) &\Leftarrow \Sigma_{i\in v(q)}\, a_i.P(q{:}\,a_i) && \text{perfect agent}\\
QS(q'',q,q_c) &\Leftarrow QS_N(q'',q) \leftthreetimes_{\overrightarrow{P}} QS_E(q,q_c)\\
QS_N(q'',q) &\Leftarrow \textbf{if } q''=\varepsilon \textbf{ then } \Sigma_{i\in v(q)}\, a_i.QS_N(\varepsilon,q{:}\,a_i)\\
&\quad \textbf{else } 1.QS_N(tl(q''),q)\\
&\quad +tx^{-1}(q'').QS_N(q'',q)\\
QS_E(q,q_c) &\Leftarrow f.QS(q_c,q,q_c)+\Sigma_{a\neq 1}\, a.QS_E(q{:}\,a_i,q_c)\\
&\quad +tx(q'').QS_E(q,q'')
\end{aligned}$$

where q_c is a final subsequence of q, and q'' a final subsequence of q_c. q is a trace of $P(\varepsilon)$. Then $QS(q'',q,q_c)\approx P(q)\mid f^*\mid t^*$.

PROOF. The (weak) bisimulation consists of all pairs $\langle QS(q'',q,q_c), P\mid f^*\mid t^*\rangle$ where the subsequence conditions are fulfilled, and q is a trace of $P(\varepsilon)$.

$$\mathcal{N}(q'',q,q_c) \qquad \langle\, QS(q'',q,q_c),\quad P(q)\mid f^*\mid t^*\,\rangle$$

Case Recovery, $q''\neq\varepsilon$. (Normal rule for lhs). $\longrightarrow$*:*

$$QS(q'',q,q_c)\xrightarrow{1} QS(tl(q''),q,q_c) \qquad P(q)\mid f^*\mid t^*\stackrel{\varepsilon}{\Longrightarrow} P(q)\mid f^*\mid t^*$$

leading to $\mathcal{N}(tl(q''),q,q_c)$

Case Normal, $q''=\varepsilon$. (Checkpoint rule for lhs). $\longleftrightarrow$*:*

$$QS(\varepsilon,q,q_c)\xrightarrow{a_i} QS(\varepsilon,q{:}\,a_i,q_c) \qquad P(q)\mid f^*\mid t^*\xrightarrow{a_i} P(q{:}\,a)\mid f^*\mid t^*$$

leading to $\mathcal{N}(\varepsilon,q{:}\,a_i,q_c)$

Case Failure (displace) $\longleftrightarrow$*:*

$QS(q'', q, q_c) \xrightarrow{f} QS(q_c, q, q_c)$ $\qquad P(q) \mid f^* \mid t^* \xrightarrow{f} P(q) \mid f^* \mid t^*$ *leading to* $\mathcal{N}(q_c, q, q_c)$

Case Checkpoint (checkpoint rule) $\longleftrightarrow$*:*

$QS(q'', q, q_c) \xrightarrow{t} QS(q'', q, q'')$ $\qquad P(q) \mid f^* \mid t^* \xrightarrow{t} P(q) \mid f^* \mid t^*$ *leading to* $\mathcal{N}(q'', q, q'')$

□

At a more concrete level, a similar exchange of information across $\dagger\!\!\dagger$ takes place both in the fallible agent and the backup agent, and as long as new particles t and x are used, the SDT applies routinely to give us separation as before.

4.6.4 Can we now do without displacements?

Since there is a way for the exception agent to affect the behaviour of the normal agent, we could now try to avoid the displace. Obviously, the system should be programmed so that the normal agent cooperates! There are many issues to be discussed here; the only point we wish to make for now is that it is an interesting formal exercise to try this. The result is a sort of symmetric checkpoint operator, where we cannot formally tell which is the exception agent. It has some interesting properties; it is commutative, and the SDT holds without any synchronisation conditions, which is to be expected: there are no displacements to be synchronised.

This operator shows very close similarities to the γ- and Γ- conjunction operators of [Mil83]; in particular a special case of the checkpoint product appears in connection with those operators.

5

On the Non-Derivability of Operators in CCS

5.1 Different notions of derivability

We have seen so far several examples of derived operators: $\delta P \sim \mathrm{fix}X\,(1.X + P)$ in all our calculi, $P \,|\, Q \sim P \times \delta Q + \delta P \times Q$ in ASCCS and derived MCCS, and $a.[\,] \sim a{:}\,\delta[\,]$ in ASCCS. In all these cases, we have an expression not involving the new operator that is strongly congruent to an expression with the new operator outermost. As the last example emphasises, we have a context, not involving the new operator, and with as many holes as the operator's arity, that is strongly congruent to a context involving just the new operator with holes for its parameters.

This is a purely algebraic notion of derivability. We only need remember that it is parametrised by $\sim$; any other equivalence will do just as well. However, we will not need this generality in this chapter, and we keep our notation simple by ignoring this aspect. Thus we will only say "derivable", not "$\sim$-derivable".

DEFINITION 5.1.1. *Let Σ be any signature, and $\rho \notin \Sigma$ be an operator of arity n. Then ρ is said to be derivable from Σ if there is a context $\mathcal{E}$, with n holes and generated by Σ, such that $\mathcal{E}[P_1, \ldots, P_n] \sim \rho(P_1, \ldots, P_n)$ for all $P_1, \ldots, P_n$. If no such $\mathcal{E}$ exists[2], ρ is said to be non-derivable from Σ.* □

[2] *We allow $\mathcal{E}$ to use new names not in Act; in that case, we extend Act to include these new names, and correspondingly adjust the definition of $\sim$, which depends on Act.*

Thus the three operators δ, $|$ and $a.$ above are straightforward examples of derived operators according to our definition. Now consider the following proof of derivability.

PROPOSITION 5.1.2. *$\dashv\!\vdash$ is derivable in MCCS.*

PROOF. Assume that the elements of *Act* are $a_1, a_2, \ldots$. Let $b_1, b_2, \ldots, c_1, c_2, \ldots$ be new names. Then

$$P \dashv\!\vdash Q \sim (P[b_i/a_i] \mid Q[c_i/a_i] \mid Z) \setminus\setminus \{b_i, c_i\}$$

where $Z \Leftarrow \sum_i b_i^{-1} a_i.Z + \sum_i c_i^{-1} a_i.Z'$

and $Z' \Leftarrow \sum_i c_i^{-1} a_i.Z'$

In the expression equivalent to $P \dashv\!\vdash Q$, P and Q have their action translated into b's and c's respectively by the morphism operators. Z is a synchroniser which picks up these translated actions and retranslates them into a's; P and Q cannot act except through Z because of the restriction. After the first action by Q, Z moves to a state where it refuses to pass on any actions by P. It is easy to formally show the bisimulation. □

The expression replacing the $\dashv\!\vdash$ operator is strikingly different from our three examples at the start because of the condition it poses on the action set. As it happens, this way of showing derivability is often used; [Mil83] uses it to show the derivability of the *γ-conjunction* operator, and [dS85] uses it as the basis of his proof that any new operator whose operational semantics is given by structured inference rules is derivable from MEIJE-SCCS.

A third, and this time apparently different, notion of derivability is used by [Mil83]. To prove that morphism is derivable, we note that "for any morphism $\phi : Act \to Act$, the actions of $E[\phi]$ are characterised as follows:

(1) if $E \xrightarrow{a} E'$, then $E[\phi] \xrightarrow{\phi(a)} E'[\phi]$,

(2) if $E[\phi] \xrightarrow{b} G'$, then $E \xrightarrow{a} E'$ for some a and E' such that $b = \phi(a)$ and $G' \equiv E'[\phi]$.

Therefore we must find a derived operator satisfying these two properties". The derived operator is a syntactic form using other operators of the calculus, so it would appear that our contexts would be acceptable, and the two conditions above then are no more than a restatement of a strong congruence relation between the new and old forms. We conclude that there is no real difference between our notion of derivability and this one. In passing, we note that the proof of derivability of morphism also needs to extend *Act*.

5.2 How to prove operators non-derivable

The only general method the author knows to prove an operator ρ non-derivable from a signature Σ is to find a property that is preserved by equality and by all the operators of Σ, but which is not preserved by the new operator ρ.

Examples are easy to find: Given $\mathbf{N}$, the set of natural numbers, and $add2 : \mathbf{N} \times \mathbf{N} \to \mathbf{N}$, the function that adds 2 to any number, we can show that the successor function *succ* cannot be derived because any expression we build with *add2* preserves oddness or evenness, while *succ* does not. No composition of bishop's moves on a chessboard will ever give a knight's move because the former preserve colour while the latter does not. Using only the operators of CCS other than "fix", we can only produce agents with finite derivations; this proves that we cannot derive "fix" from the others. And so on.

Formally, the version we need for CCS is this. We need to find a property Π of agents preserved by strong congruence, i.e. if $\Pi(P)$ and $P \sim Q$ then $\Pi(Q)$. We also demand that for any binary operator $\sigma \in \Sigma$, $\Pi(P)$ and $\Pi(Q)$ should imply $\Pi(\sigma(P,Q))$. Similarly for other arities. Then any expression $\mathcal{E}$ (binary, say) composed of the operators in Σ will have the property that $\Pi(P)$ and $\Pi(Q)$ imply $\Pi(\mathcal{E}(P,Q))$. If we claim that $\mathcal{E}$ is the derived form of ρ, we must have $\mathcal{E}(P,Q) \sim \rho(P,Q)$ and therefore $\Pi(\rho(P,Q))$. If ρ does not preserve Π, so that we know that for some P and Q, $\mathcal{E}(\rho(P,Q))$ does not hold despite $\Pi(P)$ and $\Pi(Q)$, we have a contradiction, and there cannot exist any such $\mathcal{E}$; ρ is non-derivable.

Strong congruence in this argument is our "equality"; it is the equivalence relation we use in our definition of derivability.

But even the use of this standard mathematical technique seems comparatively rare, at least in the author's experience. This might explain why some of the observations we make in this chapter have apparently not been made before, despite their being obvious from the literature; proving operators non-derivable appears not to have been a concern, though proving them derivable is common enough.

In this chapter, we will present a special case of this general technique that seems particularly well suited to transition systems, and use this as well as other versions of the general technique, to prove several of the operators we have encountered non-derivable from various others. The applicability of our technique is a sufficient but not necessary condition for non-derivability, so there are many interesting questions we will not be able to answer. Those

we do tell us more about the relationships between the various calculi we considered in Chapter 2.

