University of Edinburgh

Department of Computer Science

Testing Equivalences and Fully Abstract Models for Communicating Processes

by

Rocco De Nicola

THESIS

CST-36-85

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

November, 1985

Testing Equivalences and Fully Abstract Models for Communicating Processes

by

Rocco De Nicola

Degree: Doctor of Philosophy
University of Edinburgh
1985

To the memory of my grandparents

Rocco and Maria Giuseppa

Abstract

In this thesis, a new denotational, interleaving-based, model for processes is proposed and justified in terms of a simple operational semantics. Indeed we first propose a new operational semantics for processes and then show how this semantics naturally leads to a denotational model. Our operational approach is based on *labelled transition systems* and on an experimentation notion which allows to consider as identical processes which are behaviourally equivalent. The basic idea behind our equivalence consists in noting that experience of machines or processes is gained by interacting with them. Two processes are considered to be equivalent if they "pass" the same tests from a given set. When applied to nondeterministic processes, the experimentation notion leads to definitions of when a process "may pass" a test and when a process "must pass" a test and to various *Testing Equivalences*. These new notions of program behaviour lead to new domains for processes which are called *Representation Trees* and are based on particular kinds of trees. Every tree associated with a process contains information about the possible sequences of actions which the process may perform and the future of the process after every possible sequence of actions. The thesis studies *Testing Equivalences* and *Representation Trees*, their interconnections, their equational characterization, and their relationships with other proposed abstract models of parallelism.

We apply the general framework for testing processes to a particular language, Milner's *CCS*, and examine the substitutive relations generated by three *testing preorders*. We give three *sound and complete proof systems* for these preorders and show how the proof systems lead to three *fully abstract* denotational models for CCS, i.e. models in which processes are distinguished if and only if they are distingushed by the associated set of tests. These models are constructed in a very abstract way from the syntax of the language and from the equations of the proof systems, but we show that they are isomorphic to classes of Representation Trees.

We also study *Refusal Sets*, another denotational model proposed to define the semantics of *TCSP*, an abstract version of Hoare's CSP. We investigate the congruence induced on TCSP terms by Refusal Sets and show that some induced equalities do not correspond to any operational intuition about processes behaviour. Considerations on the reasons for the mismatch between operational intuitions and denotational semantics lead us to define a new domain for processes which is obtained by imposing boundedness constraints on refusal sets and which is again isomorphic to a class of representation trees. This new domain permits us to give a complete proof system and a fully abstract model for TCSP too.

Acknowlegments

I would like to thank, first of all, my supervisor Matthew Hennessy for his guidence and for providing technical feedbacks and giving precise suggestions whenever they were needed.

Many thanks also to Robin Milner, who acted as a supervisor during my first year at Edinburgh and pointed out problems which have had a great influence on the research which is at the basis of this thesis.

Many other people in Edinburgh, students, staff and visitors, contributed to the ideas discussed in this thesis and made may studies an exiciting experience. In particular I would like to thank Ilaria Castellani, Luca Cardelli, Gerardo Costa, Mark Millington, Kevin Mitchell, Don Sannella, Colin Stirling and Andrzej Tarlecki for many interesting discussions, for finding always the time for listening and reading my half baked ideas and for giving helpful suggestions. I have also benefitted from discussions and correspondence with Steve Brookes (Pittsburgh), Tony Hoare (Oxford) and Ugo Montanari (Pisa).

My stay in Edinburgh has been supported by a scholarship of the Italian Research Council (CNR) and by a scholarship of the University of Edinburgh. During my final year I have been employed by the Istituto di Elaborazione dell'Informazione in Pisa. Many thanks to them for granting me long leaves of absence to work in Edinburgh.

Last but not least, many thanks to Susanna, my companion, whose sympathetic understanding helped me during all the gestation and the final writing. She never let "The Thesis" oppress me.

Declaration

This thesis has been composed by myself and the work is my own, under the guidance of my supervisor Matthew Hennessy. Chapter 2 and some parts of Chapter 3 are essentially /DH84/, a paper produced in collaboration with M. Hennessy. Chapter 4 is a revised version of /DeN83/ and Chapter 5 is an extended and revised version of /DeN84/.

CONTENTS

0. Introduction 1

1. Behavioural Equivalences for Transition Systems 12
- 1.0 Introduction 12
- 1.1 Labelled Transition Systems 14
- 1.2 Strings Equivalence 18
- 1.3 Observational Equivalence and Bisimulation 19
- 1.4 Weak Equivalences 23
  - 1.4.1 Kennaway's Equivalence 24
  - 1.4.2 Darondeau's Equivalence 28
  - 1.4.3 Failures Equivalence 30
  - 1.4.4 Examples and Discussion 30
- 1.5 A Theory of Testing 32
- 1.6 Testing Transition Systems 38
  - 1.6.1 Testing Equivalences 38
  - 1.6.2 Alternative Characterizations 41
- 1.7 Alternative Forms of Testing 45
- 1.8 Comparisons 50

2. Proof Systems for CCS based on Testing 53
- 2.0 Introduction 53
- 2.1 CCS 55
- 2.2 Testing Equivalences for CCS 59
- 2.3 Alternative Characterizations 67
  - 2.3.1 New Tests 67
  - 2.3.2 Minimal Sets of Observers 71

2.4 Three Proof Systems for CCS 73
2.5 Detailed Proofs for Finitary CCS 84
2.5.1 Soudness 84
2.5.2 Normal Forms 89
2.5.3 Completeness 97
2.6 Extending the Results to Infinite CCS 101

3. Fully Abstract Models for CCS 108
3.0 Introduction 108
3.1 Algebraic Relations and Fully Abstract Models 111
3.2 Inequational Models for CCS 119
3.3 Representation Trees: "Concrete" Models for CCS 120

4. Models and Axioms for a Theory of CSP 136
4.0 Introduction 136
4.1 TCSP and Refusal Sets 138
4.2 A Complete Set of Axioms for a Subset of TCSP 143
4.2.1 The Axioms System 143
4.2.2 The Completeness Theorem 151
4.3 Bounded Refusal Sets: A New Domain for Processes 159
4.4 TCSP and Bounded Refusal Sets 167
4.5 A Complete Proof System for TCSP 171

5. Models and Operators for Nondeterministic Processes 181
5.0 Introduction 181
5.1 Preemptive Representation Trees 183
5.2 Operators for Nondeterminism 186
5.3 A comparison between CCS and TCSP 191

6. Conclusions 200

7. References 206

0. INTRODUCTION

The need for programming languages with primitives which make it possible to describe concurrent activities is generally acknowledged. This need is prompted by two main factors: the desire to take advantage of the advance in hardware design and manufacturing, multiprocessor machines are now commonplace, and the fact that many problems tackled by computer users are naturally formalized in term of independent agents which only occasionally synchronize or exchange messages with other agents. In the last fifteen years, many new programming languages with constructs which allow some form of parallel execution and communication have been proposed. Earlier approaches were mostly based on modelling communication via shared memory and the main problem was to find appropriate linguistic constructs to describe and control concurrent activities on shared resources. Examples are *semaphores* /Dij68/, *monitors* /Hoa74/ and *critical regions* /BrH78/. In the late seventies, language constructs for explicit communication were suggested (cf. /Hoa78/ and /Fel79/). Since they make it possible to express the logical dependencies between communicating systems more directly and to enforce a neater programming discipline, they have been the object of great interest. In fact these constructs, in particular the CSP handshake /Hoa78/, have influenced the definition of recent general-purpose programming languages such as ADA /Ich80/ and CHILL /BLW82/.

While all this activity has been taking place,more and more people have become aware of the importance of formal techniques to define the semantics of programming languages and to specify, analyse and prove correctness of systems. Research in this field has lead to a considerable growth of the understanding of the theory of sequential systems. A theory known

as *denotational semantics*, whose foundations have been laid by Scott and Strachey, was very successful in modelling many sequential programming languages and systems. It is mainly based on the idea that the only relevant aspect of a sequential system is the relation between its possible inputs and the corresponding outputs. This means that it is possible to define the semantics of systems as mathematical functions from the domain of input values to the domain of output values. This idea has paved the way to a satisfactory formal treatment of sequential systems, /Sco76, Sto77, Gor79/. The main focus of attention in this area now is on the best way of using these techniques for the specification of sequential programming languages and systems and for software development /BJ78, Mos83/.

It soon became apparent that the need for formal techniques to define the semantics of concurrent systems was even more stringent. In fact, concurrent systems can exhibit pathological behaviours, such as starvation or deadlock, which are not possible for sequential systems. Unfortunately the same techniques used for sequential systems cannot be used for concurrent ones. It is inadequate to use a function from the input domain to the output domain to denote the meaning of a concurrent system. The intermediate states through which a system passes before delivering its output cannot be ignored; they are important to determine how a system would influence the overall behaviour of any larger system which uses it as a subpart. Moreover since it is often useful to abstract from the relative speed of the subparts and/or from the scheduling policies adopted, in many cases it is not possible to have a single value as output but sets of possible values must be considered.

The example below, borrowed from /Lam83/, attempts to justify these ideas. Let us consider the following two program statements in which the angle brackets denote atomic (indivisible) actions:

A: ⟨x := x+1⟩

B: begin ⟨x := x+y+1⟩; ⟨x := x-y⟩ end

It is not difficult to be convinced that, whatever the initial values of x and y, A and B always have the same effect on them, i.e. x is incremented by 1 and y is left unchanged. However, consider the following statement where cobegin indicates that the two clauses are to be executed concurrently:

cobegin ⟨y := y-7 par C⟩ end

Substituting A for C, we obtain a statement which increases x by 1; substituting B for C we get a statement which increments x by either 1 or 8.

Once it is recognised that input-output functions are not sufficient to define a mathematical semantics for concurrent systems, there is no general agreement as to what class of mathematical entities is suitable to model their meaning. In fact, the proposed models do not even represent concurrency in the same way. While models like Petri Nets /Pet80/ or Event Structures /Win80, CFM83/ represent concurrent activities in terms of causal independence, other models, which want to stress that communication is the basic concept of parallel systems, simulate concurrency by nondeterministic interleaving of atomic actions, i.e. they describe the fact that a set of events may occur concurrently (independently of each other) by saying that they may occur in any order (**interleaving**). It still remains to be understood whether there is any loss in terms of analytical power when taking this approach to concurrent systems modelling.

Recently and mainly for reasons of simplicity, a large class of models based on interleaving has been proposed and has been proved useful to describe and analyse many interesting parallel systems. Unfortunately, even after the common choice to reduce concurrency to nondeterministic interleaving, there are still different choices of mathematical entities as domains. The heterogeneity arises mainly from the different aspects of systems one is willing to capture and this in turn depend on the use of the model one has in mind.

Below, we list a few models based on interleaving, which aim at a generalization of the denotational approach to concurrent systems:

Resumptions: These are particular kind of functions based on Powerdomains, a generalization of Scott construction to denote nondeterministic computations. *Powerdomains* /Plo76, Smy78/ can be regarded as the domain analogues of powersets with elements which represent the "sets" of different courses which a nondeterministic computation can follow. Resumptions, proposed by Plotkin /Plo76/ and based on powerdomains, have been used by Milne and Milner /MM79/ to define a mathematical model of concurrent computations together with a set of operations to construct processes. The model was later abandoned by its proposers who claimed that it induces unwanted identifications and that its abstract nature does not allow a complete understanding of the semantics obtained.

Traces: The Traces model, /Hoa81/, identifies a process with the set of its possible sequences of communications (its traces). It turns out to be suitable to investigate potential communications but insensitive to deadlock.

Streams: This model, /KM77/, is only defined for a subclass of parallel deterministic processes. It associates a function from the set of communications on the input channel to the set of communications on the output channel to each deterministic parallel process. This model is a neat extension of Scott's ideas but problems have arisen in extending it to nondeterministic systems.

Synchronization Trees: A synchronization tree, /Mil80/, is a tree whose nodes represent the states of a process and whose arcs are labelled to denote potential communications with the environment. This model is basic and concrete: Tree semantics arise naturally when concurrency is simulated by nondeterministic interleaving. Unfortunately Synchronization Trees (when used to model systems behaviour) over-discriminate and need to be factorized by certain equivalences /Mil80, Bro83a, DH84/.

Refusal Sets: In addition to traces, the model based on refusal sets proposed by Brookes, Hoare and Roscoe, /BHR84/, associates the set of communications a process is able to refuse after every trace to every process. This model has problems in handling divergent or under-specified processes /DeN83/. It will be discussed in detail later.

In the above descriptions, when claiming that a model is too abstract, or over-discriminates, or induces unwanted identifications, we are always referring to some kind of intuition about the operational behaviour of processes. Since the mathematical semantics of parallelism is not well assessed and its theory is not yet well understood, many people think that the only way to get a correct semantics is to formalize the operational intuition and check any other proposed theory against this formalization. Milner, /Mil83/, suggests that " .... operational semantics, since it can be set up with so few preconceptions, must be the touchstone for assessing mathematical models ....".

Indeed, the operational approach to programming language semantics was the first one to be proposed. Basically, it describes the meaning of a program by specifying a convenient abstract machine and by modelling the execution of the program on that machine /Lan64, Weg72/. Initially, this approach was not generally accepted because the abstract machines used were too concrete, too close to the actual machines, and the specifications were far too detailed. Lately there has been a renewed interest in this approach. Plotkin /Plo81/, borrowing such concepts as compositionality and abstract syntax from denotational semantics, has proposed a new technique for operational specifications, which is sufficiently abstract and formal. The semantics of a program is described in terms of a transition system whose transition relation is defined via a set of axioms. Such *Structured Operational Semantics* (SOS) is mathematically tractable and it becomes possible to reason about its relationship with denotational semantics, e.g. we can decide if the two semantics "agree".