5.2.1 A new technique for non-derivability proofs

This is to find an equivalence relation that is preserved by all the operators in Σ but not by ρ. That is to say, we find a congruence wrt Σ that the new operator sees through.

THEOREM 5.2.1. *Let Σ be a signature generating expressions $\mathbf{E}$ and agents $\mathbf{P}$, and let $\bowtie$ be a congruence relation, $\sim \subseteq \bowtie$. Let ρ be an n-ary operator. If there exist P, Q and i such that $P \bowtie Q$ but not*
$\rho(P_1, \ldots, P_{i-1}, P, P_{i+1}, \ldots, P_n) \bowtie \rho(P_1, \ldots, P_{i-1}, Q, P_{i+1}, \ldots, P_n)$,
then ρ is non-derivable from Σ.

PROOF. This is just a special case of our general argument above. If ρ were derivable, we would have had an n-ary context $\mathcal{E}$ such that $\mathcal{E} \sim \rho$. Then since $\mathcal{E}$ would preserve $\bowtie$, we would get

$$\begin{aligned} &\rho(P_1, \ldots, P_{i-1}, P, P_{i+1}, \ldots, P_n) \\ \sim &\mathcal{E}(P_1, \ldots, P_{i-1}, P, P_{i+1}, \ldots, P_n) \\ \bowtie &\mathcal{E}(P_1, \ldots, P_{i-1}, Q, P_{i+1}, \ldots, P_n) \\ \sim &\rho(P_1, \ldots, P_{i-1}, Q, P_{i+1}, \ldots, P_n) \end{aligned}$$

Since $\sim \subseteq \bowtie$, we can replace $\sim$ by $\bowtie$ in the above equation, and we have our contradiction. □

As an immediate corollary, let us name the most well known case.

PROPOSITION 5.2.2. *$+$ is non-derivable from the other operators of CCS; it is also non-derivable from the other operators of MCCS.*

PROOF. In each case, all the other operators preserve $\approx$; $+$ doesn't. □

The technique is not necessarily applicable. Consider the natural numbers generated by 0 and *succ*. Suppose we try to prove that $+$ is not derivable from this signature. Let $\bowtie$ be any equivalence relation over $\mathbf{N}$, such that $= \subseteq \bowtie$ and preserved by *succ*, i.e. $n \bowtie m$ implies $succ(n) \bowtie succ(m)$. That $n \bowtie m$ implies $(p + n) \bowtie (p + m)$ can be shown by induction on p. (Obviously, we have to have a definition of $+$ for all of this to make sense. It could be axiomatic, or operational, in terms of *succ*.) Thus addition cannot see through any equivalence preserved by *succ*. But $+$ is not derivable from *succ* because from just 0 and *succ* we cannot generate a context with two holes in it!

Now observe that in our technique we need to show that the erstwhile congruence $\bowtie$ is *not* preserved by the new operator. Thus we need an equivalence

relation for which there is a failure test. So an equivalence defined by equational axioms will not do, for they can tell us when two terms are equal, but never when they are unequal. However, an axiomatisation of $\bowtie$ over the signature Σ can be useful to show that the example agents P and Q are in fact $\bowtie$-related. We shall see such an application later.

This is why our technique is particularly suited to use with any formal system where the equivalences are set up operationally, for then we have a simple way to prove that two agents are not related. For example to show that $a.\mathbf{0} + b.\mathbf{0} \not\sim a.\mathbf{0}$, we just have to find one action, b, that prevents the bisimulation conditions from being fulfilled.

Milner shows in [Mil83] that $\sim$ is preserved by the operators of SCCS, i.e. that it is a congruence. The only difficult part of this proof is the case of the "fix" construct. To show that the other operators preserve $\sim$ is easy. This is because $\sim$ is in fact preserved by any operator defined by the usual transition semantics. Consider an n-ary operator ρ, and let $Q \sim R$. Then $\rho(P_1, \ldots, P_{i-1}, Q, P_{i+1}, \ldots, P_n) \sim \rho(P_1, \ldots, P_{i-1}, R, P_{i+1}, \ldots, P_n)$. For every action of the left hand side that is not a derivation from an action of Q, the right hand side can do the same action by an identical derivation. If Q contributes to the left hand side, R can match this action, thus ensuring that the right hand side has an exactly similar derivation. Further, the resulting states on both sides continue to have the same form, and so lead back into the bisimulation.

5.3 Some derivability and non-derivability results

Before we begin on our results, we quote a result from [dS85], which says that SCCS augmented with the Δ operator gives a complete calculus from which any operator defined by structured inference rules can be derived, using a technique rather like our proof of the derivability of $\frac{1}{}$. The Δ operator is defined by the rules

$$\textit{Persistent Delay} \qquad \frac{E \xrightarrow{a} E'}{\Delta E \xrightarrow{a} \delta \Delta E'}$$

We shall refer to SCCS augmented with this operator as SCCSΔ.

We organise our results by operator, and prove non-derivability from the rest of the calculus if possible, or from smaller subsets of the rest if that is of interest.

PROPOSITION 5.3.1. *The action prefix operator a: is non-derivable from the rest of SCCSΔ, ASCCS, MCCS and CCS.*

PROOF. The property of asynchrony is this: P is asynchronous if all its proper derivatives are idle. In ASCCS, $a:$ is the only operator that does not preserve asynchrony. The action prefix operator $a.$ of ASCCS is derived from that of SCCS, and is designed to do so. The other operators all preserve asynchrony. Example: If P and Q are asynchronous, we have $P \times Q \rightarrow P' \times Q'$, where P' and Q' are both idle. So is $P' \times Q'$ by Proposition 2.2.2. Therefore $P \times Q$ is asynchronous. □

In any calculus, we can show that the action prefix used in that calculus is non-derivable from the rest of it simply because all other operators preserve the property "has no actions" of any agent.

We know that $+$ is not derivable from the rest of CCS or MCCS. Its derivability from the rest of ASCCS and SCCS is not clear. [dS85] remarks that it can be derived from SCCSΔ, but does not give a proof.

PROPOSITION 5.3.2. *Restriction is not derivable from the other operators of SCCS (or any of the other calculi).*

PROOF. Because the rest only build contexts that will accept all actions. □

PROPOSITION 5.3.3. $\times$ *is not derivable from the rest of SCCS, MCCS or CCS.*

PROOF. The rest of SCCS is the same as the rest of MCCS minus $|$; therefore it is enough if we prove the result for MCCS. All we need to do is to observe that $\times$ can detect any delay discrepancies, and therefore can see through $\approx^{cM}$. A similar proof goes through for CCS. □

PROPOSITION 5.3.4. $\times$ *is derivable neither from derived MCCS nor from the rest of ASCCS.*

PROOF. Syntactically, of course, the rest of ASCCS looks like MCCS minus the $|$. But the action rule makes a difference; if it did not, the $\approx^{cA}$ would have been the same as $\sim$. Now we know that $\approx^{cM}$ is a congruence for derived MCCS as well, i.e. for the rest of ASCCS with the $|$ thrown in. We have $P+1.P \approx^{cM} 1.P$ but taking $P \equiv a.\mathbf{0}$ and $R \equiv b.\mathbf{0}$, we have $R \times (1.P + P) \xrightarrow{ab} \mathbf{0}$ but $R \times (1.P)$ has no such move. □

Call $|$ as defined as in MCCS $|_M$. We know that it can be derived from SCCS. So can $|_C$, which is $|$ in CCS. If P and Q do only a's, we extend Act to b's and c's and then

$$P \mid_C Q \sim ((P[b_i/a_i] \times \delta Q[c_i/a_i] + \delta P[b_i/a_i] \times Q[c_i/a_i])\ Z) \setminus\setminus \{b_i, c_i\}$$

where $Z \Leftarrow \sum_i b_i^{-1} a_i.Z + \sum_i c_i^{-1} a_i.Z' + \sum_i b_i^{-1} c_i.Z + \sum_i b_i c_i^{-1}.Z$

Either P or Q is allowed to act alone and have its actions relayed through Z, but they can also act together by the third and fourth terms, when they must do complementary actions.

$|_C$ can also be derived from $|_M$ by a very similar method:
$P \,|_C\, Q \sim (P[b_i/a_i] \,|_M\, Q[c_i/a_i] \mid Z) \setminus\setminus \{b_i, c_i\}$ where Z is the same as for the derivation from SCCS.

But can $|_C$ be derived from particulate MCCS?

It is obvious that $|_M$ cannot be derived from $|_C$; within CCS there is no way of inferring non-particulate actions, whereas particulate MCCS, with the same syntax, does produce them. This is confirmed by the following:

PROPOSITION 5.3.5. *$|_M$ is not derivable from CCS.*

PROOF. The rest of the operators preserve $\sim_\Lambda$ but $|_M$ does not. □

PROPOSITION 5.3.6. *Non-particulate action prefixes are not derivable from particulate MCCS.*

PROOF. Particulate MCCS cannot enforce synchronisation; with non-particulate action prefixes, we can. □

PROPOSITION 5.3.7. *The action prefix a: of MCCS cannot be derived from the a. of derived MCCS.*

PROOF. What we mean by a: here is just that it has the rule $a{:}P \xrightarrow{a} P$ as opposed to the rule $a.P \xrightarrow{a} \delta P$ for $a.P$. We have a strong congruence $\sim$ on derived MCCS and a strong congruence $\sim_\Pi$ on MCCS, reverting to old notation for a moment. We know that $a.(1.P + P) \sim P$ but $a.(1.P + P) \not\sim_\Pi a.P$. Now on MCCS define a relation $\sim$ which relates those agents that are congruent in derived MCCS. It is preserved by all operators of MCCS and by $a.$, (this is what happens in derived MCCS), but not by a: □

PROPOSITION 5.3.8. *(due to Robin Milner)*
Δ is not derivable from the other operators of SCCS.

PROOF. Suppose $\Delta(X) \sim E(X)$. Pick a not used as a guard in E, i.e. a does not occur non-negatively in any guard in E. Let $A \Leftarrow a{:}\mathbf{0}$. We show that $\Delta(A) \not\sim E[A/X]$.

In fact we show that for any E, with $Free(E) \subseteq \{X\}$, such that a is not used as guard in E, that $E[A/X] \xrightarrow{ab} P$ implies $P \sim \mathbf{0}$. Here ab is an action such that a occurs non-negatively in it.

The proof is by induction on the inference $E[A/X] \xrightarrow{ab} P$. We show three cases. If $E \equiv b{:}\,E'$, then a does not occur non-negatively in it. So the inference is impossible. If $E \equiv E_1 + E_2$ then we must infer the same for E_1 or E_2 by a shorter inference.

Lastly consider $E \equiv \mathrm{fix}Y.F$. If $Y \equiv X$, then $E[A/X] \equiv E$. So $F[E/X] \xrightarrow{ab} P \Longleftrightarrow (F[E/X])\,[A/X] \xrightarrow{ab} P$. Hence $P \sim \mathbf{0}$. If $Y \not\equiv X$, then this case is an application of the inference rule for fix; and we need to assume the action for a shorter inference. Ruled out by induction hypothesis. □

Lastly a proof that demonstrates the usefulness of axiomatisation to set up congruences that we can then see through with the new operator. The following is particularly instructive since it clarifies the importance of | in CCS; just because there is an expansion theorem, we should not conclude that | is derivable, even in the absence of recursion.

PROPOSITION 5.3.9. *| is not derivable from + and action prefix in CCS.*

PROOF. Without |, we can get a tighter congruence. See [HM85]. If we define $\approx$ in terms of $\stackrel{\lambda}{\Longrightarrow}$, where $\lambda \neq 1$, the congruence contained in it can be characterised by a set of 6 axioms, of which we only quote the relevant one here.

$$\mu.(x + 1.y) = \mu.(x + y) + \mu.y$$

This is not preserved by |. A counterexample is $c.\mathbf{0} \mid a.(b.\mathbf{0} + 1.\mathbf{0}) \not\approx c.\mathbf{0} \mid (a.(b.\mathbf{0} + \mathbf{0}) + a.\mathbf{0})$; again please note that this uses $\approx$ defined by observations other than 1. □

6

Conclusions and Future Work

6.1 Conclusions

We have studied bisimulation proofs and developed a new layout for bisimulation proofs, both for agents and expressions, the latter being useful for clearly laying out the side-conditions. Ensuring that the side conditions imply each other in matching pairs in a bisimulation is usually the crucial part of proving general laws.

We introduced a new calculus, MCCS, that can be seen as lying between CCS and ASCCS; like the latter, it allows synchronisation but is not a synchronous calculus. We have studied its properties, deriving most of them from the related calculi, SCCS, ASCCS and CCS. In the process, we clarified the relations between these older calculi, finding them to be closer than had been previously expressed in [Mil83]. In particular, we studied why strong congruence laws do not carry over from SCCS through to CCS (via ASCCS and MCCS). We have also isolated the discrepancies introduced by the translation from a particulate sub-calculus of ASCCS to CCS.

Despite being a more powerful calculus than CCS, MCCS has the same observational congruence as the former. Otherwise, MCCS shares several features with ASCCS, but not those we see as troublesome; thus it is possible to regard MCCS as a more usable version of ASCCS. From this point of view, the rest of the thesis contributes the first examples of use for ASCCS, or at any

rate for an asynchronous calculus that is not restricted to particulate actions.

We defined a class of new operators to capture the structure of systems separable into normal and exception processing. New operators have been frequently suggested as the obvious way to attack specific applications; unfortunately there are not very many examples in the literature. But the few there are, are fairly simple, and so it is at first sight surprising that our new operators should have proved so troublesome. One reason certainly is the need to argue about sorts and particles of audited or checkpointed actions; this leads to large numbers of cases and matters are quite delicately dependent on the exact definitions we use. It is far from obvious that our definitions and chosen method of analysis, separating the normal and exception processing, are the best possible. We hope we have a made a contribution to all these questions by beginning a study.

The two examples we have, the semaphore example and the FTS, convince us that we have made significant progress: the new proofs are notably smaller and clearer than the old ones. Neither is a large or impressive example, but that is the point, considering how matters look in the Appendix, where the same FTS appears enormous. We can report that most of the structure one might hope to see in the FTS is now explicit in our solutions. Among notable features are the localisation of the weak equivalence to a single final step, and the ability to state the intermediate lemmas and propositions in an individually meaningful way. Perhaps most interesting is that the difficulty in the old proof, apart from interleaving, now seems to lie mostly in the data structures. That these difficulties were hidden under several other lesser ones only made them harder to isolate and deal with.

We must point out that we have not yet dealt satisfactorily with several backup agents in an FTS; in this respect we revert to a modest amount of brute force! To compensate, we are now able to deal with checkpoints, which seemed quite hopeless to us earlier. We conclude that we have a good basis on which to proceed to further extensions, including distributed checkpointing, mentioned by [HH86].

As for the Synchronised Displacement Theorem, we show that the absence of other natural distributive laws, etc., makes it about as good a law as we are likely to find. We have also shown that the condition is not as restrictive as initial appearances suggest; at least, we do not need to insist that the exception agents all have to act together all the time, but a failure of one component has to be taken note of by all the others.

Our new technique for proving operators non-derivable, while it is very sim-

ple, seems to be a bit of a novelty because the whole issue of proving anything non-derivable seems to have received rather little attention in Computer Science. Our technique emphasises the importance of operational definitions of both operators and equivalences.

We continued our closer look at the calculus of [Mil83] by studying the operators involved. We proved many of the operators in SCCS and CCS to be primitive, and also developed a better understanding of the way synchrony and asynchrony are related in this chain of calculi.

6.2 Future Work

The "Future work" section of [Pra84] (the Appendix) expresses a definite unwillingness to deal with examples like the FTS too often, particularly in view of the meagre returns, and suggests looking at other formalisms for the same problem. This we have done, and found worthwhile; we are now ready to look around for examples with greater confidence.

Thus an immediate goal is to complete the extensions to the FTS satisfactorily. This is important not only to complete a demonstration of the power of our tools, but also in order to go on and tackle variations of the problem, or additions to it. The most significant contribution this thesis makes is that it breaks down a very large and almost monolithic proof into acceptable steps, as well as reducing the size of the whole very significantly. It is obviously important to know whether the same tools we used here can have good effects elsewhere. As we said in the introduction, we do not expect in the near future to affect the practice of fault tolerance, but we hope to learn from new examples more about CCS and programming and proof techniques.

Issues to be explored include other ways of analysing exception handling; both with related operators as well as with other laws than the SDT. The settle rule for the audit operator is a good example of variations on the design decisions we have presented in this thesis.

On the slightly more theoretical side, completing the study of operators is one obvious suggestion for future work. Further work on the relation between ASCCS and MCCS (the search for safe expressions) might also prove rewarding.

References

[AL81] T. Anderson and P. A. Lee. *Fault Tolerance: Principles and Practice.* Prentice-Hall, London, 1981.

[Bac83] Roland Backhouse. Specification and proof of a regular expression recogniser in synchronous CCS. Internal Report CSR–130–83, University of Edinburgh, March 1983.

[BdN86] T. Bolognesi and R. de Nicola. A tutorial on LOTOS. Technical report, C.N.R. Pisa, March 1986.

[BK86] J.A. Bergstra and J.W. Klop. Algebra of communicating processes. In J.K. Lenstra J.W. de Bakker, M. Hazewinkel, editor, *Proc. CWI Symp. Math. and Comp. Sci.* North-Holland, 1986.

[Bou84] Gerard Boudol. Notes on algebraic calculi of processes. Technical report, INRIA, Sophia-Antipolis, 1984. Lectures given during the Autumn School at Colle-sur-Loup.

[Bri85] Ed Brinksma. On the improvement of LOTOS. Technical Report SEDOS/C/N6, Twente University of Technology, September 1985.

[dN85] Rocco de Nicola. *Testing Equivalences and Fully Abstract Models for Communicating Processes.* PhD thesis, University of Edinburgh, 1985.

[dNH84] Rocco de Nicola and Matthew Hennessy. Testing equivalences for processes. *Theoretical Computer Science*, 34:83, 1984.

[dS85] R. de Simone. Higher level synchronising devices in MEIJE-SCCS. *Theoretical Computer Science*, 37(3):245–268, 1985.

[HH86] Jifeng He and C.A.R. Hoare. Algebraic specification and proof of a distributed recovery algorithm. Technical report, Programming Research Group, Oxford University, 1986.

[HM85] Matthew Hennessy and Robin Milner. Algebraic laws for nondeterminism and concurrency. *JACM*, 32(1):137–162, January 1985.

[Hoa84] C. A. R. Hoare. Notes on communicating sequential systems. In M. Broy, editor, *Control Flow and Data Flow: Concepts of Distributed Programming*, pages 123–204. NATO, August 1984.

[Hoa85] C. A. R. Hoare. *Communicating Sequential Processes.* Prentice Hall, 1985.

[Lar86] Kim Larsen. *Context Dependent Equivalences in CCS.* PhD thesis, University of Edinburgh, 1986.

[LM86] Kim Larsen and Robin Milner. A complete protocol verification using relativised bisimulation. Technical Report ECS-LFCS-86-13, LFCS, University of Edinburgh, September 1986.

[Mil72] Robin Milner. An algebraic definition of simulation between programs. In *IJCAI Proceedings 72.* IJCAI, 1972.

[Mil80] Robin Milner. *A Calculus of Communicating Systems.* Lecture Notes in Computer Science. Springer-Verlag, 1980.

[Mil83] Robin Milner. Calculi for synchrony and asynchrony. *Theoretical Computer Science*, 25:267–310, 1983.

[Mil84] Robin Milner. Lectures on a calculus for communicating systems. In M. Broy, editor, *Control Flow and Data Flow: Concepts of Distributed Programming*, pages 205–228. NATO, August 1984.

[Mil85] G. J. Milne. Circal and the representation of communication, concurrency and time. *ACM Transactions on Programming Languages and Systems*, 7, April 1985.

[Mil86] Robin Milner. Process constructors and interpretations. In H.-J. Kugler, editor, *Information Processing 86*, pages 507–514. IFIP, 1986.

[Mit83] Kevin Mitchell. A semi-formal proof of the alternating bit protocol. 1983.

[OIS85] Technical Committee OIS/121. Open systems interconnection: Lotos, a formal description technique. Draft for Development DD124, British Standards Institution, 1985.

[Par81] D. Park. Concurrency and automata on infinite sequences. In *Proc. 5th. GI Conf., LNCS 104*, 1981.

[PB77] P.M.Merlin and B.Randell. Consistent state restoration in distributed systems. Technical Report 113, University of Newcastle upon Tyne, October 1977.

[Plo81] G.D. Plotkin. A structural approach to operational semantics. Technical report, Aarhus University, 1981.