Informally speaking, we should say that an operational semantics and a denotational semantics agree whenever they induce the same identifications on terms. Two well-formed program fragments are considered operationally equivalent if and only if, whenever they are

embedded in a context to form a program, they give rise to the same overall operational behaviour. On the other hand we have that also a denotational semantics induces a natural equivalence on program fragments. In general, the semantic function is not one-to-one: The same element of the semantic domain is associated to more than one program fragment. Two such fragments are considered equivalent if they are denoted by the same element of the domain. The correspondence between operational and denotational semantics may be precisely stated. Following Milner /Mil73/, we say that a denotational semantics is **consistent** (**adequate**) if the denotational equivalence implies the operational one and that it is **complete** if the converse holds. Finally we say that a denotational semantics is **fully abstract** with respect to an operational one if and only if it is consistent and complete

In this thesis we propose a new denotational, interleaving based, model for processes and justify it in terms of a simple operational semantics. In fact, we first propose a new operational semantics for processes and then show how this naturally leads to a denotational model for nondeterministic processes. This gives a new model of concurrency based on nondeterministic interleaving of actions. The operational approach we propose is based on SOS and on an experimentation notion which allows to consider as identical processes which are extensionally (behaviourally) equivalent, i.e. processes which can be interchanged in any program context without affecting the final result.

The basic idea behind the proposed extensional equivalence is very simple and certainly not new. We first read about it in /Moo56/, where it was used to define equivalence relations between deterministic machines. The idea consists in noting that one's view of a machine or of a process depends on one's experience of it, and this experience is gained either by using or by interacting with the machine or process. Also, the behaviour of programs or processes can be investigated by a series of tests. For example when considering sequential programs we can define as tests pairs of predicates about the input domain and the output domain. It is very easy to see that the input-output function of a program can be characterized by such tests. For more general programming languages, more general kinds of tests are

needed. Indeed the nature of the programming language should suggest the type of tests which are suitable to investigate the behaviour of programs in that language. For example, if the language contains real time constructs then the tests should take time into consideration; when the programs involved are nondeterministic it is not only important to know if a process responds favourably or unfavourably to a test but also whether the program responds consistently each time the test is performed.

Given a set of processes and a set of appropriate tests, two processes will be equivalent (with respect to this set of tests) if they pass exactly the same tests. In order to be able to take into account **divergent experiments**, i.e. experiments which do not give an answer within a finite amount of time and in order to be able to talk about the results of experiments on **partially specified processes**, we shall consider preorders among processes instead of equivalences between them. (A preorder $\lessdot$ generates an equivalence $\simeq$ in a natural way, $\simeq = (\lessdot \cap \lessdot^{-1})$. We shall introduce two different preorders: the first is formulated in terms of the ability to respond positively to a test, the second is formulated in terms of the inability to respond negatively to a test. In the first case, given two processes P and Q, P is "less than" Q in case given any test if P "may pass" it then also Q "may pass" it. In the second case P is "less than" Q if whenever P "must pass" (cannot fail) a test then also Q "must pass" that test. The natural equivalence is obtained by taking the equivalence associated with the conjunction of these two preorders; this is a third preorder. Summing up we will define three **testing preorders** on processes based on the following:

$p \sqsubseteq_3 q$ if for every experiment e p may satisfy e implies q may satisfy e

$p \sqsubseteq_2 q$ if for every experiment e p must satisfy e implies q must satisfy e

$p \sqsubseteq_1 q$ if $p \sqsubseteq_2 q$ and $p \sqsubseteq_3 q$

where p may satisfy e if there exists an interaction between the process p and the experimenter e which is "satisfactory"

p must satisfy e if every interaction between the process p and the experimenter e is "satisfactory"

All the three equivalences generated by the three preorders have a correspondence with intuitive ideas about equivalence of programs. Consider the three program fragments below, where loop ≡ while *true* do *sleep* and or denotes a nondeterministic choice construct:

A: x := x+1

B: x := x+1 or loop

C: loop

The equivalence generated by the may-based preorder will identify A and B and distinguish B and C. The equivalence generated by the must-based preorder will identify B and C and distinguish A and B. Finally, the equivalence based on both "may" and "must" will distinguish all three programs.

As mentioned above these new notions of program behaviour lead to new domains for processes. These new domains are particular kinds of trees (**Representation Trees**) which share many features with Synchronization Trees /Mil80/ and Refusal Sets /BHR84/, but overcome some of the defects of these two models.

The representation tree associated with a process will contain the following information:

1. The possible sequences of actions (communications), the process may perform.

2. The possible future of the process after every possible sequence of actions. The possible future is denoted by means of sets of sets of actions, (**Acceptance Sets**) associated to every possible sequence of actions the process may perform. In general, the future depends on internal nondeterministic choices, and is characterized by the set of communications the process must accept at that stage of its progress.

We will present three different classes of Representation Trees, each one corresponding to a particular testing preorder. These models are equipped with continuous operators

corresponding to the operations on processes introduced in /Mil80/ and /BHR84/. We will also give complete equational characterizations of the models we propose. These characterizations, apart from being a good starting point for mechanized proof systems, are excellent touchstones since they allow to determine all the identifications induced by the models. In fact, determining an equational characterization of the Refusal Sets model of /BHR/ in the same way as for Representations Trees, we will be able to detect some inadequacies of Refusal Sets for the definition of the semantics of partially specified processes.

This thesis is developed around the idea of operationally defined **Testing Equivalences** and around **Representation Trees**, the denotational counterparts of testing equivalences. It studies their interconnections, their equational characterization, and the relationships they have with other proposed abstract models of parallelism. Moreover, it shows how both Testing Equivalences and Representation Trees can be used to define the semantics of CCS /Mil80/ and TCSP /BHR84/, two languages which have played a fundamental role in the development of concurrency theory. Below we give a brief overview of its content.

In Chapter 1 we revise some of the equivalence notions proposed in the literature for various models of parallelism, /BHR84, Der82, Hoa81, Ken81, Mil80, Par81/, by adapting all of them to labelled transition systems (LTS's), /Kel76/. We then formalize the intuitive notions of **testing preorders** by setting up a general framework within which experiments and the tabulation of possible results can be discussed. We show how LTS's can be immersed in this general setting and how, by changing the tabulations of the possible results and by slightly changing the view of what it means for a process to pass a test, we can obtain different preorders on transition systems. Finally, by using alternative characterizations of testing equivalences which are independent of the notion of a test, we can relate them to the other equivalences.

In Chapter 2 the general framework for testing processes is applied to a particular language, Milner's CCS /Mil80/. This is a primitive language to describe communicating

processes but has the advantage of having a simple and well-defined operational semantics. We take as our set of tests those tests which can be described in CCS and use its parallel combinator to perform our experiments. We then examine the substitutive relations generated by three testing preorders. (Two processes are substitutively related if they are related in every context). We give three **sound and complete proof systems** for these relations. Each of these systems consists essentially of a set of axioms to manipulate process expressions and a form of ω-induction.

In Chapter 3, after recalling some notions and some known results of domain theory and algebraic semantics /TWW78, Gue81, Sto77/, we show how the complete proof systems of the previous chapter can be used to obtain naturally three **fully abstract** denotational models for CCS, i.e. models in which processes are distinguished if and only if they are distingushed by the associated set of tests. These models are constructed in a very abstract way from the syntax of the language and from the equations of the complete proof systems. However, in the final section of the chapter we introduce the **Representation Tree** models and show that the previous abstract models are isomorphic to them. The isomorphism is proved by stressing the similarities between the representation trees and the normal forms for CCS used in the completeness proofs of the previous chapter.

In Chapter 4 we study **Refusal Sets**, the model of parallelism proposed by Brookes, Hoare and Roscoe /BHR84/ to define the semantics of an abstract version (**TCSP**) of Hoare's CSP /Hoa78/. Using the techniques developed in Chapter 2, we investigate the congruence induced on TCSP terms by the denotational semantics based on refusal sets. We give a complete set of axioms for a finitary subset of the calculus and show that some derivable axioms do not correspond to any operational intuition about the behaviour of processes. Considerations on the reasons for the mismatch between operational intuitions and denotational semantics lead us to define a new domain for processes which is obtained by imposing boundedness constraints on refusal sets (**Bounded Refusal Sets**). This new domain allows us to obtain a **complete proof system for TCSP** by working in the same way as for the previous chapter.

In Chapter 5 we present yet another domain for communicating processes. This is an extension of Representation Trees, which allows a better treatment of internal (invisible) moves of processes (**Preemptive Representation Trees**). The new domain is also equipped with a set of continuous operators based on those defined in Chapters 2 and 4 for CCS and TCSP, respectively. This allows us to study the relationships of the new domain with two of the models for CCS and TCSP (Representation Trees and Bounded Refusal Sets) discussed in the previous chapters. In fact, we prove that these models are just submodels of the domain of Preemptive Representation Trees. All these results allow us to compare CCS and TCSP with particular reference to their expressive power and to study the relationships between the various operators for nondeterminism which they introduce.

The final section contains some concluding remarks together with suggestions for future research.

1. BEHAVIOURAL EQUIVALENCES FOR TRANSITION SYSTEMS

1.0. Introduction

In many cases, it is useful to have theories for establishing whether two systems are equivalent or whether a system is a satisfactory "approximation" of another. If the same formalism is used to describe what is required of a system (**its specification**) and how it can actually be built (**its implementation**) then it is possible to use theories based on equivalences or approximations to prove that a particular implementation is correct with respect to a given specification. If a step-wise development method is used then it is very useful to be able to substitute large specifications with equivalent concise ones. In general it is useful to be able to interchange subsystems proved behaviourally equivalent, in the sense that one subsystem may replace another as part of a larger system without affecting the behaviour of the overall system.

Roughly speaking, we say that a system S_1 approximates (is equivalent to) a system S_2 whenever "some" aspects of the behaviour of the two systems are compatible. The kind of equivalences, or approximations involved depends very heavily on how the systems under consideration will be used. In fact, the way a system is used determines the behavioural aspects which must be taken into account and those which can be ignored. Because of this, it is important to know for every equivalence the properties of systems it preserves.

Not surprisingly, many different theories of equivalences have been proposed in the literature for models which are intended to be used to describe and reason about concurrent or nondeterministic systems. This is mainly due to the large number of properties which may be relevant to the analysis of systems. Almost all the equivalences are based on the idea of considering two systems as equivalent whenever no external observation can distinguish them. But there is still disagreement on what are "reasonable" observations and how their outcomes can be used to distinguish or identify systems. The major proposals have been made for several CCS-like languages, but they can be easily extended to other formalisms. In the following we will present and compare some of the equivalences defined in the literature and propose six new ones, each of which aims at capturing slightly different aspects of systems behaviour. These equivalences will also be related to those discussed previously.

In order to be able to study their interrelations, we will adapt all the equivalences presented to **Labelled Transition Systems** /Kel76, Plo81/. In the presentation of each equivalence, we will stress the aspects of system behaviours which are ignored and the identifications which are forced. We will show that various equivalences, defined in different ways and following different intuitions about systems behaviour /Dar82, BHR84, DH84, Ken81/, turn out to be the same or to differ only in minor detail for a large class of transition systems. A similar comparison has been attempted in /BR83/ but this paper considers a different class of equivalences and the stress is more on their logical implications than on their operational significance.

The rest of this chapter is organized as follows. In the first section we introduce Labelled Transition Systems and fix notations which will be used throughout the rest of the thesis. In the second and the third sections, we present respectively **String equivalences** /Hoa81/ and various **Observational equivalences**/Mil80, Mil84, HM85/. In the fourth section, we discuss **Kennaway's equivalence** /Ken81/ **Darondeau's equivalence** and **Failure equivalence** / BHR84/ and relate them to the equivalence generated by a new preorder on transition systems which paves the way to the study of their relationships. In the fifth

section, we introduce our own theory of systems equivalence based on testings and propose a general setting for defining **Testing equivalences** for processes /DH84/. The sixth and the seventh sections are dedicated to applying the general setting (or modifications thereof) of section 1.5 to Labelled Transition Systems. The final section is dedicated to a brief comparison of all the equivalences discussed.

1.1. Labelled Transition Systems

Since their appearance Keller's **Transition Systems** /Kel76/ have proved to be a model which to a certain extent underlies many proposed models of parallelism. We will present a particular class of Transition Systems with a particular emphasis on the methodologies for formal verification it supports. In particular, we will discuss methods to reduce systems to simpler ones and to prove the equivalence of two given systems. Transition Systems are an abstract relational model based on two primitive notions, namely those of state and transition. Given any other model for which it is possible to define a notion of global state and a notion of indivisible action causing a state transition we can define for each object of the model a corresponding transition system. This correspondence determines an **interleaving semantics** for the model and many systems properties can be studied purely in terms of Transition Systems. In this way this highly abstract conceptual model constitutes a significant part of most models of parallelism in which many of their properties can be formulated.

We will consider a particular class of nondeterministic transition systems which can be used to model systems controllable through interactions with a surrounding environment, but also capable of performing internal or hidden actions which cannot be influenced or even seen by any external agent. This model is named Labelled Transition Systems (**LTS**) and is a slight

modification of the model defined by Keller in /Kel78/.

Definition 1.1.1 A **Labelled Transition System** is a quadruple $(Q, A, -\mu\rightarrow, q_0)$ where Q is a countable set of states (p, p', q, q', ...), A is a countable set of elementary actions (a, b, c, ...), $-\mu\rightarrow$, $(\mu \in A \cup \{\tau\})$ is a set of binary relations on Q and $q_0 \in Q$ is the initial state. □

In this definition each of the relations $-a\rightarrow$ describes the effect of the execution of the elementary action a and, if $q, q' \in Q$, then $q-a\rightarrow q'$ indicates that, by performing action "a", a system, when in state q, can reach state q'. After Milner /Mil80/ the special simbol τ is used to denote internal actions and $q-\tau\rightarrow q'$ indicates that a system in state q can perform a silent move to state q'.