[PRA82] K. V. S. Prasad, S. Ramani, and J. Abraham. Design and implementation of a simple message switching system. Technical report, NCSDCT, Tata Institute of Fundamental Research, 1982.

[Pra84] K. V. S. Prasad. Specification and proof of a simple fault tolerant system in CCS. Internal Report CSR–178–84, University of Edinburgh, December 1984.

[San82] M. Sanderson. *Proof Techniques for CCS.* PhD thesis, University of Edinburgh, November 1982.

[SS81] R. D. Schlichting and F. B. Schneider. Fail-stop processors: an approach to designing fault-tolerant computing systems. Technical Report TR 81-479, Cornell University, November 1981.

[SW83] M. Shields and M. Wray. A CCS specification of the OSI network service. Internal Report CSR–136–83, University of Edinburgh, August 1983.

[Win84] Glynn Winskel. Synchronization trees. *Theoretical Computer Science*, 34:33–82, 1984.

Appendix

This is an extract from my technical report CSR-178-84, Department of Computer Science, University of Edinburgh, 1984, which formed the Appendix to the submitted thesis. Here we leave out most of Sections 2 and 3. Sections 1.2.3 and 1.3 of the main body explain the significance of the Appendix, but scattered references to it occur throughout the thesis.

SPECIFICATION AND PROOF OF A SIMPLE FAULT TOLERANT SYSTEM IN CCS

K.V.S.Prasad[1]
Department of Computer Science
University of Edinburgh
Edinburgh EH9 3JZ

ABSTRACT

We model failure, and a fail-stop version of a deterministic agent, in CCS. Augmenting this last with stable storage gives us a fault tolerant system, which we prove observationally equivalent to the original agent. We thus make a contribution to the small list of applications of CCS. However, the proof is disappointingly long and complicated for such a simple example. We develop new notation and techniques for the proof, and also apply standard methods such as step-wise refinement and parametrisation. We notice a relation between data transformation and strong bisimulation, and find that inductive definitions for auxiliary functions go well with bisimulation.

---

[1]I am grateful to the University of Edinburgh for a Postgraduate Studentship

Table of Contents

List of Figures

1. Introduction

In this paper, we use CCS (Calculus of Communicating Systems) to formulate a simple fault tolerant system ('FTS') and to prove that its behaviour is equivalent to that of the corresponding fault free system. We have two aims: one is to understand how to model failure and fault tolerance in CCS, and to understand precisely in what senses the two systems referred to are equivalent. We also want to pave the way for further modelling. The second aim is to develop the techniques necessary to carry out a large proof in CCS.

We begin, in this section, with a brief look at fault tolerance. We then rapidly review bisimulation, and survey what has been done so far in the field of proof techniques for CCS.

Section 2 presents our first simple system and proves that a suitable equivalence is observational equivalence. The proof uses a mixture of algebra and bisimulation, with some stepwise refinement. The bisimulations here are fairly complex, despite the use of indexing and parametrisation to simplify them. Following the first CCS formulation, this section also relates our FTS to the practical system [Prasad, Ramani and Abraham 82] that was the original motivation for this work.

Section 3 generalises the first system, and uses more sophisticated techniques in the proof as well as a more elaborate sequence of steps, breaking the proof up into smaller, more meaningful bits. Readers can therefore skip Section 2, but are encouraged to glance through it, particularly the first two subsections.

The final sections review what we have achieved, and present some conclusions.

1.1 Fault Tolerance

The work on fault tolerant software complements that on correctness, formal specifications, proof systems and so on. The distinction becomes clear if we consider the operation of a correct program, formally proven to satisfy its specification, on unreliable hardware. Its results will obviously be unpredictable. Thus a fault tolerant system should be programmed to handle anticipated hardware failures, at least.

More recently, attempts have been made [Anderson and Lee 81] to try to handle unanticipated faults as well, including residual design (algorithmic) faults in the software. Obviously, this does not mean the system takes care of "all"

situations, but that a broad class of error situations is covered. We shall not be taking this approach.

1.1.1 Fail-stop processes

Consider distributed programs consisting of processes communicating with each other by messages. Failures can occur either in the nodes on which the processes run, or in the communication channels, and can be the result of either hardware fault or design fault.

A failure is detected by an acceptance test either by the faulty process itself or by a communicant. Acceptance tests are done, for example, by redundant computations and consistency checks, or by computing assertions. We shall not go into the techniques of error detection; we shall just assume that failures, defined as deviations from the specified "normal" behaviour, are detected by some means at some point after they occur.

If all failures are detected by the faulty node itself, and the failed process is withdrawn from active service until it is remade, we call the process a *fail-stop process*. Fail-stop processes are formally defined by [Schlichting and Schneider 81], who also consider the problem of implementing them. We will not concern ourselves with this problem. For our purposes, the main feature of a fail-stop process is that it never sends out any erroneous information, and therefore errors do not propagate in a system consisting solely of fail-stop processes. Detection of failure by a process other than the faulty one always takes the form of recognising that the failed process has failed to communicate when it should have done so.

If we do not make the fail-stop assumption, erroneous information might be passed between processes and acted on, resulting in erroneous states in more than just the faulty process. By the time the error has been detected, we will need damage assessment techniques and backward error recovery in order to transform the system state back to a well defined and error free one. If consistency of the state is defined, we will have to ensure it is restored as well [Merlin and Randell 77].

In either case, we have to treat the fault, for example by "remaking" the faulty process and resuming service.

1.1.2 A simple error recovery scheme

We now describe a simple error recovery scheme that can be used with deterministic fail stop processes. Suppose such a process is augmented by a reliable backup system ("stable storage"). It can then checkpoint itself

periodically; the checkpointed states are held on the stable storage, as is a log of all messages sent to the message since the last checkpoint. Upon failure, the process is initialised by the backup system to the most recent checkpoint. After this, it is fed back all the messages in the log, allowing it to relive the computations it should have gone through since the last checkpoint. This brings the process up to date and completes the recovery. Note that any messages generated by the failed process during the rerun have to be suppressed.

The rationale behind restarting a program that has just demonstrated the presence of bugs in it is familiar to anyone who has ever re-booted an operating system. Of course if the fault is not in the software, but is instead a transient hardware fault, there is no objection at all to restarting the affected program, and since this is what we shall do in our CCS models later on, the notion of 'fault' in those models is best taken as representing a transient hardware fault.

This is the basic scheme used in Sections 2 and 3, and presented formally there.

1.2 Bisimulation

We assume that the reader is familiar with CCS [Milner 80], [Milner 83a]. We will be using the original asynchronous calculus.

Observational equivalence can be proved by algebra, including the use of laws such as

$$B \approx \tau.\, B$$

$$B + \tau.\, B \approx \tau.\, B$$

$$\alpha.\, \tau.\, B \approx \alpha.\, B$$

$$\alpha.\, (B + \tau.\, C) + \alpha.\, C \approx \alpha.\, (B + \tau.\, C)$$

$\approx$ is a congruence with respect to all operators except +, and we can get around this in practice by putting guards in front:

$$B \approx C \Rightarrow \alpha.\, B \approx^{c} \alpha.\, C$$

However, algebra cannot show the equivalence of recursively defined agents. For these, we have to use the technique of least fixed points. An alternative technique is bisimulation.

Definition 1.2.1: $\mathcal{S}$ is a weak bisimulation iff whenever $P\mathcal{S}Q$

(i) if $P \xrightarrow{a} P'$ then for some Q' $Q \xRightarrow{a} Q'$ and $P'\mathcal{S}Q'$

(ii) if $Q \xrightarrow{a} Q'$ then for some P' $P \xRightarrow{a} P'$ and $Q'\mathcal{S}P'$

Note: In this paper, we allow $P \xRightarrow{\tau} Q$ to mean $P \xRightarrow{\varepsilon} Q$. Thus, if $P \xrightarrow{\tau} P'$, Q might be able to match this by doing nothing: $Q \xRightarrow{\varepsilon} Q$.

If we can find such an $\mathcal{S}$, we write $P \approx Q$. We refer to this as observational or weak equivalence.

Definition 1.2.2: $\mathcal{S}$ is a strong bisimulation iff whenever $P\mathcal{S}Q$

(i) if $P \xrightarrow{a} P'$ then for some Q' $Q \xrightarrow{a} Q'$ and $P'\mathcal{S}Q'$

(ii) if $Q \xrightarrow{a} Q'$ then for some P' $P \xrightarrow{a} P'$ and $Q'\mathcal{S}P'$

If we can find a strong $\mathcal{S}$, we write $P \sim Q$. This is a congruence, and we call it strong equivalence. $P \sim Q \Rightarrow P \approx Q$.

1.3 Proof Techniques

As envisaged in [Milner 80], CCS can be used both for specification and implementation, and we can reason about them using the calculus. Proofs are not conducted in a separate formal language, but instead resemble ordinary proofs in mathematics. This is potentially a great advantage, because of the large amount of mathematical experience available to us. However, while CCS has been used as a vehicle for theoretical research, and a lot of work has been done on its mathematical foundations, there has been comparatively little work done on applying CCS, or on developing techniques for proof.

[Backhouse 83] is the only work we know that gives a large proof of equivalence in a structured way, with meaningful individual steps. The primary tool used here is the Expansion Theorem, and there is also some bisimulation.

A few other large proofs of equivalence are also available, [Sanderson 82], [Mitchell 83]. While these are important in that they establish that proofs are possible, they do not go very far towards developing a useable technique or methodology. This is because they consist of large quantities of extremely detailed but opaque symbolic manipulation. As we shall discover in the course of this paper, it is very hard to do much better with programs even a few lines long, but we do make some progress.

The obvious response to such proofs is to try to build proof checkers. [Sanderson 82] describes a simple algorithm to generate bisimulation relations

(between agents that satisfy certain conditions). Applied blindly, this yields a mass of symbols and no insight, except possibly in the case where the two agents we are looking at are already simple enough, and we have good reason to believe that they are indeed equivalent.

In the sequel, we shall refer occasionally to our own attempts to build a proof checker to handle more structured proofs, using Sanderson's algorithm as one of the basic tools. However, the main thrust of this paper is to gain an understanding of the problem, and to develop techniques. Thus our view is that we are still formulating what we want of a proof checker.

We defer more comments about proof techniques to the Conclusions section.

1.4 Notation

We use subscripts α on τ as comments. τ_α means exactly the same as τ, but the subscript tells us that the silent action was the result of a communication between two agents, one using α and the other $\overline{\alpha}$.

L(P) represents the sort of P.

We use P||Q for $(P|Q)\backslash\alpha_1\backslash\ldots\backslash\alpha_n$ where $\{\alpha_1,\ldots,\alpha_n\}=\text{names}(L(P)\cap\overline{L(Q)})$.

Bisimulations occur throughout this paper, and we found that developing a notation to display them was a useful achievement. Some explanation of our notation: A bisimulation proof posits a relation $\mathcal{S}$ which consists of one or more pairs, each pair consisting of one state from each of the two agents that we want to prove observation equivalent to each other. The first pair consists of the initial states, i.e., the agents themselves. The pairs in our proofs are labelled, and each heads a different paragraph in the proof.

For each pair in $\mathcal{S}$, we have to show that every atomic action of the left hand member, L, can be matched by a sequence of actions (with the same observable content) by the right hand member, R. Lines starting with $\rightarrow$ deal with this. For each action of L we use a single line. We also have to show that the resulting states, L' and R', say, are in $\mathcal{S}$. The label at the end of the line gives the index of <L', R'> in $\mathcal{S}$. If the resulting L' and R' are identical, we don't bother to mention the identity relation at the end of the line.

The lines corresponding to actions of R start with $\leftarrow$. If a line of the proof belongs to both $\leftarrow$ and $\rightarrow$, we mark it with $\longleftrightarrow$.

We have also introduced a notational short cut. If a pair <L,R> leads to <α. L', α. R'>, we have simply claimed that the result of the action was the pair <L',R'> (or sometimes, to be a little more explicit, "α. <L', R'>"). This saves us

the clutter of introducing another pair $<\alpha.\ L',\ \alpha.\ R'>$ into the bisimulation relation, where each member has only one action, leading to the pair $<L',R'>$.

Lastly, we use algebraic simplification on R' where necessary. This simplification, indicated by '=' in the proofs, is actually $\approx^c$. Thus we write $R \rightarrow R'=R''$ leading to the pair $<L',\ R''>$. That is, we are using the concept of "bisimulation upto bisimulation".

2. A first version of a simple fault tolerant system

2.1 CCS formulation

We present a formal version of the scheme described in the introduction.

We will model one example of a perfect process, a fallible version of it, and a perfect backup system, and try to prove that the last two are together equivalent to the first. The perfect process we have chosen for our first attempt, PER, is fairly typical of a process in a distributed system in that it waits for an input, computes a response consisting possibly of several messages, and then goes back to wait for the next input. We have already incorporated a simplification, however. PER has no internal 'state' between inputs.

$$PER_1 \Leftarrow in\ v.\ PER_2(v)$$
$$PER_2(v) \Leftarrow \textbf{if } B(v) \textbf{ then } \overline{out}\ f(v).\ PER_2(g(v)) \textbf{ else } PER_1$$

Suppose k is the lowest integer such that $B(g^k(v))$ is true. Then PER outputs k values, exactly, and we can write more conveniently

$$PER_1 \Leftarrow in\ v.\ PER_{2,0}$$
$$PER_{2,i} \Leftarrow \overline{out}\ u_i.\ PER_{2,i+1} \quad \text{for } i = 0..k\text{-}1$$
$$PER_{2,k} \Leftarrow PER_1$$

Thus $PER_{2,i}$ corresponds to $PER_2(g^i(v))$ in our first description, and u_i to $f(g^i(v))$. Note that k, $u_1 \ldots u_k$, and $PER_{2,1} \ldots PER_{2,k}$ are all functions of v, but we will usually drop the "(v)" in these expressions and in others, in order to reduce the clutter.

Now for the fallible version P of PER. As long as it doesn't fail, P does the same thing as PER. Failure is modelled by incorporating a malevolent 'failer' F, which causes failures by communicating with P. P is always ready to fail, by receiving an f, and then going into a state from which it can only be restarted.

$$P_1 \Leftarrow pin.\ P_{2,0} + f.\ P_3$$
$$P_{2,i} \Leftarrow \overline{pout}\ u_i.\ P_{2,i+1} + f.\ P_3 \quad \text{for } i=0..k\text{-}1$$
$$P_{2,k} \Leftarrow \overline{pfinish}.\ P_1 + f.\ P_3$$
$$P_3 \Leftarrow restart.\ P_1$$

We shall see shortly why we use private versions pin and $\overline{pout}$ rather than the in

and $\overline{out}$ of PER, and also why we need the $\overline{pfinish}$ to signal the end of output corresponding to an input value.

If P fails while computing a response, it must be restarted and fed back the current input. If it fails between inputs, the response to the previous input has already been delivered, so we just have to restart it. The restarting is done by S (for "spy"), so called because it eavesdrops on all inputs to P. That is, S picks up input values on *in* from the outside world and relays them, via *pin*, to P.

$$S_1 \Leftarrow in\ v.\ S_2(v)$$
$$S_2(v) \Leftarrow \overline{pin}\ v.\ S_3(v) + pinf.\ S_4$$
$$S_3(v) \Leftarrow pfinish.\ S_1 + pinf.\ S_4$$
$$S_4(v) \Leftarrow \overline{restart}.\ S_2(v)$$

Rather than model detection of failure by time-outs, we have chosen to have F inform S, via *pinf*, of the fact.

$$F_1 \Leftarrow \overline{f}.\ F_2$$
$$F_2 \Leftarrow \overline{pinf}.\ F_1$$

We see that *pfinish* is needed to tell S when it may go on to accept the next input. Until then, it must hold on to v in case it needs it to feed back to P. Note that S is willing to accept a *pinf* only at points where it is trying to talk to P. Note also how we use the fact that P has no state between inputs. Without this simplification, S would need to store a queue of input messages for P which would have to be fed back to it after a *restart*. (This queue would be zeroed whenever P checkpoints itself. An interesting special case would have no checkpoints, and a permanent store of all inputs to P. This is the case we deal with in Section 3.)

But all is not yet well. Whenever P is restarted, it will generate again all the output corresponding to the input value. But some of this output might already have been relayed to the outside world before failure. We therefore introduce another agent DS (for "duplication suppressor"), which picks up the values on *pout* and counts the messages generated during the rerun, using a 'local counter' l. DS starts relaying them to the outside world, via *out*, only after P has caught up with history. This happens when the local counter equals the 'external counter' e, which counts the number of messages relayed to the outside.

DS also needs to be informed of when a rerun has begun. We do this by

modifying S to relay *pinf* to DS as *inf*. Though not necessary, we also inform DS of a successful termination, via *finish*, so that it can initialise its counters.

The whole system is summarised in Fig. 2-1.

A very important feature of the system is the division of the backup system into S and DS. The consequences of this pervade all of this section and the next one, where the more general version of the system has the same feature. Before we arrived at this system, we had tried to work with a version that combined S and DS into a single backup agent that looked after both input and output. This turned out to be very difficult. But with the separation of S and DS, we can partition the proof into smaller chunks. It turns out that this partition is a particularly good one, because, for example, FTS with DS removed has an easily described behaviour.

2.1.1 Practical motivation

Error recovery for databases by means of checkpoints and audit trails of transactions since the last checkpoint is an old technique. But it can also be applied individually to each process of a distributed system, and at a very low cost under certain conditions. Further, if we have fail stop processes, the recovered process not only has the correct internal state, its communicants do not even notice any change in its behaviour in terms of messages. (In fact, they cannot find out about its state except by implications from messages). What we have modelled in CCS is a simplified version of such an error recovery scheme proposed in [Prasad, Ramani and Abraham 82]as an enhancement to a message switching system.

We describe the system briefly to set our CCS programs in a real context, and to indicate that there are practical situations where even such simple recovery schemes are applicable. The formulation in Section 3 uses a few more of the features described below, but not all.

The system consisted of a large number of simple communicating processes, and it was found in practice that errant behaviour on the part of one process was detected fairly soon, either by other processes doing consistency checks or by an operator. The large number of processes meant that the fault was localised very quickly once it was detected, and since most of the processes performed just one simple task, the cause of failure was also easily detected. For example, it often happened during development that a known special case occurred that the process was not yet programmed to handle.

The proposed error recovery scheme was merely an automated version of a manual scheme that was evolved to handle these failures: kill the errant

The perfect process is defined by

$PER_1 \Leftarrow in\ v.\ PER_{2,0}$
$PER_{2,i} \Leftarrow \overline{out}\ u_i.\ PER_{2,i+1}$ for i = 0..k-1
$PER_{2,k} \Leftarrow PER_1$

The fallible version of the perfect process:

$P_1 \Leftarrow pin\ v.\ P_{2,0} + f.\ P_3$
$P_{2,i} \Leftarrow \overline{pout}\ u_i.\ P_{2,i+1} + f.\ P_3$ for i = 0..k-1
$P_{2,k} \Leftarrow \overline{pfinish}.\ P_1 + f.\ P_3$
$P_3 \Leftarrow restart.\ P_1$

The demon agent that causes failure:

$F_1 \Leftarrow \overline{f}.\ F_2$
$F_2 \Leftarrow \overline{pinf}.\ F_1$

The spy agent that eavesdrops on inputs, receives and relays information of failure, and restarts P:

$S_1 \Leftarrow in\ v.\ S_2(v)$
$S_2(v) \Leftarrow \overline{pin}\ v.\ S_3(v) + pinf.\ S_5(v)$
$S_3(v) \Leftarrow pfinish.\ S_4 + pinf.\ S_5(v)$
$S_4 \Leftarrow \overline{finish}.\ S_1$
$S_5(v) \Leftarrow \overline{inf}.\ S_6(v)$
$S_6(v) \Leftarrow \overline{restart}.\ S_2(v)$

The duplication suppressor that filters output:

$DS(l,e) \Leftarrow finish.\ DS(0,0) + inf.\ DS(0,e)$
$+ pout\ u_l.$ **if** l>=e **then** $\overline{out}\ u_l.\ DS(l+1, e+1)$ **else** $DS(l+1, e)$

The complete fault tolerant system:

$FTS \Leftarrow (P_1|F_1|S_1|DS(0,0))\backslash\{in, out\}$

To prove: $FTS \approx PER_1$

Figure 2-1: SYSTEM I: PER has no state between inputs

process, use manual overrrides to patch up damage to the data base, and restart the process. A restarted process was reset either to its initial state, or to its last checkpoint if it had one. Its communications with other processes were then simulated from a terminal until it was brought up to date, and it was then linked back normally to other processes.

To translate this into our terms: Quick discovery and localisation of the

fault, followed by killing the faulty process, made the latter look like a fail-stop process. The patching up was necessary because this imitation was not perfect. The interactive update took care of suppressing duplicated output.