A transition system can obviously be "unrolled" into a tree. The initial state is the root and the transition relation is represented by the arcs labelled with elements from $A \cup \{\tau\}$; the various nodes will stand for other states. In the following all the examples will be expressed in terms of trees.

Very often in this chapter and through the rest of the thesis it will be necessary to abstract from internal actions, to consider sequences of visible actions, to say that a system may perform a particular action, etc.. Below we establish some notations for these notions which will help in shortening formulas.

- A will be used to denote the set of visible actions; a, b, c... will be used to denote its elements.
- A^* will denote strings over A; s, s', s_1, ...will be used to denote its elements, ε will be used to denote the empty string;
- A_τ will denote $A \cup \{\tau\}$, i.e. the set of visible actions extended with the distinct invisible action τ; this set will be ranged over by μ_1, μ_2, μ_3, ...

If p, q, p_1, p_2 .... are states of a transition system then

$p \dashrightarrow q$ will stand for $p - \tau \rightarrow q$;

$p - \mu_1 \mu_2 .... \mu_n \rightarrow q$ will be used to abbreviate there exists p_i with $0 < i < n$ such that
$p = p_0 - \mu_1 \rightarrow p_1 - \mu_2 \rightarrow ..... \rightarrow p_{n-1} - \mu_n \rightarrow p_n = q$;

$p - \mu_1 \mu_2 .... \mu_n \rightarrow$ will abbreviate there exists q such that $p - \mu_1 \mu_2 .... \mu_n \rightarrow q$;

$p - \mu_1 \mu_2 .... \mu_n \not\rightarrow$ will abbreviate <u>not</u> exists q such that $p - \mu_1 \mu_2 .... \mu_n \rightarrow q$;

$p = \Rightarrow q$ or $p = \varepsilon \Rightarrow q$ will stand for $p - \tau^n \rightarrow q$ with $n \geq 0$;

$p = a \Rightarrow q$ will stand for there exists p_1 and p_2 such that $p = \Rightarrow p_1 = a \Rightarrow p_2 = \Rightarrow q$;

$p = a_1 a_2 .... a_n \Rightarrow q$ will abbreviate there exist p_i $0 < i < n$ such that
$p = p_0 = a_1 \Rightarrow p_1 = a_2 \Rightarrow p_3$ ..... $\Rightarrow p_{n-1} = a_n \Rightarrow p_n = q$;

$p = s \Rightarrow$ will stand for there exists q such that $p = s \Rightarrow q$;

$p = s \not\Rightarrow$ will stand for <u>not</u> exists q such that $p = s \Rightarrow q$.

POW(A) will be used to denote the set of subsets of A

FPOW(A) will denote the set of finite subsets of A.

In the following it will also be useful to be able to talk about immediate moves from a particular state and about the possibility for a system to perform an infinite number of internal moves without ever performing a visible action. These and other interesting properties of Transition Systems are captured by the following definitions.

<u>Definition 1.1.2</u> If $(Q, A, -\mu\rightarrow, q_0)$ is an LTS and $q \in Q$ we have

i. $Init(q) = \{ a \in A \mid q = a \Rightarrow \}$

ii. $Traces(q) = \{ s \in A^* \mid q = s \Rightarrow \}$ □

<u>Definition 1.1.3</u> $\Downarrow$ is the least predicate (used in postfix form) on states of a transition system T which satisfies: $(\forall p', p - \tau \rightarrow p'$ implies $p' \Downarrow)$ implies $p \Downarrow$. □

It is easy to see that $p \Downarrow$ iff p cannot perform the infinite sequence $p - \tau \rightarrow p_1 - \tau \rightarrow p_2 - \tau \rightarrow ....$

We will denote its converse by $\Uparrow$, i.e. $p \Uparrow$ (read p diverges) if not $p \Downarrow$ (read p converges).

Two important subsets of LTS are the strongly convergent and the image finite ones.

<u>Definition 1.1.4</u> A transition system $T = (Q, A, -\mu\rightarrow, q_0)$ is **strongly convergent** if and only if for all $s \in A^*$ $q_0 = s \Rightarrow q$ implies $q \Downarrow$. □

<u>Definition 1.1.5</u> A relation R over states of a transition system T is **image finite** if for each state q of T the set $\{q' \mid q \, R \, q'\}$ is finite. □

The induced transition systems whose binary relation is $=s\Rightarrow$ are particularly interesting since they allow the analysis of aspects of systems which can be inferred if only their externally visible actions are taken into account. Many of the proposed approaches to systems simulation or systems reduction are based on, or can be reduced to, ignoring particular actions or particular sets of actions which for one reason or another have to be (can be) considered internal. In fact, all the equivalences we discuss will take this possibility into account. For reasons of simplicity, we will only consider labelled transition systems whose relation $-\mu\rightarrow$ is **image finite**. Please note that the induced relation $=s\Rightarrow$ does not necessarily have this property.

The rest of the chapter will be dedicated to defining and discussing various behavioural equivalences for Labelled Transition Systems.

1.2. Strings Equivalence

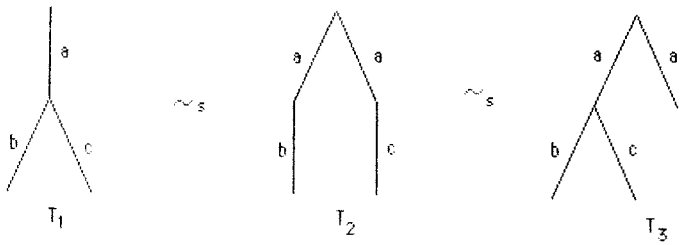
A natural proposal for systems equivalence is to consider as equivalent two systems which can perform exactly the same sequences of **visible** actions /Hoe81/. In this way, we can abstract from the internal (invisible) actions of a system.

Definition 1.2.1 If $T_1 = (P, A, -\mu\rightarrow_1, p_0)$ and $T_2 = (Q, A, -\mu\rightarrow_2, q_0)$ are two transiton systems then we have:

$$T_1 \sim_s T_2 \quad \text{iff} \quad \text{for all } s \in A^* \; q_0 = s\Rightarrow \text{ if and only if } p_0 = s\Rightarrow \qquad \square$$

It is clear that we have $T_1 \sim_s T_2$ iff $Traces(q_0) = Traces(p_0)$ and it is obvious that $\sim_s$ is an equivalence relation, which we call **strings equivalence**.

Example



$\square$

In fact, the equivalence $\sim_s$ is the equivalence used in automata and languages theory over the years and is the basis of many proposed semantics for Hoare's CSP /Hoe81, BHR84, OH83/. Unfortunately, when considering systems which do not run in isolation but exchange information or synchronize with other systems, it is important to know whether certain communications will always take place or whether there is the possibility of deadlock.

If we want to be able to model and distinguish such situations we find that, in spite of its simplicity, strings equivalence is not a useful notion. In fact, referring to the above example, if we try to exchange the sequence of messages "ab" with system T_1 we will be always successful while this is not so if we try to do the same with the systems T_2 or T_3. Moreover T_2 and T_3 might exhibit very different reactions as well; while T_2 would always accept the pair of messages "b, c" after the acceptance of the message "a", T_3 might not. It should also be noted that if we let τ^ω denote the transition system which can only perform an infinite sequence of internal moves and we let NIL denote the system which cannot perform any move at all we have $\tau^\omega \sim_s$ NIL. In fact the set of their visible traces is the same; indeed the possibility of **divergence** (performing calculations ad infinitum without delivering any visible result) is ignored by strings equivalence.

1.3. Observational Equivalence and Bisimulation

There are various ways to "strengthen" strings equivalence in order to be able to differentiate the transition systems of the above example and others similar. The additional discriminating power a new equivalence needs to have is to be able to take into account not only the sequences of actions a system may perform but also "some" of the intermediate states the system goes through while performing a particular sequence of actions. In fact, differing intermediate states can be exploited in different ways to produce different overall behaviours. The first proposal in this direction was put forward by R. Milner. In /Mil80/ and in previous related works a so called **observational equivalence** is defined and then applied to a **C**alculus of **C**ommunicating **S**ystems (CCS). Observational equivalence ($\approx$) is defined as the intersection of a decreasing sequence of equivalences $\approx_k$ ($k \geq 0$) where $\approx_0$ is the universal relation and, for each $k > 0$, the equivalence $\approx_k$ is defined in terms of $\approx_{k-1}$.

Definition 1.3.1 If $T = (Q, A, -\mu\rightarrow, q_0)$ is an LTS and $p, q \in Q$ then

1. $p \approx_0 q$ is always true
2. $p \approx_k q$ iff for all $s \in A^*$
   - i. If $p=s\Rightarrow p'$ then for some q', $q=s\Rightarrow q'$ and $p' \approx_{k-1} q'$
   - ii. If $q=s\Rightarrow q'$ then for some p', $p=s\Rightarrow p'$ and $p' \approx_{k-1} q'$
3. $p \approx q$ iff $p \approx_k q$ for all $k \geq 0$ □

This relation between states of a particular transition system can be easily extended to a relation between two transition systems.

Definition 1.3.2 If $T = (P \cup Q \cup \{t_0\}, A, -\mu\rightarrow_1 \cup -\mu\rightarrow_2, t_0)$ is the transition system obtained from the union of $T_1 = (P, A, -\mu\rightarrow_1, t_0)$ and $T_2 = (P, A, -\mu\rightarrow_2, q_0)$ where P and Q are disjoint then:

- i. $T_1 \approx_k T_2$ if and only if $p_0 \approx_k q_0$ in T and
- ii. $T_1 \approx T_2$ if and only if $p_0 \approx q_0$ in T □

An alternative way of defining observational equivalence has been put forward by D. Park, /Par81/. It has been inspired by the homomorphism notions also used in automata theory (e.g. see the weak homomorphism of /Gin68/). This alternative definition has been used and discussed in detail in /Mil83/ and /Mil84/.

Definition 1.3.3 If T_1 and T_2 are two LTSs, as in the previous definition, then we say that T_1 **bisimulates** T_2 **via** $R \subseteq P \times Q$ if

1. $(p_0, q_0) \in R$
2. $(p, q) \in R$ implies for all $s \in A^*$

   if $p=s\Rightarrow p'$ then for some q', $q=s\Rightarrow q'$ and $(p', q') \in R$

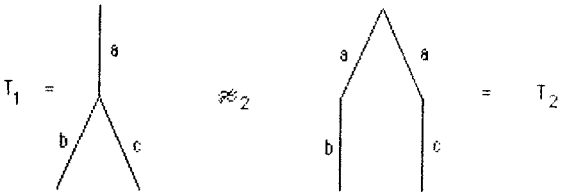
   and if $q=s\Rightarrow q'$ then for some p', $p=s\Rightarrow p'$ and $(p', q') \in R$

□

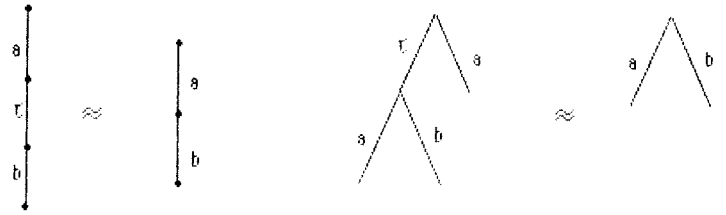
The last two definitions of equivalence are discussed extensively in /San82/ and /HM85/; in particular there it is shown that $\approx$ and bisimulates via R ($\approx_R$) coincide if the relation $=s\Rightarrow$ is **image finite**. However they are seen to be different relations when applied to infinite transition systems, indeed $T_1 \approx_R T_2$ implies $T_1 \approx T_2$ but not viceversa.

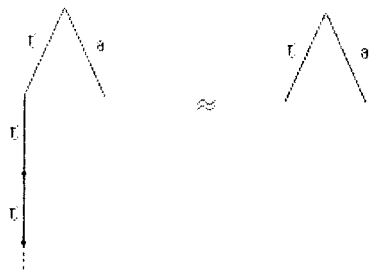
We give now some examples of transition systems (represented as trees) which either are both observational equivalent and bisimilar (the relation R will be evident) or are both neither observational equivalent nor bisimilar. We will use $\approx$ and $\not\approx$ to express this.

Examples

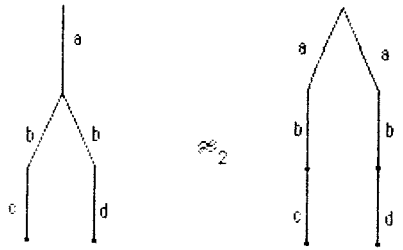


Since if the initial states of T_1 and T_2 are denoted respectively by p and q and the states after a are denoted by $\underline{b+c}$, $\underline{b}$ and $\underline{c}$ with the obvious correspondence, we have $p=a\Rightarrow\underline{b+c}$ while $q=a\Rightarrow\underline{b}$ or $q=a\Rightarrow\underline{c}$ and $\underline{b} \not\approx_1 \underline{b+c} \not\approx_1 \underline{c}$. On the other hand, some systems are reported below which are easily proved observationally equivalent:





The last identification shows that both observation equivalence and bisimulation have problems in coping with infinite transition systems, in particular with transition systems with an infinite number of consecutive internal moves. This is because they consider equivalent systems which computationally are very different. In fact, a system which can either compute for ever or perform the action "a" and then stop is considered equivalent to a system which can perform a silent move or action "a" and then stop. However we also have:



Yet it seems intuitively clear that there is no way of distinguishing these two processes if only the visible actions they may perform and the way they may react to external experiments are considered. Therefore it seems that the notion of observational equivalence may be too discriminating from some points of wiew. In fact, it implicitly assumes that an observer is able each time to make copies of the observed system and then test them separately. In many cases, it may be more appropriate to observe only the overall results of complete experimentations ("linear runs").

1.4. Weak Equivalences

The main reason for $\approx$ being finer (more discriminating) than one may like seems to be the recursive nature of its definition. In a certain sense, in order to decide whether two agents are observationally equivalent, it is necessary to check that they can perform the same sequences of actions and that the subagents reached after each sequence still have an equivalent behaviour. Because of this, some of the resulting distinctions are only concerned with the internal structure of processes.

An interesting critique of observational equivalence is given in /Dar82/; here the author also proposes an alternative equivalence. A similar critique is put forward by J. Kennaway in /Ken81/ and another equivalence, based on ideas discussed in /KH80/, is proposed. Similarly to observational equivalence both these new equivalences are based on recursively proposing external experiments to processes and on comparing the outcomes of these experiments. However, in these cases the particular kind of permitted experiments gives less insight into the structure of systems.

In the present section we will describe and discuss Kennaway's **weak equivalence** by adapting the definition given for his **NCSP** calculus (/Ken81/ pg. 91-93) to transition systems. Moreover we will consider Darondeau's definition of equivalence and discuss its extension to general transition systems, in fact only a particular subset of finite LTS is considered in /Dar82/. We will show that neither of these equivalences exploits the full power of recursion by giving alternative, non-recursive, definitions for them. We conclude the section by showing that a similar definition may also be given for the equivalence induced by the denotational approach of /BHR84/ which will be discussed in details in Chapter 4.

1.4.1 Kennaway's Equivalence

We shall begin with some definitions and some notations based on those of section 1.

We will let L, M and L_x, M_x range over the finite subsets of A, A_x respectively. Moreover we will let P, Q and R denote sets of states.

Definition 1.4.1 (Kennaway's equivalence)

a. For $\mu \in A_x$

$p \underline{\text{after}}\ \mu = \{p' \mid p \Rightarrow_\mu p'\}$

$P \underline{\text{after}}\ \mu = \bigcup \{p \underline{\text{after}}\ \mu \mid p \in P\}$;

b. For $L_x \subseteq A_x$, L_x *finite*

$p\ \underline{\text{MUST}}_\tau\ L_x$ iff $\exists \mu \in L_x$ such that $p \Rightarrow_\mu$

$P\ \underline{\text{MUST}}_\tau\ L_x$ iff $p\ \underline{\text{MUST}}_\tau\ L_x$ for all $p \in P$;

c. $P \approx_0 Q$ is always true

$P \approx_{n+1} Q$ if and only if

i. for all *finite* $L_x \subseteq A_x$ $P\ \underline{\text{MUST}}_\tau\ L_x$ iff $Q\ \underline{\text{MUST}}_\tau\ L_x$

ii. for all $\mu \in A_x$ $P \underline{\text{after}}\ \mu \approx_n Q \underline{\text{after}}\ \mu$;

d. $P \approx_{ken} Q$ if and only if $P \approx_n Q$ for all $n \geq 0$ □

As already mentioned above, we can give a non-recursive characterization of Kennaway's equivalence. First we need some additional definitions. We need to generalize the function after to sequences of actions and to restrict the predicate $\underline{\text{MUST}}_\tau$ to finite sets of **visible** actions; the new predicate will be simply called MUST.

Definition 1.4.2.

a. For $s \in A^*$

$p \underline{\text{after}}\ s = \{p' \mid p \Rightarrow_s p'\}$

$P \underline{\text{after}}\ s = \bigcup \{p \underline{\text{after}}\ s \mid p \in P\}$;

b. For $L \subseteq A$, L *finite*

$p\ \underline{\text{MUST}}\ L$ iff $\forall p'$ such that $p \Rightarrow p'$, $\exists a \in L$ such that $p' \stackrel{a}{\Rightarrow}$

$P\ \underline{\text{MUST}}\ L$ iff $p\ \underline{\text{MUST}}\ L$ $\forall p \in P$ □

From these definitions we can easily derive a set of results which will be helpful in successive proofs.

Proposition 1.4.3

i. $(P \underline{\text{after}}\ \varepsilon)\ \underline{\text{MUST}}\ L$ iff $P\ \underline{\text{MUST}}\ L$

ii. $(P \underline{\text{after}}\ a) \underline{\text{after}}\ s = P \underline{\text{after}}\ as$

iii. $\emptyset\ \underline{\text{MUST}}\ L$ for any $L \subseteq A$ □

Definition 1.4.4

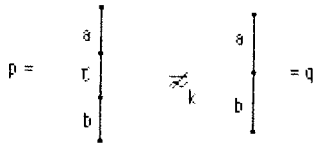
a. $P \sqsubseteq Q$ if and only if for all $s \in A^*$, for all finite $L \subseteq A$

$(P \underline{\text{after}}\ s)\ \underline{\text{MUST}}\ L$ implies $(Q \underline{\text{after}}\ s)\ \underline{\text{MUST}}\ L$;

b. $P \simeq Q$ if and only if $P \sqsubseteq Q$ and $Q \sqsubseteq P$ □

Unfortunately, as it stands, Kennaway definition has a number of major drawbacks, which contradict the philosophy which seems to be at the basis of its definition.

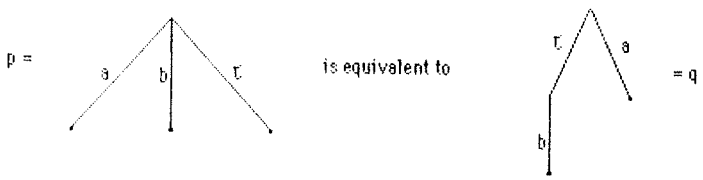
Example



In fact we have that $(q \underline{\text{after}}\ a) \underline{\text{after}}\ \tau\ \underline{\text{MUST}}_k\ L$ for every finite $L \subseteq A$ while $(p \underline{\text{after}}\ a) \underline{\text{after}}\ \tau\ \underline{\text{MUST}}_k\ L$ for any finite L such that b is not in L. □

The problem seems to be originated by the fact that we can test what happens after a process has performed an invisible (τ-) action. However, if we do not allow μ to be equal to τ in definition 1.4.1 then we will find that unwanted identifications are induced by the new equivalence.

Example



In fact it is easy to see that p $\underline{\text{MUST}}_k$ L_x iff q $\underline{\text{MUST}}_k$ L_x and that p $\underline{\text{after}}$ s = q $\underline{\text{after}}$ s for every $s \in A^+$. □

However, the above processes would react differently to external experiments. In fact q must always accept a b-experiment while p may not. In this case the source of the problem seems to be the nature of the definition of $\underline{\text{MUST}}_k$, which does not allow the behaviour of a process to be tested after initial invisible actions.

We now propose a modified version of Kennaways equivalence which does not allow any experiment on τ-moves. This version is that referred to as Kennaway's equivalence in /DH84/. It is based on the notion of $\underline{\text{MUST}}$ in definition 1.4.4. We will denote it by $\approx_{nk}$.

Definition 1.4.5 (New Kennaway's equivalence)

$P \approx_0 Q$ is always true

$P \approx_{n+1} Q$ if and only if

i. for all *finite* $L \subseteq A$ P $\underline{\text{MUST}}$ L iff Q $\underline{\text{MUST}}$ L

ii. for all $a \in A$ P $\underline{\text{after}}$ a $\approx_n$ Q $\underline{\text{after}}$ a;

$P \approx_{nk} Q$ if and only if $P \approx_n Q$ for all $n \geq 0$ □

Proposition 1.4.6 $P \approx_{nk} Q$ if and only if $P \simeq Q$

Proof

1. (if)

Suppose $P \not\approx_{nk} Q$, then by definition there exists $n \geq 0$ such that $P \not\approx_n Q$. By induction on n we show that this implies there exist $s \in A^*$ and $L \subseteq A$ such that (P $\underline{\text{after}}$ s) $\underline{\text{MUST}}$ L and (Q $\underline{\text{after}}$ s) $\underline{\text{MUST}}$ L, or viceversa in P and Q.

a. Induction basis.

We have $P \not\approx_1 Q$ implies there exists some L such that P $\underline{\text{MUST}}$ L and Q $\underline{\text{MUST}}$ L. It follows trivially that (P $\underline{\text{after}}$ ε) $\underline{\text{MUST}}$ L and (Q $\underline{\text{after}}$ ε) $\underline{\text{MUST}}$ L, or viceversa in P and Q.

b. Inductive step.

We have $P \not\approx_{n+1} Q$ if and only if i) $P \not\approx_1 Q$ or ii) there exists an $a \in A$ such that P $\underline{\text{after}}$ a $\not\approx_n$ Q $\underline{\text{after}}$ a. In case i) the claim follows from the induction basis. In case ii) we have from the inductive hypothesis that for some $a \in A$, $s \in A^*$ (P $\underline{\text{after}}$ a) $\underline{\text{after}}$ s $\underline{\text{MUST}}$ L and (Q $\underline{\text{after}}$ a) $\underline{\text{after}}$ s $\underline{\text{MUST}}$ L, i.e. (P $\underline{\text{after}}$ as) $\underline{\text{MUST}}$ L and (Q $\underline{\text{after}}$ as) $\underline{\text{MUST}}$ L.

2. (only if)

Suppose there exists some $s \in A^*$ and some finite $L \subseteq A$ such that (P $\underline{\text{after}}$ s) $\underline{\text{MUST}}$ L and (Q $\underline{\text{after}}$ s) $\underline{\text{MUST}}$ L. We prove by induction on s that $P \not\approx Q$.

a. Induction basis.

The case $s = \varepsilon$ is trivial since p $\underline{\text{after}}$ ε = $\{p' \mid p \Rightarrow p'\}$.

b. Inductive step.

In case s = as' then (P $\underline{\text{after}}$ a) $\underline{\text{after}}$ s' $\underline{\text{MUST}}$ L whereas (Q $\underline{\text{after}}$ a) $\underline{\text{after}}$ s' $\underline{\text{MUST}}$ L. By induction (P $\underline{\text{after}}$ a) $\not\approx$ (Q $\underline{\text{after}}$ a) and so we have $P \not\approx Q$. □

1.4.2. Darondeau's Equivalence

While Kennaway's equivalence is based on looking for the set of actions the various processes must accept after performing any sequence of actions, Darondeau's "**fully**" **observational equivalence** is based on the idea of considering as equivalent two processes which can reject the same sets of actions. As already mentioned, for technical reasons, in /Dar82/ the equivalence was defined only for a subset of LTS, but it generalizes smoothly. Again, we need to define formally what it means for a process in a particular state p to refuse a set of actions L. In /Dar82/ this is denoted by p↓L, we will use the more explicit p refuses L.

Definition 1.4.7

a. p refuses L iff there exists p' such that $p \Rightarrow p'$ and $p' \not\Rightarrow$ for all $a \in L$

b. P refuses L iff there exists $p \in P$ such that p refuses L □

Note that refuses is just the dual of MUST.

Lemma 1.4.8

i. p refuses L if and only if p MUST L

ii. P refuses L if and only if P MUST L

Proof The claim follows from simple logical manipulations □

Definition 1.4.9

Let $\mathbf{E}(R) = \{\langle P, Q\rangle \mid$ i. for all finite L, P refuses L iff Q refuses L and

ii. for all $a \in A$ $\langle P \text{ after } a, Q \text{ after } a\rangle \in R\}$

$P \approx_D Q$ if and only if $\langle P, Q\rangle \in \cup\{R \mid R \subseteq \mathbf{E}(R)\}$ □

As for bisimulation, we have that, since the relation **E** is monotonic, Darondeau's equivalence is defined as the maximal fixed point of the equation $\approx_D = \mathbf{E}(\approx_D)$. Now, lemma

1.4.8 permits the definition of **E** to be reformulated in terms of MUST.

Corollary 1.4.10

$\mathbf{E}(R) = \{\langle P, Q\rangle \mid$ i. for all finite L, P MUST L iff Q MUST L and

ii. for all $a \in A$ $\langle P \text{ after } a, Q \text{ after } a\rangle \in R\}$

Proof Follows from the above lemma. □

In the same way as for Kennaway's equivalence we can now give a non-recursive characterization of Darondeau's equivalence.

Proposition 1.4.11 $P \approx_D Q$ if and only if $P \simeq Q$.

Proof We use the alternative characterization of $\approx_D$ given in the previous corollary.

1. (only if)

Suppose there exists s and L such that (P after s) MUST L and (Q after s) MUST L, then the claim can be proved by induction on the length of s and by using part ii. of proposition 1.4.3. The fact that $\approx_D \subseteq \mathbf{E}(\approx_D)$ is crucial.

2. (if)

Since $\approx_D$ is defined by $\cup\{R \mid R \subseteq \mathbf{E}(R)\}$, it is sufficient to prove that $P \simeq Q$ implies $\langle P, Q\rangle \in \mathbf{E}(\simeq)$. This follows from part i. and ii. of proposition 1.4.3, which makes it possible to prove $P \simeq Q$ implies (P after s) $\simeq$ (Q after s) for every $s \in A^*$. □

1.4.3 Failures Equivalence

In /Bro83a/ yet another equivalence is proposed for a subclass of LTS, the synchronization trees of /Mil80/. This "**Failures**" **equivalence** stems out of the same ideas which led to the denotational model for a theory of CSP based on **refusals sets** /BHR84/. This theory will be discussed in detail in chapter 4. We are able to show that failures equivalence is just another definition of $\simeq$.

Definition 1.4.12 (failures equivalence)

$P \approx_f Q$ if and only if for all $s \in A^*$, for all finite $L \subseteq A$ we have

$$\exists q' \in (Q \text{ after } s) \text{ such that } Init(Q') \cap L = \emptyset$$

if and only if

$$\exists p' \in (P \text{ after } s) \text{ such that } Init(P') \cap L = \emptyset$$

□

Proposition 1.4.13 $P \approx_f Q$ if and only if $P \simeq Q$.