Another reason why the scheme worked so well in practice: Several of the processes behaved like PER, returning to an initial state after processing each message sent to them, and therefore needed no checkpointing. They also needed no log of input messages or updating! And even for processes with an internal state, it turned out that many of the messages sent to them were merely queries that did not alter their state. Such messages were not recorded in the log.

The same recovery mechanism also permitted continued development even when the system was online. The older versions were merely aborted and newer ones restarted in their place.

2.2 Outline of proof

We shall use the fact that

$$(A|B)\backslash\alpha = (A\backslash\alpha)|(B\backslash\alpha) \text{ if } \alpha,\overline{\alpha} \notin L(A) \cup \overline{L(B)}$$

This allows us to move restriction inwards, and get

$$\begin{aligned}
&(P_1|F_1|S_1|DS(0,0))\upharpoonright\{in, out\}\\
&= (P_1|F_1|S_1|DS(0,0))\backslash\{pin, pout, pfinish, finish, f, pinf, inf, restart\}\\
&= ((P_1|F_1|S_1)\backslash\{pin, pfinish, pinf, f, restart\} \mid DS(0,0))\backslash\{pout, finish, inf\}\\
&= (((P_1|S_1)\backslash\{pin, pfinish, restart\}|F_1)\backslash\{f, pinf\}|DS(0,0))\backslash\{pout, pfinish, inf\}\\
&= ((P_1\|S_1)\|F_1)\|DS(0,0)
\end{aligned}$$

We want to prove that this is equivalent to PER_1. We do this by guessing a specification PSF for $(P_1\|S_1)\|F_1$ and proving

$$PSF\|DS(0,0) \approx PER_1.$$

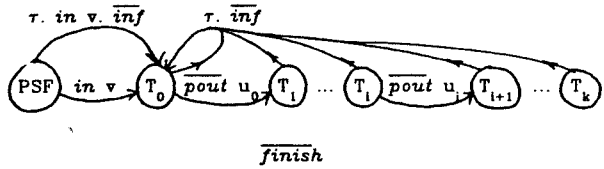
Next we guess a specification PS for $P_1\|S_1$ and show

$$PS\|F_1 \approx PSF.$$

Finally, we show

$$P_1\|S_1 \approx PS.$$

Note that in these last two equivalences, we do not need to show $\approx^c$, since in the context of "|", observational equivalence is substitutive.



$PSF \Leftarrow in\ v.\ T_0 + \tau.\ in\ v.\ \overline{inf}.\ T_0$

$T_i \Leftarrow \overline{pout}\ u_i.\ T_{i+1} + \tau.\ TF \quad$ for i = 0..k-1

$T_k \Leftarrow \overline{finish}.\ PSF + \tau.\ TF$

$TF \Leftarrow \overline{inf}.\ T_0$

Figure 2-2: Specification of PSF

The definitions of DS and PER are reproduced below for reference

$DS(l,e) \Leftarrow finish.\ DS(0,0)$
$\quad + inf.\ DS(0,e)$
$\quad + pout\ u_l.$ **if** $l \geq e$ **then** $\overline{out}\ u_l.\ DS(l+1, e+1)$
$\quad$ **else** $DS(l+1, e)$

$PER_1 \Leftarrow in\ v.\ PER_{2.0}$

$PER_{2.i} \Leftarrow \overline{out}\ u_i.\ PER_{2.i+1} \quad$ for i = 0..k-1

$PER_{2.k} \Leftarrow PER_1$

To prove: $PER_1 \approx PSF \| DS(0,0)$

3. A Generalisation, and a Different Proof

3.1 Why another example?

We will begin afresh with a description of a new fault-free system, the "perfect process", and try to build a fault tolerant system equivalent to it. We have two aims: to generalise the perfect process a little from our first example, and to improve our proof. We will take a few steps away from our first toy example, and though we will still have only a very simple system, it will actually be quite general, as we shall see.

For the proof, we have several goals: to make it more readable and structured, and to make the intermediate steps more meaningful rather than have opaque symbolic manipulations that somehow suddenly produce the desired result. We try to encapsulate each (still rather large) stretch of such manipulations into a chunk that is simple, if tedious, and whose end results are more plausible. We are trying to create a proof with a larger ratio of interest to pure drudgery.

When we tried to build a Prolog system to do our first proof, we found that the bisimulation could be done easily enough, and it was only the notational manipulation (indexing for example) that required a proof checker of general mathematical sophistication. This situation continues with the present proof: the symbolic manipulation within each stage (though not in the transition between stages) is mechanical enough but still requires a system of general mathematical sophistication (indexing, case analysis, applying constraints, transforming parameters etc.). The CCS specific requirements of the system are as before.

3.2 Formulation

We start by trying to write down the most general CCS agent we can have as our perfect process, $P^{\dagger}$. Then we will try to produce a fault tolerant system FTS equivalent to it, using stable storage as the technique to recover from faults. As before, FTS mimics the actions of $P^{\dagger}$ until a fault occurs. At this point it backs up to the initial state, and recomputes until it reaches the state at which it failed. In order to write such an FTS, we need $P^{\dagger}$ to be deterministic. If $P^{\dagger}$ were non-deterministic, we could not guarantee that FTS will produce the same behaviour each time we back up to an earlier point in the computation.

Consider an arbitrary sum of guards:

$$P^{\dagger}(s) \Leftarrow \sum_{i \in I(s)} \alpha_i x.\ P^{\dagger}(f(s,i,x))$$

which at each stage offers a choice of actions, the set offered being a function of its state. It then evolves to the next state, which is a function of its current state and the action taken.

We do not consider $\alpha_i = \tau$, because we are assuming that $P^{\dagger}$ is deterministic. If

$$P^{\dagger}(s_1) \xrightarrow{\alpha} P^{\dagger}(s_2) \quad \text{and} \quad P^{\dagger}(s_2) \xrightarrow{\tau} P^{\dagger}(s_3)$$

we can take $P^{\dagger}$ from s_1 to s_3 in a single step by merely rewriting the function f.

We can also impose another restriction. If there is a choice of output actions, the process communicating with $P^{\dagger}$ has to choose, and we can regard this choice as input, because its decision constitutes information that flows in to $P^{\dagger}$. Thus we can work with

$$P^{\dagger}(s) \Leftarrow \text{if } in(s) \text{ then } \sum_{i \in I_a} \alpha_i x.\ P^{\dagger}(f(s,i,x)) \text{ else } \overline{\beta}_{j(s)} h(s).\ P^{\dagger}(g(s))$$

As written, $P^{\dagger}$ offers either a choice of input actions, or a single output at each stage. However, we can achieve the effect of an agent A that offers a mixture of inputs and outputs as follows. Each state s of A is represented by two states s_1 and s_2 of $P^{\dagger}$, s_1 being a preliminary input state, in which the outside world can tell $P^{\dagger}$ what action it wishes A to perform in s. This will then be performed by $P^{\dagger}$ in s_2. Since the output action can be influenced by previous input, we can offer a protocol. For example, if the communicant wants a particular output $\overline{\beta}_j$ there might be a corresponding request input α_j. Thus to get $\overline{\beta}_j$, an external agent will have to do $\overline{\alpha}_j \beta_j$ rather than just β_j. So as long as $P^{\dagger}$ guarantees that all possible output actions have a corresponding request in each input sort I(s), it can effectively offer a mixture of inputs and outputs.

Why don't we have arbitrary numbers of agents like $P^{\dagger}$ composed with |? We have not allowed $P^{\dagger}$ to be composed of other agents in the form Q|R because in this case we expect to have to surround each of these agents with its own recovery mechanism, so that communication between Q and R can also be spied on.

Now for our fallible version P of $P^{\dagger}$. As long as it doesn't fail, it behaves like $P^{\dagger}$. Failure is modelled as before, as being caused by a demon F that communicates with P, and can occur at any point between α and β actions. Once P has failed, it has to go through a restart protocol, and will then relive its history starting from the initial state, s_0.

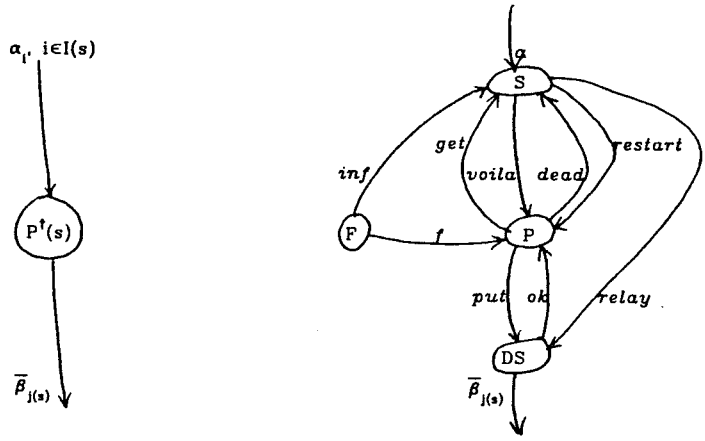


Figure 3-1: Pictures of the Perfect and the Fault Tolerant Systems

$P(s) \Leftarrow$ **if** in(s) **then** $(\overline{get}\ I(s).\ PI(s) + f.\ PF)$
else $(\overline{put}(j(s),h(s)).\ PO(g(s)) + f.\ PF)$
$PI(s) \Leftarrow voila(i,x).\ P(f(s,i,x)) + f.\ PF$ where i∈I(s)
$PO(s) \Leftarrow ok.\ P(s) + f.\ PF$
$PF \Leftarrow \overline{dead}.\ restart.\ P(s_0)$

$F_1 \Leftarrow \overline{f}.\ F_2$
$F_2 \Leftarrow \overline{inf}.\ F_1$

As we see from the specification above, P does not itself do α's and $\overline{\beta}$'s but makes requests in the form of $\overline{get}$'s and $\overline{put}$'s to infallible backup agents S and DS, described below. α's are actually done by S in response to a *get* from P, and the resulting value x is relayed to P by a $\overline{voila}$. Thus S spies on all input to P, and can keep a record of it by appending each input (port and value) to q. When P fails, S is informed by *inf*. It then restarts P, and feeds it back all its input, using the queue pointer d to count off the records. Recovery is complete, and *get*'s once again produce actual α's, after d has counted up to |q|.