Proof: It is very easy to see that $\simeq$ can be rewritten as follows: $P \simeq Q$ if and only if for all $s \in A^*$, for all finite $L \subseteq A$ ($\forall q' \in (Q \text{ after } s)$, $Init(Q') \cap L \neq \emptyset$ if and only if $\forall p' \in (P \text{ after } s)$, $Init(P') \cap L \neq \emptyset$). The claim now follows from simple logical manipulations.

□

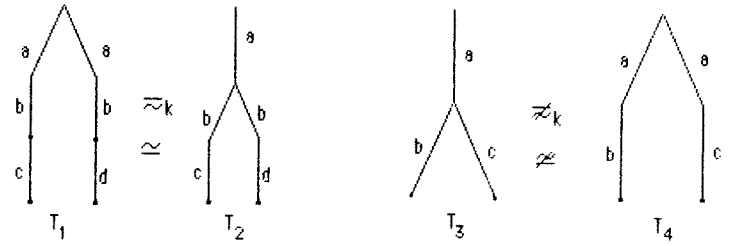
1.4.4 Examples and Discussion

We conclude this section with some examples which will highlight the identifications induced by the various equivalences presented in this section. Note that in the above we have only given definitions of equivalences between states of labelled transition systems. However, from these definitions we can easily derive equivalence relations between LTS's, working as in definition 1.3.2. In fact, given any relation **Rel** defined on sets of states and two transition

systems $T_1 = (A, P, -\mu\rightarrow_1, p_0)$ and $T_2 = (A, Q, -\mu\rightarrow_2, q_0)$, with P and Q disjoint, we can define $T = (A, P \cup Q \cup \{t_0\}, -\mu\rightarrow_1 \cup -\mu\rightarrow_2, t_0)$ and T_1 **Rel** T_2 if and only if p_0 **Rel** q_0, in T.

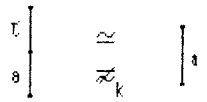
Since propositions 1.4.6, 1.4.11 and 1.4.13 show that the modified version of Kennaway's equivalence ($\approx_{nk}$), Darondeau's equivalence ($\approx_D$) and failures equivalence ($\approx_f$) all coincide with $\simeq$ as defined in 1.4.4, in the examples below we will only mention the latter and discuss its relations with the original Kennaway's equivalence ($\approx_K$).

Examples



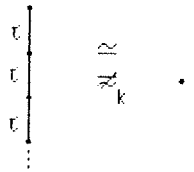
In fact if we let p_i denote the initial states of T_i ($i = 1,2,3,4$) then we have $Traces(p_1) = Traces(p_2)$. Moreover, for every $s \in A^*$, the sets of possible moves of T_1 and T_2, after they have performed s, coincide. This suffices to show that $T_1 \simeq T_2$. On the other hand we have (p_3 after a) MUST {b} while (p_4 after a) MUST {b}, i.e. $T_3 \not\simeq T_4$. With respect to Kennaway's equivalence, we have that, since $p_i - \tau\rightarrow$ for all $i \in \{1, 2, 3, 4\}$, then $\simeq$ and $\approx_k$ induce the same identifications (Proposition 1.4.6). Indeed $\approx_{nk}$ and $\approx_K$ coincide when we consider agents which cannot perform invisible actions.

On the other handside we have:



because the system on the left can perform both a τ- and an a-move while that on the right can only perform an a-move.

The above example and the one given in section 1.4.1 show that $\approx_K$ **is not fully extensional** in the sense that it can distinguish systems which differ only for invisible moves. However, $\simeq$ still has some problems in handling divergent systems. In fact, similarly to strings and observational equivalences, it identifies both a system which can perform an infinite number of internal actions and a system which does not perform any action. It is easy to check that:



In the next section, we propose a different approach to behavioural equivalences explicitly based on the notion of testing by external observers. This approach takes care of all the above problems.

1.5 A Theory of Testing

The external behaviour of programs or processes, in general of systems, can be investigated by means of a series of tests, /Moo56/. For example, with sequential systems, we can associate a test with a pair consisting of a predicate on the input domain and a predicate on the output domain. It is very easy to see how the input-output function of a program can be characterized by a set of such tests. For more general systems, more general kinds of tests are needed. When the processes involved may be nondeterministic, it is important not only to

know whether given a particular test a process responds favourably or unfavourably but also if the process responds consistently every time the test is performed. In fact, all the approaches to systems equivalences discussed in the previous sections are based on the notion of external observation and presuppose implicitly the existence of a set of observers, of a way of observing and of criteria for judging the results of the various observations.

In general, one can think of a set of processes and a set of relevant tests. Two processes will be equivalent (with respect to this set of tests) if they pass exactly the same tests. This natural equivalence can be broken down into two preorders on processes. The first will be formulated in terms of the ability to respond positively to a particular test, the second in terms of the inability not to respond positively to a test. In the latter case, a process p will be "less than" a process q if, whenever p must respond positively to a particular test, q must also respond positively. The natural equivalence between processes is obtained by taking the equivalence associated with the conjunction of these two preorders (this is a third preorder).

The rest of the section is devoted to setting up a rather general framework within which we can discuss testing of processes and tabulation of possible outcomes. It should be possible to adopt this general setting to various models of computation. In the next section, we show how this applies to transition systems.

We assume a predefined set of states, States, and we let s range over it. A **computation** is a non-empty (possibly infinite) sequence of states. We let Comp denote the set of computations, ranged over by c.

Let $\mathbf{O}$, $\mathbf{P}$ (ranged over by o, p respectively) be a set of predefined observers and processes. Observers may be thought of as agents which perform tests. The effect of observers performing tests on processes may be formalised by saying that for every o and p there is a non-empty set of computations **Comp**(o, p). If $c \in$ **Comp**(o, p) then the result of o testing p may be computation c. To indicate that a process passes a test, we define some subset of States, denoted Success, to be successful states. A computation is successful if it contains a successful

state. On the other hand, a computation will be called unsuccessful if it contains no successful state. To develop a useful theory we need one further ingredient. The semantic theory of sequential computations, developed in /Sco76/ and /Sto77/ was greatly facilitated by hypothesizing the existence of "partial objects". For example, the symbol Ω is often used to denote a partial program whose behaviour is totally undefined. It will also be convenient for us to consider such partial objects. For this reason we assume the existence of a unary post-fixed predicate on states, $\uparrow$. Informally $s\uparrow$ means that s is a partial state, whose properties are under-defined.

We may now tabulate the effect of an observer o testing a process p by noting the types of computations in Comp(o, p).

For every $o \in \mathbf{O}$, $p \in \mathbf{P}$ let $\mathcal{R}(o, p) \subseteq \{T, \bot\}$, (the result set) be defined by:

i. $T \in \mathcal{R}(o, p)$ if there exists $c \in Comp(o, p)$ such that c is successful;

ii. $\bot \in \mathcal{R}(o, p)$ if there exists $c \in Comp(o, p)$ such that

a. c is unsuccessful or

b. c contains a partial state s which is neither successful nor preceded by a successful state.

Note that we do not differentiate between an experiment which deadlocks, i.e. the computation terminates without reaching a successful state, and an experiment which diverges, i.e. the computation continues forever without reaching a successful state. These both contribute $\bot$ to the result set. The existence of partially-defined states introduces an additional auxiliary notion of divergence, i.e. when a computation reaches a partially-defined state before reaching a successful state. This also introduces $\bot$ into the result set.

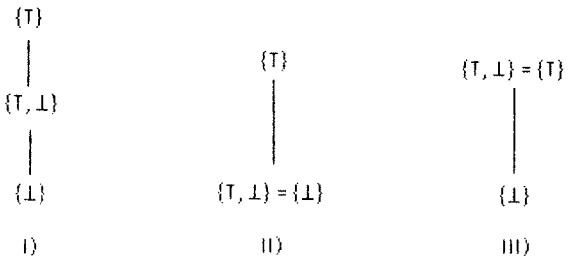
Thus, in effect, we can distinguish between processes which cannot fail a test (the result set is $\{T\}$) and processes which may pass a test (the result set is $\{T, \bot\}$). This will be elaborated upon shortly. A natural equivalence between processes immediately suggests itself:

$p \sim_{\mathbf{O}} q$ if for every $o \in \mathbf{O}$ we have $\mathcal{R}(o, p) = \mathcal{R}(o, q)$.

However, the conceptual framework we introduce will enable us to formulate the equivalences in terms of preorders, i.e. relations which are transitive and reflexive. A preorder $\sqsubseteq$ generates an equivalence in a natural way, $\simeq = (\sqsubseteq \cap \sqsubseteq^{-1})$. In general, preorders (or partial orders) are easier to deal with mathematically and we will recover equivalence $\sim_{\mathbf{O}}$ by studying a preorder which generates it. This will give us a certain amount of freedom since, in general, there may be more than one preorder which generates a given equivalence. Finally, preorders are more primitive than equivalences and therefore we may use them to concentrate on more primitive notions which combine to form equivalence $\sim_{\mathbf{O}}$.

The set $\{T, \bot\}$ may be viewed as the simple two point lattice $\mathbf{\Phi}$:

Each result set can be viewed as a subset of this lattice. The theory of **Powerdomains**, /Plo76, Smy78/, provides us with general methods for ordering subsets of (complete) partial orders. In /Hen82/ it was argued that three different powerdomain constructions arise naturally and that they correspond to three natural views of nondeterministic computations. Since the partial order $\mathbf{\Phi}$ is so trivial, we can completely avoid descriptions of powerdomain constructions and give only the resulting orderings on the nonempty subsets of $\mathbf{\Phi}$.



Ordering I) corresponds to the **Egli-Milner** Powerdomain of Φ; we will denote it by $\sqsubseteq_1$. Ordering II) corresponds to the **Smyth** Powerdomain of Φ. This reflects the view that possible failure is catastrophic; we denote this order by $\sqsubseteq_2$. Finally, ordering III) corresponds to the dual of the Smyth construction and is called the **Hoare** Powerdomain in /Hen82/. The sets $\{\top, \bot\}$ and $\{\top\}$ are identified and are both greater than $\{\bot\}$. This ordering reflects the view that failure is unimportant and is therefore ignored. We denote this preorder by $\sqsubseteq_3$. Note that ordering I) takes a less committed view of failure, it is neither ignored nor considered catastrophic but is simply considered as under-specification.

These three different orderings on result sets generate three different orderings on processes.

Definition 1.5.1 Given a set of observers $\mathbf{O}$, a set of processes $\mathbf{P}$, a set of computations and a notion of successful state, let $\sqsubseteq^{\mathbf{O}}_i \subseteq \mathbf{P} \times \mathbf{P}$ $(i = 1,2,3)$ be defined by :

$p \sqsubseteq^{\mathbf{O}}_i q$ if for all $o \in \mathbf{O}$ we have $\mathcal{R}(o, p) \sqsubseteq_i \mathcal{R}(o, q)$ □

We denote the related equivalences by $\simeq^{\mathbf{O}}_i$. The following results are trivial to establish.

Proposition 1.5.2 a) $p \sim^{\mathbf{O}} q$ if and only if $p \sqsubseteq^{\mathbf{O}}_1 q$ and $q \sqsubseteq^{\mathbf{O}}_1 p$

b) $p \sqsubseteq^{\mathbf{O}}_1 q$ if and only if $p \sqsubseteq^{\mathbf{O}}_2 q$ and $p \sqsubseteq^{\mathbf{O}}_3 q$ □

Thus we have reformulated the natural equivalence $\sim^{\mathbf{O}}$ as the equivalence generated by a preorder $\sqsubseteq^{\mathbf{O}}_1$. This preorder is further broken down into two more primitive preorders $\sqsubseteq^{\mathbf{O}}_2$, $\sqsubseteq^{\mathbf{O}}_3$. The relevance of these primitive preorders can be motivated by the following definition and proposition.

Definition 1.5.3 a) p may satisfy o if $\top \in \mathcal{R}(o, p)$

b) p must satisfy o if $\mathcal{R}(o, p) = \{\top\}$ □

Thus, we have that p may satisfy o if there is a resulting successful computation whereas p must satisfy o if every resulting computation is successful.

Proposition 1.5.4

a) $p \sqsubseteq^{\mathbf{O}}_3 q$ if for all $o \in \mathbf{O}$ p may satisfy o implies q may satisfy o

b) $p \sqsubseteq^{\mathbf{O}}_2 q$ if for all $o \in \mathbf{O}$ p must satisfy o implies q must satisfy o □

In the remainder of this chapter, we apply this general theory to labelled transition systems and, in the next chapter, to the CCS language. To do this for every language or model, we need to specify :

$\mathbf{P}$ - a set of processes;

$\mathbf{O}$ - a set of observers;

States - a set of states, together with a subset of successful states and an underdefined predicate $\downarrow$ on states;

Comp - a method for assigning a non-empty set of computations (sequences of states) to every ⟨process, observer⟩ pair.

1.6. Testing Transition Systems

1.6.1. Testing Equivalences

In this section, we show how to view labelled transition systems as a particular instance of the general setting of the previous section.

Processes will be sets of LTS's over an alphabet A U {τ} of elementary actions. The set of all such processes will be denoted by $\mathcal{T}$ and ranged over by T, V, T_1, T_2 .....;

Observers will be sets of LTS's over A U {τ, w}, where w is not in A and will be the action used to "report success". The set of observers, used to perform experiments on LTS's, will be denoted by $\mathcal{E}$ and ranged over by E, E_1, E_2, ...;

States will be pairs $\langle p, e \rangle$ where p is a state of a process and e is a state of an experimenter. A successful state is a state whose right component is able to perform a w-move. Please note that in this case we will have $\langle p, e \rangle \downarrow$ for every pair p, e, since we do not have partially specified states.

Computations Given two transition systems T, E, from $\mathcal{T}$ and $\mathcal{E}$ respectively, with initial states t and e, a computation from $\langle t, e \rangle$ is a finite or infinite sequence of pairs of states $\langle t_n, e_n \rangle$ where:

1. $\langle t_0, e_0 \rangle$ is $\langle t, e \rangle$;
2. i. $\langle t_n, e_n \rangle - \tau \rightarrow \langle t_{n+1}, e_{n+1} \rangle$ if $t_n - \tau \rightarrow t_{n+1}$ and $e_n = e_{n+1}$ or
   $e_n - \tau \rightarrow e_{n+1}$ and $t_n = t_{n+1}$
   ii. $\langle t_n, e_n \rangle - a \rightarrow \langle t_{n+1}, e_{n+1} \rangle$ if $t_n - a \rightarrow t_{n+1}$ and $e_n - a \rightarrow e_{n+1}$
3. if the sequence is finite with $\langle t_k, e_k \rangle$ as final element then $\langle t_k, e_k \rangle - \mu \not\rightarrow$ for all $\mu \in A \cup \{\tau\}$.

We will let Comp(T, E) denote the set of computations from $\langle t_0, e_0 \rangle$. With abuse of notation we will also use Comp(t_0, e_0) to denote the same thing, whenever T and E are evident from the context. From the general setting and from the previous instantiations we have:

i. T may satisfy E if there exists $s \in A^*$ such that $t_0 = s \Rightarrow$, $e_0 = s \Rightarrow e_n$ and $e_n - w \rightarrow$

ii. T must satisfy E if for every computation $\langle t_0, e_0 \rangle - \mu_1 \rightarrow \langle t_1, e_1 \rangle - \mu_2 \rightarrow \langle t_2, e_2 \rangle \rightarrow \ldots$ there exists $n \geq 0$ such that $e_n - w \rightarrow$.

After these definitions, we can introduce the three preorders on LTS which they generate and the corresponding equivalence relations. We will drop the occurrence of $\mathcal{E}$ whenever this causes no confusion.

Definition 1.6.1

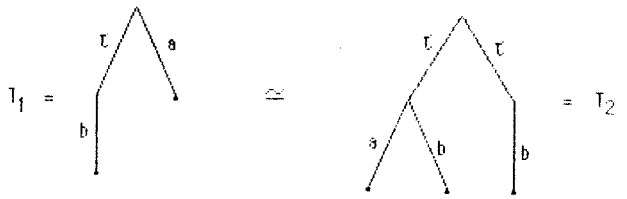
a) $T_1 \sqsubseteq_3 T_2$ if for all $E \in \mathcal{E}$ T_1 may satisfy E implies T_2 may satisfy E

b) $T_1 \sqsubseteq_2 T_2$ if for all $E \in \mathcal{E}$ T_1 must satisfy E implies T_2 must satisfy E

c) $T_1 \sqsubseteq_1 T_2$ if $T_1 \sqsubseteq_2 T_2$ and $T_1 \sqsubseteq_3 T_2$ □

We give now two examples which illustrates the kind of equivalences induced by the previous definitions. We will abbreviate $\simeq_1$ and $\sqsubseteq_1$ as $\simeq$ and $\sqsubseteq$.

Example

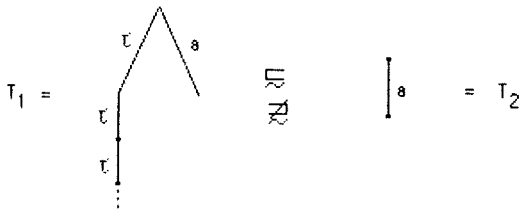


Proof

a) Suppose T_1 may satisfy E. If $e_0 = w \Rightarrow$ then we have T_2 may satisfy E; otherwise T_1 may satisfy E implies $e_0 = a \Rightarrow e' = w \Rightarrow$ or $e_0 = b \Rightarrow e'' = w \Rightarrow$, in both cases we have T_2 may satisfy E. A similar analysis will show that T_2 may satisfy E implies T_1 may satisfy E for all $E \in \mathcal{E}$.

b) Suppose T_1 must satisfy E. We have that $e_0 - w \rightarrow$ implies T_2 must satisfy E. In case $e_0 - w \not\rightarrow$ then T_1 must satisfy E implies that for every e_1 such that $e_0 = \Rightarrow e_1$ we must have that if $e_1 = b \Rightarrow e'_1$ then $e'_1 = w \Rightarrow$ and that if $e_1 = a \Rightarrow e''_1$ then $e''_1 = w \Rightarrow$. It is easy to see that in these cases T_2 must satisfy E. Once again a similar analysis will show that T_2 must satisfy E implies T_1 must satisfy E for all $E \in \mathcal{E}$. □

Example



Proof

1. $T_1 \sqsubseteq T_2$

a) T_1 may satisfy E implies $e_0 = w \Rightarrow$ or $e_0 = a \Rightarrow e' = w \Rightarrow$; in both cases we have T_2 may satisfy E.

b) Since the initial state, p_0, of T_1 is such that there is an infinite computation of the form $\langle p_0, e_0 \rangle - - \rightarrow \ldots\ldots$ we have T_1 must satisfy E implies $e_0 - w \rightarrow$ and thus, also in this case, T_2 must satisfy E.

2. $T_2 \not\sqsubseteq T_1$

It is easy to verify that if E is such that its only transitions are $\{e_0 - a \rightarrow e_1, e - w \rightarrow\}$ then we have T_2 must satisfy E while T_1 must satisfy E. □

1.6.2. Alternative Characterizations

In section 1.5 we proposed a general approach for the investigation of the behaviour of programs or processes. The general situation may be expressed as follows. Given a set of processes and a set of "relevant" tests two processes are considered to be equivalent if they pass exactly the same tests. In the first part of this section, we have adapted this general setting to a particular computation model, transition systems, by defining sets of relevant tests and what it means for a transition system to pass a test. Even though intuitive the equivalences (preorders) obtained in this way are difficult to verify. In fact, while it is relatively easy to prove that two processes are not equivalent (it is sufficient to find **one** observer which differentiate between them), it turns out to be difficult or at least long and tedious to prove that two processes are equivalent; (we need to take into account **all** the possible observers and the outcome of all their observations). However, at least in the case of transition systems, it is possible to give alternative characterizations of the equivalences (preorders) which are independent from the notion of observers by analyzing the (finite or infinite) sequences of actions each system may perform and the set of actions the system must accept. This new characterization allows us to gain insight into the discriminating power of testing equivalences and to understand their relationships with the weak equivalences studied in the previous section. We start with a series of definitions based on those of section 1.1.

Definition 1.6.2 Given any state p of a transition system T we say:

i. $p \Downarrow \varepsilon$ if $p \Downarrow$

ii. $p \Downarrow as$ if $p \Downarrow$ and $p = a \Rightarrow p'$ implies $p' \Downarrow s$ □

As it might be expected, $\not\Downarrow s$ denotes the negation of $\Downarrow s$.

Definition 1.6.3

a) $T_1 \sqsubseteq'_3 T_2$ if $Traces(p_0) \subseteq Traces(q_0)$

b) $T_1 \sqsubseteq'_2 T_2$ if for all $s \in A^*$, for all finite $L \subseteq A$, $p_0 \Downarrow s$ implies

i. $q_0 \Downarrow s$ and

ii. $(p_0$ after $s)$ MUST L implies $(p_0$ after $s)$ MUST L

c) $T_1 \sqsubseteq'_1 T_2$ if $T_1 \sqsubseteq'_2 T_2$ and $T_1 \sqsubseteq'_3 T_2$ □

Lemma 1.6.4 If $T_1 \sqsubseteq_2 T_2$ then for all $s \in A^*$ we have that $p_0 \Downarrow s$ implies

i. $q_0 \Downarrow s$ *and* ii. $s \in Traces(q_0)$ implies $s \in Traces(p_0)$.

Proof:

i. Suppose there exists $s = a_1 \ldots a_n$ such that $p_0 \Downarrow s$ and $q_0 \Uparrow s$. Then if we choose E such that its set of states is given by $\{e_i \mid 0 \leq i \leq n\} \cup \{e_f, e_w\}$, its initial state is e_0 and its transition relation is given by $\{ e_i - \tau \rightarrow e_w,\ e_i - a_i \rightarrow e_{i+1} \mid 0 \leq i < n\} \cup \{ e_n - \tau \rightarrow e_w,\ e_w - w \rightarrow e_f\}$ we have T_1 must satisfy E and T_2 must satisfy E, i.e. $T_1 \not\sqsubseteq T_2$.

ii. Suppose there exists $s = a_1 \ldots a_n$ such that $p_0 \Downarrow s$, $s \in Traces(q_0)$ and s is not in $Traces(p_0)$. Then choosing an E similar to the one in case i. but such that the transition relation does not contain $e_n - \tau \rightarrow e_w$, we again have T_1 must satisfy E while T_2 must satisfy E.

□

Lemma 1.6.5 $(p$ after $s)$ MUST $\emptyset$ if and only if $s \notin Traces(p)$.

Proof The only if part is trivial since $s \notin Traces(p)$ implies $(p$ after $s) = \emptyset$. To prove the if part, let us suppose $s \in Traces(p)$. We then have that p' exists such that $p = s \Rightarrow p'$ and for no $a \in \emptyset$ we have $p' - a \rightarrow$, i.e. $(p$ after $s)$ MUST $\emptyset$. □

Lemma 1.6.6 If $p_0 \Downarrow s$ and $T_1 \sqsubseteq'_2 T_2$ then $s \in Traces(q_0)$ implies $s \in Traces(p_0)$

Proof: The proof is immediate from the previous lemma. □

We are now ready to prove the main characterization theorem.

Theorem 1.6.7 $T_1 \sqsubseteq_i T_2$ if and only if $T_1 \sqsubseteq'_i T_2$ for $i = 1, 2, 3$.

Proof: Because of the way $\sqsubseteq_1$ and $\sqsubseteq'_1$ have been defined we only need to prove the theorem for $i = 2$ and 3.

i = 3

Let $s = a_1 \ldots a_n$, with $a_i \in A$ and let E be such that the set of its states is given by $\{e_i \mid 0 \leq i < n\} \cup \{e_f\}$, the initial state is e_0 and the transition relation is $\{ e_i - a_i \rightarrow e_{i+1} \mid 0 \leq i < n\} \cup \{e_n - w \rightarrow e_f\}$. We will have $s \in Traces(p_0)$ if and only if T_1 may satisfy E. The claim is an easy corollary of this fact.

i = 2

a. We first prove that $T_1 \sqsubseteq_2 T_2$ implies $T_1 \sqsubseteq'_2 T_2$.

From lemma 1.6.4 we have that $p_0 \Downarrow s$ implies $q_0 \Downarrow s$ for all $s \in A^*$. It remains to prove that for all finite $L \subseteq A$, for all $s \in A^*$ we have $(p_0$ after $s)$ MUST L implies $(q_0$ after $s)$ MUST L. Let $s = a_1 \ldots a_n$ and E be a transition system such that the set of states is given by $\{e_i \mid 0 \leq i \leq n\} \cup \{e_f, e_w\}$, the initial state is e_0 and the transition relation is $\{ e_i - \tau \rightarrow e_w,\ e_i - a_i \rightarrow e_{i+1} \mid 0 \leq i < n\} \cup \{e_n - a \rightarrow e_w \mid a \in L\} \cup \{e_w - w \rightarrow e_f\}$. It is easy to check that $(p_0$ after $s)$ MUST L implies T_1 must satisfy E, which in turn implies T_1 must satisfy E by hypothesis. This will imply $(q_0$ after $s)$ MUST L since we have either that s is not in $Traces(q_0)$ or that for all q such that $q_0 = s \Rightarrow q$, $q = a \Rightarrow$ for some $a \in L$.

b. $T_1 \sqsubseteq'_2 T_2$ implies $T_1 \sqsubseteq_2 T_2$.

We prove that if there exists $E \in \mathbf{E}$ such that T_2 must satisfy E then T_1 must satisfy E. We will prove that if there exists an unsuccessful computation c_2, with $c_2 \in Comp(q_0, e_0)$, then an unsuccessful $c_1, c_1 \in Comp(p_0, e_0)$ will also exist. Now c_2 may be unsuccessful for a number of reasons:

i. $c_2 = \langle q_0, e_0 \rangle - \mu_1 \rightarrow \ldots\ldots - \mu_n \rightarrow \langle q_n, e_n \rangle$ and $\langle q_n, e_n \rangle - \mu \not\rightarrow$ for all $\mu \in A \cup \{\tau\}$, $e_i - w \not\rightarrow$ and $q_i \Downarrow$ and $e_i \Downarrow$ for all $0 \leq i \leq n$.

ii. $c_2 = \langle q_0, e_0 \rangle - \mu_1 \rightarrow \ldots\ldots - \mu_n \rightarrow \langle q_n, e_n \rangle - \mu \rightarrow \ldots\ldots$ and $q_m \Uparrow$ or $e_m \Uparrow$ for some positive m smaller than n, and $e_i - w \not\rightarrow$ for all $0 \leq i < m$.

iii. c_2 is such that for all states $\langle q_n, e_n \rangle$, reachable in a finite number of steps we have $q_n\Downarrow$ and $e_n\Downarrow$, $e_n - w\rightarrow$ and $\langle q_n, e_n \rangle = \mu\Rightarrow$ for some $\mu \in A$.