$S(d,q) \Leftarrow get\ L.\ SI(d,q,L) + inf.\ SF(q)$
$SI(d,q,L) \Leftarrow$ **if** $d = |q|+1$ **then** $\sum_{i\in L}\alpha_i x.\ SG(d,\ q{:}\langle i,x\rangle)$
else $SG(d,\ q)$
$SG(d,q) \Leftarrow \overline{voila}(q_d.i,\ q_d.x).\ S(d+1,q) + inf.\ SF(q)$
$SF(q) \Leftarrow dead.\ \overline{relay}.\ \overline{restart}.\ S(1,q)$
In these equations, d is the index of the next element to be sent to P.

S shields the outside world from the effects of P's failure as far as α's are concerned. To do the same for $\overline{\beta}$'s, we need DS. Whenever P is restarted, it will generate again all the output corresponding to the input value. But some of this output might already have been relayed to the outside world before failure. This duplicated output is suppressed by DS.

Since output is deterministically computed or recomputed by P, DS does not need a queue, but only a pair of counters. The external counter e keeps track of *put*'s that have actually been relayed on to the outside as $\overline{\beta}$'s (and acknowledged by $\overline{ok}$). S relays *inf* to DS as $\overline{relay}$, informing it of failure. DS then suppresses the next e *put* requests from P because these have already been passed on once. The counter l keeps track of this, counting the messages generated during the rerun. DS starts relaying them to the outside world, via $\overline{\beta}$, only after P has caught up with history. This happens when the local counter equals the external counter e, which counts the number of messages relayed to the outside. Recovery is now complete.

$$\begin{aligned} DS(l,e) \Leftarrow\ & put(j,x).\ \textbf{if } l \geq e \textbf{ then } \overline{\beta}_j x.\ DO(l+1,\ e+1) \\ & \qquad\qquad \textbf{else } DO(l+1,\ e) \\ & + relay.\ DS(0,e) \\ DO(l,e) \Leftarrow\ & \overline{ok}.\ DS(l,e) + relay.\ DS(0,e) \end{aligned}$$

It is worth noticing the fairly complex restart protocol. The *dead* allows us to cater for the case where F does $\overline{inf}$ before doing an $\overline{f}$. This would happen for example, if we were using *inf* to model a timeout. Timeouts are not infallible as a means of detecting failure, and P can cause S to time out by merely being slow. Thus an *inf* can occur before an *f*. We can program this by starting F off in the state F_2 rather than in F_1. This symmetry is not just a luxury - we will need it in our proof, as we will see.

Without the *dead*, S would go ahead and reset DS while P is perhaps still computing. This would mess up our counters l and e. In our first example, we solved this by doing the $\overline{restart}$ before we did the $\overline{relay}$. That will no longer work with our more general P because now P might start recomputing and produce outputs right away before DS has reset itself. (This problem arises only if s_0 is an output state - otherwise it has to wait for S to accept a *get* and that would provide the necessary synchronisation).

The specifications are summarised in Fig. 3-2.

The sorts of S and DS depend on those of $P^{\dagger}$, but their structure is otherwise independent of it. S maintains a queue of inputs which can be retransmitted to P if it fails. DS can get by with counters because P will recompute the outputs.

The perfect process is defined by

$P^{\dagger}(s) \Leftarrow$ **if** in(s) **then** $\sum_{i \in I_s} \alpha_i x.\ P^{\dagger}(f(s,i,x))$ **else** $\overline{\beta}_{j(s)} h(s).\ P^{\dagger}(g(s))$

The fallible version of the perfect process:

$P(s) \Leftarrow$ **if** in(s) **then** $(\overline{get}\ I(s).\ PI(s) + f.\ PF)$

else $(\overline{put}(j(s),h(s)).\ PO(g(s)) + f.\ PF)$

$PI(s) \Leftarrow voila(i,x).\ P(f(s,i,x)) + f.\ PF$ where $i \in I(s)$

$PO(s) \Leftarrow ok.\ P(s) + f.\ PF$

$PF \Leftarrow \overline{dead}.\ restart.\ P(s_0)$

The demon agent that causes failure:

$F_1 \Leftarrow \overline{f}.\ F_2$

$F_2 \Leftarrow \overline{inf}.\ F_1$

The spy that eavesdrops on all inputs to P, and saves them for recovery:

$S(d,q) \Leftarrow get\ L.\ SI(d,q,L) + inf.\ SF(q)$

$SI(d,q,L) \Leftarrow$ **if** $d = |q|+1$ **then** $\sum_{i \in L} \alpha_i x.\ SG(d,\ q{:}\langle i,x\rangle)$

else SG(d, q)

$SG(d,q) \Leftarrow \overline{voila}(q_d{\cdot}i,\ q_d{\cdot}x).\ S(d+1,q) + inf.\ SF(q)$

$SF(q) \Leftarrow dead.\ \overline{relay}.\ \overline{restart}.\ S(1,q)$

In S(d,q) and its derivatives, d points to the next element to be sent to P.

The duplication suppressor ensures that only new output is transmitted on $\overline{\beta}$:

$DS(l,e) \Leftarrow put(j,x).$ **if** $l \geq e$ **then** $\overline{\beta}_j x.\ DO(l+1,\ e+1)$

else DO(l+1, e)

$+\ relay.\ DS(0,e)$

$DO(l,e) \Leftarrow \overline{ok}.\ DS(l,e) + relay.\ DS(0,e)$

Therefore the fault tolerant system is defined by:

$FTS(s,d,q,l,e) \Leftarrow (P(s)|F_1|S(d,q)|DS(l,e))\backslash\{\alpha,\beta\}$

where we have used α as a shorthand for $\cup I(s)$ and β for $\cup j_s$.

Note that there are constraints on the parameters of FTS:

If we start from $FTS(s_1,0,\varepsilon,0,0)$ we have

$l \leq e$

$1 \leq d \leq |q|+1$

and more complicated connections due to the history of the system.

To prove: $FTS(s_1,1,\varepsilon,0,0) \approx P^{\dagger}(s_1)$

Figure 3-2: SYSTEM II

Notice that the *"get-voila"* and *"put-ok"* pattern makes S and DS more symmetric than in our earlier example. The decision to have S discovering the failure via *inf* is arbitrary. We could just as well have had DS detect the failure and relay the information on to S.

3.3 Outline of the proof

We want to prove that FTS is equivalent to $P^\dagger$. In giving this informal outline of the proof, our intention is only to provide a rough idea of what is to come. Many of the remarks below might be difficult to fully appreciate until they appear later in formal garb, and the reader might find that this section makes more sense after seeing the whole proof.

We have to prove that

$$FTS(s_1,1,\varepsilon,0,0) \approx P^\dagger(s_1)$$

In fact, it turns out that we will be able to prove a more general result:

$$FTS(s,d,q,l,e) \approx P^\dagger(s)$$

with some conditions on the parameters. These conditions are satisfied by $FTS(s_1,1,\varepsilon,0,0)$.

As before, we move restriction inwards to get

$$\begin{aligned}&FTS(s,d,q,l,e)\\&=(P(s)|F_1|S(d,q)|DS(l,e))\upharpoonright(\alpha,\beta)\\&=(((P(s)|S(d,q))\backslash\{get,voila,dead,restart\}|F_1)\backslash\{f,inf\}|DS(l,e))\backslash\{put,ok,relay\}\end{aligned}$$

We clarify a bit of usage: we will sometimes mention agents without giving their parameters. Thus "P" will refer to the agent defined by the four lines starting with "P(s) $\Leftarrow$" in Fig. 3-2. We will then refer to P, PI, P0 and PF as "states of P".

To prove FTS(s,d,q,l,e) equivalent to $P^\dagger(s)$, we would like to proceed by stages, decomposing each agent bit by bit. But we have to do some preparation for this, because $P^\dagger$ is a very concise specification, and the very form of the equivalence shows that the extra parameters d,q,l,e in FTS are redundant at this level of description.

So we begin by expanding $P^\dagger(s)$, putting more and more redundant information into it, ensuring that each successive stage is equivalent to the previous one. When we have as much information in the specification as in the implementation FTS, we can start splitting off first DS and then F from each of

Summary of the proof

Proposition 3.4.2: $P^{\dagger}(s) \approx G(s)$

Proposition 3.5.1: $G(s) \approx Y(p{:}s)$

Proposition 3.6.2: $Y(p) \approx Z(p)$

Proposition 3.9.2: $Z(p) \approx J(q,c)$ where $p=u(q,c)$

Proposition 3.10.1: $J(q,c) \approx B(q,c)$

Proposition 3.11.2: $B(q,c) \approx C(q,c) \| DS(e,e)$ where $e=o(q,c)$

Proposition 3.12.2: $C(q,c) \approx D(q,s)$ where $s=r(q,c)$

Proposition 3.13.1: $D(q,s) \approx E(q,s) \| F_1$

Proposition 3.13.2: $E(q,s) \approx P(s) \| S(|q|+1,q)$

which proves that $P^{\dagger}(s) \approx DS(e,e) \| F_1 \| P(s) \| S(|q|+1,q)$
where the parameters are all functions of the underlying history (q,c).

Figure 3-3: Summary of the proof

them. After every such decomposition, we have to show that the remaining pieces are equivalent.

A word of apology for the names of the many agents we will define. We have not attempted to give meaningful names to every one of them (and some of them have several states), as this would have given us a host of names, and these names would have to be quite long in order to capture the slight differences between the agents. Instead, we have just used letters from all parts of the alphabet. The part of the alphabet a name comes from has historical significance with respect to the many versions the proof has been through, but can be taken by the reader to be entirely arbitrary.

In what follows, the reader might find it helpful to refer to the summary of the proof given in Fig. 3-3.

We first guess a specification G(s) for FTS(s,d,q,l,e), and prove $G(s) \approx P^{\dagger}(s)$ (Proposition 3.4.2). G naturally has only α and $\overline{\beta}$ as externally visible actions, but it records failure and recovery as silent actions. Thus G(s) and $P^{\dagger}(s)$ are only observationally equivalent, not strongly equivalent. In particular, we use the fact that observational equivalence permits one of the agents (G) to have infinite chatter even if the other doesn't. The infinite chatter in G can come about if failures occur repeatedly, and don't let it get on with the computation.

Our next step is to convert G(s) into a specification Y(p) where p is a sequence of states, the last of which is the current state s. Thus p stores the whole state sequence of FTS. We show that $G(s) \approx Y(p)$ (Proposition 3.5.1). This is

our first data transformation, and is a strong bisimulation.