In all these cases, we can prove that an unsuccessful $c_1 \in Comp(p_0, e_0)$ exists; this is sufficient to prove the claim.

i. We have that there exists $s \in A^*$ such that $\langle q_0, e_0 \rangle = s \Rightarrow \langle q_n, e_n \rangle$, $q_0\Downarrow s$, $e_0\Downarrow s$ and (q_0 after s) MUST Init(e_n). We may have either 1. $p_0\Uparrow s$ or 2. $p_0\Downarrow s$. In case 1., $e_0 = s \Rightarrow$ implies that there exists an unsuccessful computation from $\langle p_0, e_0 \rangle$. In case 2. we have that $T_1 \sqsubseteq'_2 T_2$ implies that if $s \in Traces(q_0)$ then $s \in Traces(p_0)$. Therefore c_1 must exist such that $\langle p_0, e_0 \rangle = s \Rightarrow \langle p_n, e_n \rangle$, since ($q_0$ after s) MUST Init(e_n) implies (p_0 after s) MUST Init(e_n) by hypothesis and in both c_1 and c_2 experimenter E may go through the same sequence of states, we have that if c_2 is unsuccessful then so is c_1.

ii. We may have that there exists $s \in Traces(q_0) \cap Traces(e_0)$ such that $q_0\Uparrow s$ or $e_0\Uparrow s$; now we have that this and $T_1 \sqsubseteq'_2 T_2$ implies that either $p_0\Uparrow s$ or $e_0\Uparrow s$ (lemma 1.6.6). Since experimenter E goes through the same sequence of states, we have that there exists an unsuccessful computation from $\langle p_0, e_0 \rangle$.

iii. We have $q_n\Downarrow$ for all the states of the computation, i.e. we have that for any $s \in A^*$ such that $\langle q_0, e_0 \rangle = s \Rightarrow \langle q_n, e_n \rangle$, $q_0\Downarrow s$. From lemma 1.6.6 we have that either $p_0\Uparrow s$ or $\langle p_0, e_0 \rangle = s \Rightarrow \langle p_m, e_n \rangle$ for some $m \geq 0$. In both cases, using reasoning similar to case i., we can prove that there is an unsuccessful c_1. □

1.7. Alternative Forms of Testing

Most of the notions (observer, state, computation) used in section 1.5 to set up the general framework for testing appear natural and correspond to precise intuitions. However, there are a few which are more debatable. Among these we have: the way the possible outcomes of observations can be tabulated and, more importantly, the way the computations generated by testing a process p with an observer o can be noted. In particular there are different choices about noting the effects of experiments (observations) which lead to infinite computations or to diverging computations.

In section 1.5 we chose to consider as successful a computation which had gone through a successful state before going through a diverging one. This choice has been vindicated by the simple alternative characterization of the equivalences which the obtained general framework induces on transition systems. However we could have taken an apparently more natural approach by considering successful only those computations which never go through partially specified states and always terminate. I.e. we could have considered successful only those computations which when cannot progress any further are able to report success and moreover are finite and fully specified. The present section discusses this alternative, first by slightly modifying the general setting of section 1.5 and then by applying this to labelled transition systems, as in section 1.6.

In section 1.5 we had:

$\perp \in \mathcal{R}(o, p)$ if there exists $c \in Comp(o, p)$ such that

a. c is unsuccessful or

b. c contains a partial state s which is neither successful nor preceded by a successful state.

The new approach will imply that given any process p and any observer o we will have a new result set $\mathcal{R}'(o, p)$ such that:

$\perp \in \mathcal{R}'(o, p)$ if there exists $c \in Comp(o, p)$ such that

a. c is finite and its final state (o', p') is such that $o' \text{-w}\not\rightarrow$ *or*

b. c contains a state s which is partially defined *or*

c. c is infinite.

On the other hand we have:

$T \in \mathcal{R}'(o, p)$ if $T \in \mathcal{R}(o, p)$.

When applied to transition systems, the new general setting would imply a new definition for must satisfy and new "must-based" preorders, respectively must strictly satisfy and $\sqsubseteq_{\mathfrak{M}2}$, $\sqsubseteq_{\mathfrak{M}1}$. Note that the definition of may satisfy and $\sqsubseteq_3$ are not influenced by these changes. If we keep the same notation and conventions of the previous section we have:

T_1 must strictly satisfy E if every computation from $\langle q_0, e_0 \rangle$ is finite and if $\langle q_0, e_0 \rangle \text{-}\mu_1\rightarrow \langle q_1, e_1 \rangle \ldots \text{-}\mu_n\rightarrow \langle q_n, e_n \rangle$ is a computation then $e_n \text{-w}\rightarrow$.

Definition 1.7.1

a) $T_1 \sqsubseteq_{\mathfrak{M}2} T_2$ if for all $E \in \mathbf{E}$

T_1 must strictly satisfy E implies T_2 must strictly satisfy E

b) $T_1 \sqsubseteq_{\mathfrak{M}1} T_2$ if $T_1 \sqsubseteq_{\mathfrak{M}2} T_2$ and $T_1 \sqsubseteq_{\mathfrak{M}3} T_2$ □

Furthermore, for $\sqsubseteq_{\mathfrak{M}i}$, as for $\sqsubseteq_i$, it is possible to give an alternative characterization which is independent of the notion of observer and is based on testing for the set of sequences a process may perform and on the notion of MUST as defined in section 1.4. This new characterization gives us a better understanding of the differences between the equivalences obtained by immersing transition systems into the two general settings for systems testing.

Definition 1.7.2 Given two transition systems T_1 and T_2, and letting $\mathfrak{D}(s, p)$ be equal to $\{a \mid a \in A, p \Downarrow sa\}$ we have:

$T_1 \sqsubseteq'_{\mathfrak{M}2} T_2$ if for all $s \in A^*$, for all finite $L \subseteq A$,

$L \cap \mathfrak{D}(s, p_0) = \emptyset$ and $p_0 \Downarrow s$ implies

i. $q_0 \Downarrow s$ and

ii. (p_0 after s) MUST L implies (p_0 after s) MUST L □

As for $\sqsubseteq_i$ and $\sqsubseteq'_i$ in the previous section, we can prove that $\sqsubseteq_{\mathfrak{M}i}$ and $\sqsubseteq'_{\mathfrak{M}i}$ coincide. Also in this case we need some lemmas.

Lemma 1.7.3 If $T_1 \sqsubseteq_{\mathfrak{M}2} T_2$ then for all $s \in A^*$ we have that $p_0 \Downarrow s$ implies

i. $q_0 \Downarrow s$ and ii. $s \in Traces(q_0)$ implies $s \in Traces(p_0)$.

Proof To prove i., let us suppose there exists a trace s such that $p_0 \Downarrow s$ and $q_0 \not\Downarrow s$. We can then prove that there exists an experiment E such that T_1 must strictly satisfy E and T_2 must strictly satisfy E. If $s = a_1 \ldots a_n$ it will be sufficient to have an E such that its set of states R is given by $\{e_i \mid 0 \le i \le n+1\} \cup \{e_f\}$, its initial state is e_0 and its transition relation is given by $\{e_i \text{-w}\rightarrow e_f,\ e_i \text{-}a_i\rightarrow e_{i+1} \mid 0 \le i \le n\} \cup \{e_{n+1} \text{-w}\rightarrow e_f\}$. To prove ii. suppose there is an s in $Traces(q_0)$ and not in $Traces(p_0)$, then if we take the experimenter E^S equal to E apart from the fact that $e_{n+1} \text{-w}\not\rightarrow$ we have that T_1 must strictly satisfy E^S and T_2 must strictly satisfy E^S, i.e. $T_1 \not\sqsubseteq_{\mathfrak{M}} T_2$. □

Lemma 1.7.4 If $T_1 \sqsubseteq'_{\mathfrak{M}2} T_2$ and $p_0 \Downarrow s$ then $s \in Traces(q_0)$ implies $s \in Traces(p_0)$.

Proof The proof follows from lemma 1.6.5. □

Theorem 1.7.5 $T_1 \sqsubseteq_{\mathfrak{M}2} T_2$ if and only if $T_1 \sqsubseteq'_{\mathfrak{M}2} T_2$.

Proof

a. (only if)

The proof follows the same pattern as that for part a. of theorem 1.6.7 with E replaced by

an E' defined as follows. The set of states of E' is given by $\{e_i \mid 0 \le i \le n\} \cup \{e_f, e_w\}$, the initial state is e_0 and the transition relation is given by $\{ e_i\text{-}w\rightarrow e_f,\ e_i\text{-}a_i\rightarrow e_{i+1} \mid 0 \le i < n\} \cup \{ e_n\text{-}a\rightarrow e_w \mid a \in L\} \cup \{e_w\text{-}w\rightarrow e_f\}$. In fact it is not difficult to prove that given an s and an L such that $p_0\Downarrow s$ and $L \cap \mathbf{I}(s, p_0) = \emptyset$, we have that ($p_0$ after s) MUST L implies T_1 must strictly satisfy E'. The latter, by hypothesis, implies T_2 must strictly satisfy E' and this implies (q_0 after s) MUST L. Now the claim follows since from lemma 7.3 we also have $p_0\Downarrow s$ implies $q_0\Downarrow s$.

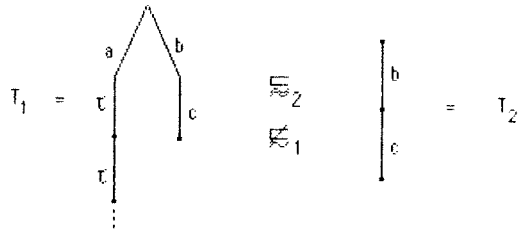
b. (if)

This proof also follows the same pattern as the corresponding one in theorem 1.6.7. We have that $c_2 \in Comp(q_0, e_0)$ may be unsuccessful for a number of reasons. We will consider only one of these reasons since we can deal with the others exactly as in theorem 1.6.7. It may be that there exists an $s \in Traces(q_0) \cap Traces(e_0)$ such that $\langle q_0, e_0\rangle = s \Rightarrow \langle q, e\rangle$, $q_0\Downarrow s$, $e_0\Downarrow s$ and $\langle q, e\rangle\text{-}\mu\not\rightarrow$ for all $\mu \in A \cup \{\tau, w\}$. This implies ($q_0$ after s) MUST Init(e). We have to distinguish two cases: 1. $Init(e) \cap \mathbf{I}(s, p_0) = \emptyset$ and 2. $Init(e) \cap \mathbf{I}(s, p_0) \neq \emptyset$. In case 1. we also have that (p_0 after s) MUST Init(e) and from this we can conclude there is an unsuccessful $c_1 \in Comp(p_0, e_0)$, ($\langle p_0, e_0\rangle = s \Rightarrow \langle p, e\rangle\text{-}\mu\not\rightarrow$ for all $\mu \in A \cup \{\tau, w\}$). In case 2. we would then have there exists $p \in$ (p_0 after s) such that there exists an $a \in Init(e) \cap Init(p)$ such that $\langle p_0, e_0\rangle = sa \Rightarrow \langle p', e'\rangle$ and $p'\uparrow$, i.e. there is an infinite computation from $\langle p_0, e_0\rangle$ and so an unsuccessful one. □

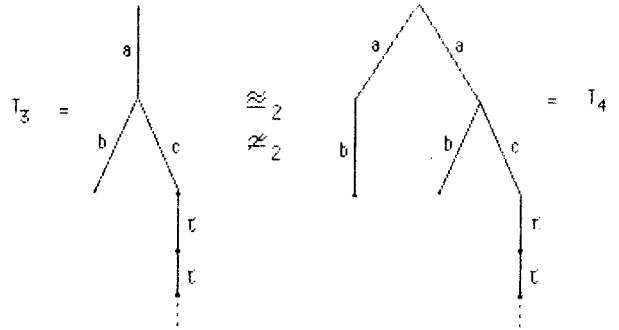
The alternative characterizations of $\sqsubseteq_2$ and $\overline{\approx}_2$ given in definitions 1.6.3 and 1.7.2 should suffice to convince the reader that, if we consider only transition systems which do not contain diverging states, (i.e. states from which is possible for a system to perform an infinite sequence of τ-actions), then the two preorders coincide. This will be proved in the next section, for the moment it should be noted that, if we consider diverging systems, the two preorders turn out to be significantly different. In particular, the preorder $\overline{\approx}_2$ seems to overestimate the fact that after performing a visible action a system may diverge. Indeed it overestimates divergence to the extent of ignoring that the system may perform this action.

Some examples will help to explain this point.

Examples



In fact we have (T_1 after s) MUST L implies (T_2 after s) MUST L for all $s \in A^*$ and for all finite $L \subseteq A - \{a\}$. This result does not correspond to any intuitive notion of approximation.



This result can be proved using reasonings similar to the previous cases. However, also this induced equivalence does not match intuitions about systems behaviour. In fact we have that T_3, after having accepted an a-experiment , will certainly accept a c-experiment, while T_2 may or may not accept c; depending on which of the two a-experiments it has accepted. Note that we have $T_1 \not\sqsubseteq_2 T_2$ since T_1 MUST {a} while T_2 MUST {a}, and $T_3 \not\sqsubseteq_2 T_4$ since (T_3 after a) MUST {c} while (T_4 after a) MUST {c}. Indeed, since the general setting which generates $\overline{\approx}_2$ seemed more natural than the general setting which generates $\sqsubseteq_2$, we began our studies on

testing equivalences by working with $\sqsubseteq_{\approx 2}$, but its difficulties in handling divergent terms convinced us to continue by studying only $\sqsubseteq_2$, and to apply the general setting of the latter to Milner's CCS.

1.8. Comparisons

We have been able to give an alternative characterization based on $\sqsubseteq$, (se Definition 1.4.4) for almost all the equivalences or preorders we have discussed in the previous sections. Because of this, it is not difficult to study their interrelations. Many results are trivial to establish, indeed most of the equivalences differ only because of the way they treat divergence. For this reason, we will report below two groups of results, one for general transition systems and another for strongly convergent ones (see Definition 1.1.4).