We then expand Y(p) into Z(p), which shows the recovery process in more detail. Instead of a single silent action, it consists of a series of them: upon failure, Z sets a pointer to the start of the stored state sequence, and counts off the states till it reaches the end - the current state, at which failure occurred. Failure can also occur during this process. From the outside, all recovery states look exactly the same - they are all equivalent to the state just before failure occurred. Proposition 3.6.1, which proves this, is a weak bisimulation. Proposition 3.6.2, which proves $Y(p) \approx Z(p)$, follows trivially by algebra.

At this point, we have seen the essence of the recovery process (at this level of description anyway), and are now ready for another data transformation. We need the new form of data in order to link up with later stages in the proof.

We go back to G(s) and convert it into another one, H, which maintains a queue q of inputs (like S) and a count c of outputs, and uses these to calculate its current state. We start with G rather than Z because this transformation is a little complicated. H is for explanatory purposes only, and does not occur in the string of equivalences that constitute the whole proof. The relation between (q,c) and s is captured by a function r(q,c), defined inductively. H(q,c) does not describe recovery in any more detail than G(s), and the equivalence of the two is proved by strong bisimulation as Proposition 3.8.1. The bisimulation essentially shows that the relation r between the parameters of G and H is preserved as the two agents evolve. There are also some consistency constraints on the parameters of H, and we show that these are preserved if we start from a consistent state, $H(\varepsilon,0)$.

We then expand H(q,c) into a specification J(q,c) which corresponds to Z(p). The equivalence between J and Z is proved in Proposition 3.9.2, in a similar way to Proposition 3.8.1: a function u relates (q,c) to p, and a strong bisimulation shows that the relation is preserved as J and Z evolve.

We should mention here that an alternative chain of equivalences to do the whole proof would go from $P^\dagger$ to G to H to J. This would have the advantage that we would not need Y and Z at all. However going from H to J turns out to be slightly messy technically, and we prefer the longer but somewhat cleaner route.

Our final specification for the whole FTS, B, shows in more detail all the points at which failure can occur. The equivalence of B(q,c) and J(q,c) is trivial, and is shown in Proposition 3.10.1.

At this point, we have put in as much detail as we want to in our description of the whole FTS. We now separate it out into DS and C: we guess a specification

C(q,c) for FTS with DS removed, and prove that $C(q,c)\|DS(e,e) \approx B(q,c)$ (Proposition 3.11.2). This turns out to be just algebra, but made possible only because we have the correct level of detail in B, and because of some relations between the queue of inputs and the history of the system. These are the consistency constraints that tie up the behaviours of C and DS. It is clear from the form of the equivalence in Proposition 3.11.2 that we need a relation between (q,c) and e, the external counter of DS. We define, inductively, a function o(q,c) which gives the total number of outputs FTS generates in going from s_0 to r(q,c), and it will turn out that e=o(q,c).

We are now through with the need for storing the counters, which relate to DS, and can simplify the data representation. We go back to storing the current state, rather than recalculating it every time from history. The new specification, D(q,s), still needs the queue of inputs for recovery. Proposition 3.12.2 proves that $C(q,c)\approx D(q,s)$. This is essentially a proof of correctness of data transformation, and is a strong bisimulation.

Before doing Proposition 3.12.2, we have to modify C a little, to take into account the fact that C and D have different ideas about when recovery is complete. For D, this is when all input has been fed back to P. If there is a string of output states after this, it cannot tell when it has passed the last recomputed output states and arrived at new states that have not occurred before. C can, because it marks this last state. Proposition 3.12.1 proves that the modification to C preserves its behaviour.

We can now pull F out of D(q,s), giving us (we guess) E(q,s). Proposition 3.13.1 proves that $E(q,s)\|F_1 \approx D(q,s)$. This is just algebra, made easy because F is so simple.

It only remains to show, as Proposition 3.13.2, that $E(q,s) \approx P(s)\|S(|q|+1,q)$. This is a strong bisimulation, rather long but very straightforward. The reason why we don't try to do it by algebra is that E retains the recursive structure we have had for our specifications right through, whereas this is not apparent in P‖S. The jump is easiest to show by bisimulation. Also, E has recovery states and ordinary states, like all the other specifications, but on the other side, only S makes the distinction. P does not know whether it is reliving history or seeing it for the first time.

4. Conclusions

We have found one simple way to model failure in CCS, and discovered some important aspects of fault recovery by stable storage, including the fact that the intuitively clear "equivalence" between FTS and $P^{\dagger}$ is in fact observational equivalence, rather a surprising result at first sight, because this is a very strong equivalence.

In fact, we have found that observational equivalence is particularly suited to describing the relation between the two. Foe example, the two systems are not equivalent under a testing equivalence because of the infinite chatter possible for the FTS. Testing equivalences are capable of distinguishing infinite chatter. Except for this one situation, observational equivalence is generally stronger than testing equivalences. Thus our example allows us to use a strong relation (it would be unsatisfactory to describe it in terms of generally weaker relations), which happens to be lax in just the one case that we need.

The proof is very long, and we have discovered that we can apply some standard mathematical techniques to help. We have not learnt anything new about CCS, but we have developed some new notation which, together with the structured decomposition used, makes it possible to do a proof that was previously quite unmanageable.

We have applied some of these techniques to a few other (much simpler) proofs. Perhaps the most important is the simplest, indexing (or parametrisation). Together with our layout and notation for bisimulation, this turns out to be quite powerful. We have aplied it to a simple system (the "man-hammer" example from [Milner 83b]) for which [Sanderson 82] gave the first proof. The bisimulation, generated by his algorithm, consists of some 40 pairs. We now have a bisimulation of 9 pairs, using indexing. Not only is the new proof shorter, the individual pairs are also more readable, and the indexing captures much of the symmetry in the example.

Our technique of guessing refinements of specifications step by step applies stepwise refinement to the proof rather than to the program. It is not obvious for example, how we can derive P and S from E, or how to derive DS from B, even if we regarded the rest of the proof as putting more and more detail into a program. In any case, this is a most useful technique. In particular, it shows that we can often isolate strong bisimulations out of a larger weak bisimulation.

We notice that correctness of data transformation (often appearing in the guise of the standard mathematical technique of changing parameters) can be proved by strong bisimulation. This is a modern version of ideas that were

discussed long ago, [Milner 72].

It has been remarked [Milner 83b] that bisimulation is the counterpart of the invariant for a loop. We have several examples of relations being preserved between the two agents in a bisimulation. Two other interesting features we have noticed are that we can combine proof of equivalence with proof of well definedness of one of the agents, and that an inductive definition of the relation between the agents goes well with bisimulation.

We have already mentioned our conclusion about proof checkers: it appears that a useful proof checker for CCS proofs is likely to be a general mathematician's assistant, with only a little CCS specific knowledge built into it.

An insufficiently sophisticated tool can actually be a hindrance. If we had succeeded in building a proof checker that confirmed all of our first proof, we might not have gone on to discover the rather more elegant techniques we use in the second. This is not a philosophy of salvation through unnecessary hardship. We are pointing out that we might have abandoned our proof to the realm of large mechanical manipulation, not realising that we could convert it to more understandable mathematics.

However, we should mention that a major problem with such large proofs is that of maintaining any degree of confidence in their correctness. What we present here in section 3 is in fact the fifth or sixth version of the proof of the second system, and the difference between one version and a later one is best described as "greater believability". We leave it to the reader to imagine what the earlier proofs looked like.

5. Future work

We would like to see the proof simplified further: we have surely not yet put it as simply as possible; it might even be that with a different framework we can get an elegant proof of only a few pages.

We would like to extend the proof to handle checkpoints. There are also variations that might be of interest: can we separate out the components in a different order, for example, and would that yield a simpler proof?

It would be nice to have a few more such examples, both in order to develop the proof techniques further, perhaps even to discover a few general facts, and in order to have a broader base of examples to study fault tolerance. But the effort required to develop the proofs, and the comparatively meager returns, make it unlikely that we will have a sizeable collection in the near future. It is likely to be more immediately profitable to study other formulations of failure and fault tolerance applied to the same problem in the hope of exploiting our understanding of it and possibly even similarities in the proof.

6. Acknowledgements

The use of "we" rather than "I" throughout this paper was not a stylistic choice. Many of the ideas here are due to Robin Milner, my supervisor, as is any mathematical elegance the work may have. I thank him for his guidance, encouragement and constant support.

I also thank Roland Backhouse for a very detailed discussion of an earlier draft, which tied up many untidy loose ends. It also led me to the history representation that I now use instead of the "fat queues" I was using earlier.

REFERENCES

[Anderson and Lee 81]
T.Anderson and P.A.Lee.
Fault Tolerance: Principles and Practice.
Prentice Hall, 1981.

[Backhouse 83] R.Backhouse.
Specification and Proof of a Regular Language Recogniser in CCS.
Technical Report CSR-130-83, University of Edinburgh, Dept. of Computer Science, March, 1983.

[de Nicola and Hennessy 83]
R. de Nicola and M.C.B.Hennessy.
Testing Equivalences for Processes.
In *ICALP* 83. Springer Verlag LNCS 154, 1983.

[Merlin and Randell 77]
P.M.Merlin and B.Randell.
Consistent State Restoration in Distributed Systems.
Technical Report 113, University of Newcastle upon Tyne, October, 1977.

[Milner 72] Robin Milner.
An Algebraic Definition of Simulation Between Programs.
In *Proceedings of 2nd. IJCAI.* IJCAI, 1972.

[Milner 80] Robin Milner.
A Calculus of Communicating Systems.
Springer Verlag LNCS 92, 1980.

[Milner 83a] Robin Milner.
Calculi for Synchrony and Aynchrony.
Theoretical Computer Science 25(3):267-310, 1983.

[Milner 83b] Robin Milner.
Lecture Notes on CCS.
1983.
Unpublished notes, University of Edinburgh, Dept. of Computer Science.

[Mitchell 83] K.Mitchell.
Proof of the Alternating Bit Protocol in Synchronous CCS.
1983.
Unpublished note, University of Edinburgh, Dept. of Computer Science.

[Prasad, Ramani and Abraham 82]
K.V.S.Prasad, S.Ramani and J.Abraham.
Design and Implementation of a Simple Message Switching System.
Technical Report, NCSDCT, Tata Institute of Fundamental Research, Bombay, 1982.

[Sanderson 82] M.Sanderson.
Proof Techniques for CCS.
PhD thesis, University of Edinburgh, Dept. of Computer Science, 1982.

[Schlichting and Schneider 81]
R.D.Schlichting and F.B.Schneider.
Fail-stop Processors: an Approach to Designing Fault-Tolerant Computing Systems.
Technical Report TR 81-479, Cornell University, November, 1981.