Proposition 1.8.1 If T_1 and T_2 are two labelled transition systems then we have:

i. $T_1 \sqsubseteq_2 T_2$ implies $T_1 \sqsubseteq_{\approx 2} T_2$

ii. $T_1 \sqsubseteq_1 T_2$ implies $T_1 \sqsubseteq_{\approx 1} T_2$

iii. $T_1 \simeq_2 T_2$ implies $T_1 \cong_2 T_2$ implies $T_1 \simeq T_2$

iv. $T_1 \simeq T_2$ iff $T_1 \approx_D T_2$ iff $T_1 \approx_{nk} T_2$ iff $T_1 \approx_f T_2$

Proof i. Follows from theorem 1.6.7 and theorem 1.7.4. In fact we have:

(P after s) MUST L implies (Q after s) MUST L for all $L \subseteq A$ implies

(P after s) MUST L' implies (Q after s) MUST L' for every $L' \subseteq A'$, $A' \subseteq A$.

ii. Follows from i. since the definitions of $\sqsubseteq_3$ and $\sqsubseteq_{\approx 3}$ coincide.

iii. and iv. follow from i., Theorems 1.6.7 and 1.7.4 and Propositions 1.4.6, 1.4.11 and 1.4.13. □

Proposition 1.8.2 For strongly convergent transition systems all the equivalences $\approx_D$, $\approx_f$, $\approx_{nk}$, $\simeq_1$, $\cong_1$, $\simeq_2$, $\cong_2$ and $\simeq$ are the same.

Proof The proof follows from the same propositions and theorems used in the proof given above and from Lemmas 1.6.6 and 1.7.4, which indicate that $\sqsubseteq_3$ does not contribute to the definitions of $\simeq_1$ and $\cong_1$ when these are defined on strongly convergent systems. □

Apart from Kennaway's equivalence ($\approx_k$) which exhibits major differences, of all the equivalences considered in this chapter, it only remains to discuss strings equivalence ($\sim_s$), observational equivalences ($\approx_i$ for any $i > 0$ and $\approx$) and bisimulation ($\approx_R$).

Proposition 1.8.3 If T_1 and T_2 are two labelled transition systems then we have:

i. $T_1 \approx_R T_2$ implies $T_1 \approx T_2$ implies $T_1 \approx_2 T_2$ implies $T_1 \simeq T_2$

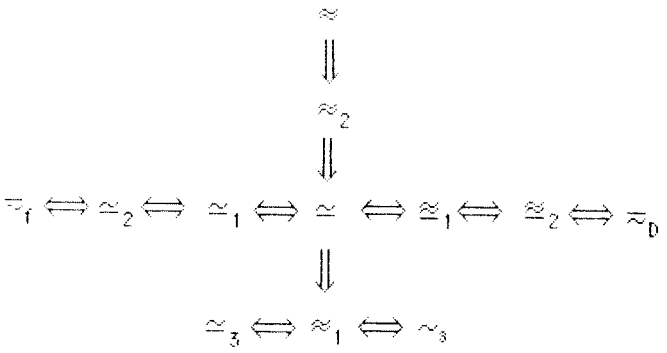
ii. $T_1 \sim_s T_2$ iff $T_1 \simeq_3 T_2$ iff $T_1 \approx_1 T_2$

Proof

i. $\approx_R$ implies $\approx$ is proved in /HM85/ and /San82/. Moreover $\approx$ implies $\approx_2$ is trivial to establish and $\approx_2$ implies $\approx_f$ is proved in /BR83/.

ii. Trivial □

We conclude the chapter with a diagram which summarizes the interrrelations between the various equivalences studied in the previous sections for strongly convergent systems. The implication sign $\Rightarrow$ can be also read as reverse inclusion between relations, i.e. $R_1 \Rightarrow R_2$ means that R_2 is coarser than R_1.



The fact that the single arrows cannot be reversed (we have proper inclusions) can be inferred from the various examples given in the previous sections.

2. PROOF SYSTEMS FOR CCS BASED ON TESTING

2.0. Introduction

In the previous chapter, we discussed Labelled Transition Systems (LTS), a very general model for concurrent and nondeterministic systems. In particular, we showed that equivalences based on external observations, can be defined naturally over these systems. This makes it possible to abstract from their internal behaviour. Unfortunately LTS's do not seem to be a convenient tool to describe such systems; in fact, although they have a simple theory and a natural semantics, they do not guarentee sufficient modularity. Attempts have been made to define calculi based on LTS's which provide a suitable mathematical setting both to describe and reason about systems. In a sense, attempts have been made to perform a step similar to that of formal languages theory, where we go from finite state automata to regular expressions. After the work of Plotkin on structured operational semantics (see e.g. /Plo81/), it became clear that such a step was possible. In fact, Robin Milner has developed an algebra of processes (CCS), based on a small number of operators. Each of these operators corresponds to primitive concepts of the systems we want to model, and the semantics of every operator is based on transition systems. Since its appearence, the Calculus for Communicating Systems (CCS), /Mil80/, has played the role of a touchstone for many of the theories of concurrency proposed. In this calculus, a process p is simply considered as an

agent which accomplishes certain actions and then becomes another agent p'. The relation is denoted by

$$p\text{-}a\rightarrow p'.$$

In this way, starting from a set of elementary actions (whose elements may be "take", "calculate", "put", "send", etc.), from a few basic agents and from a small set of combinators, Milner defines a language which allows processes to be built from smaller ones. The semantics of basic and compound processes is obtained by considering the actions they can perform and their possible configurations after performing these actions. A strong algebraic flavour characterizes the model. Some actions are exchange actions which have inverse and communication results from the co-occurrence of two actions, one the inverse of the other.

The semantics of CCS is given in terms of synchronization trees (a particular subclass of LTS) factorized by observational equivalence. This means that we have a two level semantics: a transition system based, structured operational semantics and an equivalence relation which allows abstraction from unwanted details.

In this chapter, we propose a modified version of CCS which leaves the first semantic level unchanged and immerses the language into the general testing equivalence setting presented in Section 1.5, to define the second level. We also show that interesting algebraic properties of the CCS version based on observational equivalence are preserved and that some new ones (e.g. ω-inductiveness) may be proved. The rest of the chapter is organized as follows. In Section 2.1, after a short introduction to the main features of CCS and its operational semantics (for a detailed discussion see /Mil80/ and /HM85/), we discuss testing equivalences for CCS. The CCS version considered is the "pure" calculus, i.e. without value passing. In Section 2.2 we take as our set of tests those tests which can be described in the CCS version considered and we examine the substitutive relations generated by the three preorders. (Two processes are substitutively related if they are related in every context). In this case we exploit the full power of the general setting of Section 1.5; in fact under-specified processes arise naturally

in CCS as soon as we allow recursive definitions. In Section 2.3 we discuss alternative characterizations of the various preorders. These characterizations are independent of the notions of observers and give additional insight into the discriminating power of testing equivalences. Moreover, we discuss the minimal class of observers we need in order to maintain the same discriminating power. In the successive sections, we show that the three relations over CCS terms can be characterized by three sound and complete sets of axioms. In Section 2.4, we first present the sets of axioms for terms which denote processes with finite behaviour and then extend them to processes with an infinite behaviour by using the well-known idea of syntactic approximations and the fact that the behaviour of infinite terms is completely determined by the behaviour of their finite approximants. Three complete proof systems based on the previous sets of axioms and on a very strong induction rule are presented. Sections 2.5 and 2.6 are dedicated to proving soundness and completeness of the proof systems.

2.1. CCS

In this section we review the definition of CCS and its operational semantics. We use "pure" CCS, /Mil80/, and our version will be closest to that presented in /HP80/.

Let X be a set of variables, ranged over by x. Let Σ_k be a set of operators of arity k. We use Σ to denote $U\{\Sigma_k \mid k\geq 0\}$. The set of recursive terms over Σ, REC_Σ, ranged over by t,u, is defined by the following BNF-like notation:

$$t::=x \mid op(t_1,\ldots,t_k), op \in \Sigma_k \mid rec\ x.t$$

The operation rec x. ... binds occurrences of x in the subterm t of rec x. t. This gives rise to the usual notions of free and bound variables in a term. Let FV(t) be the set of free

variables in t. If $FV(t)= \emptyset$, we say that t is **closed**. Let $CREC_\Sigma$ denote the set of closed terms and let us use p,q, as meta-variables to range over this set. A term is **finite** if it is closed and contains no occurrence of rec x. ... Let $FREC_\Sigma$ denote the set of finite terms, and let us use d, e as meta-variables for finite terms. Let t[u/x] denote the term which results from substituting u for every free occurrence of x in t. More generally, let SUB be the set of **substitutions**, i.e. mappings from variables to terms. We use ρ as a meta-varible over SUB. Let $t\rho$ denote the result of substituting $\rho(x)$ for every free occurrence of x in t, for every x in X. A **substitution** will be called **closed** if for every x in X, $\rho(x)$ is closed.

Pure CCS may be defined by choosing a particular set of operators Σ. We will let **Act** denote a countable set of unary operators ranged over by l or z.The operator $^-$ is the complement operator and $\bar{z}$ is the compement of atomic action z. It will also be convenient to have $\bar{\bar{z}} = z$. Let $\overline{Act}=\{\bar{z} \mid z \in Act\}$. A will be used to denote Act U $\overline{Act}$ and we let it be ranged over by a, b, c, a_1, a_2.... . Moreover we will let $A_\tau = A \cup \{\tau\}$, where τ is a distinguised unary operator not occurring in A. A_τ will be ranged over by μ, μ_1, μ_2 ... . We are using the same notation for unary operators as that used for elementary actions in the previous chapter. This is because, as we shall see, they are very closely related and we can thus carry over to CCS many notational conventions for actions, sequences of actions, etc. .

Let PER denote the set of total functions over A, such that $S \in PER$ complies with the rules i. $S(\tau) = \tau$, ii. $S(\bar{a}) = \overline{S(a)}$ and iii. $S(a) = S(b)$ implies $a = b$. We can now give the operator set Σ for CCS:

$$\Sigma_0 = \{NIL, \Omega\}$$

$$\Sigma_1 = \{\mu \mid \mu \in A_\tau\} \cup \{[S] \mid S \in PER\} \cup \{\backslash l \mid l \in Act\}$$

$$\Sigma_2 = \{+, |\}$$

$$\Sigma_n = \emptyset \text{ for all } n > 2$$

In accordence with /Mil80/, μ will be used in prefix form, $\backslash z$ and [S] in postfix form and + and | in infix form.

Before giving the formal semantics of the various operators we give an informal description of their intended meaning.

1. **Inaction** NIL represents the process which cannot perform any action;

2. **Undefined** Ω represents the partial process, whose behaviour is totally undefined;

3. **Action** If A_τ is a set of elementary actions, $\mu \in A_\tau$ and p is a process μp denotes a process which can only perform the action μ and then behave like p;

4. **Renaming** If p is a process and S a permutation over A, then p[S] is a process whose actions are renamings via S of the actions of process p;

5. **Restriction** If p is a process and $l \in Act$, $p\backslash l$ is a process which cannot perform any l or $\bar{l}$ action, otherwise it behaves like p;

6. **Choice** If p and q are processes then p+q is a process which can act either as p or as q. The choice depends, at least to a certain extent, on the environment in which p+q is used.

7. **Parallel Composition** If p and q are process then p|q is a process which can perform any sequence of actions obtained by arbitrary interleaving of the sequences of actions which p and q can perform. Moreover it can perform silent moves any time p and q are able to perform complementary actions.

8. **Recursion** If x is a variable and p a process the process rec x.p is used to define possible infinite processes through recursive equations. A requirement imposed in /Mil80/ is that p has to be well guarded, i.e. the first actions of p are independent of x. Because of the way we treat partially specified objects we can remove this restriction.

The operational semantics is given in terms of labelled rewrite rules over closed CCS terms. These rules allow to associate a transition system from LTS to every closed term. For each $\mu \in A_X$ we define a relation $-\mu\rightarrow$ over closed terms and use the action τ for denoting that a synchronization between two subprocesses has taken place.

Definition 2.1.1 Let $-\mu\rightarrow$ be the least relation over closed terms which satisfies

act) $\mu p \ -\mu\rightarrow p$

res) $p-\mu\rightarrow q$ implies $p\backslash l \ -\mu\rightarrow q\backslash l$ if $\mu \notin \{l,\bar{l}\}$

rel) $p-\mu\rightarrow q$ implies $p[s] \ s\text{-}s[\mu]\rightarrow q[s]$

sum) $p-\mu\rightarrow p'$ implies $p+q \ -\mu\rightarrow p'$ and $q+p \ -\mu\rightarrow p'$

com) $p-\mu\rightarrow p'$ implies $p|q \ -\mu\rightarrow p'|q$ and $q|p \ -\mu\rightarrow q|p'$

$p-a\rightarrow p'$ and $q-\bar{a}\rightarrow q'$ implies $p|q \ -\tau\rightarrow p'|q'$

rec) $t[rec\ x.t/x] \ -\mu\rightarrow p$ implies $rec\ x.t-\mu\rightarrow p$ □

Note that there is no way of distinguishing NIL and Ω since both do not appear in the above definition and thus both cannot perform any transition. To differentiate between them and to cope with partially specified terms, we need the following binary predicate on terms.

Definition 2.1.2 Let $\downarrow$ be the least predicate on closed terms which satisfies

i) $NIL\downarrow$, $ap\downarrow$

ii) $p\downarrow$, $q\downarrow$ implies $(p+q)\downarrow$, $(p|q)\downarrow$, $p[S]\downarrow$, $p\backslash l\downarrow$

iii) $t[rec\ x.t/x]\downarrow$ implies $rec\ x.t\downarrow$ □

Let $p\uparrow$ if not $p\downarrow$. So for example $\Omega\uparrow$ and $rec\ x.(ap+x)\uparrow$. Informally, $p\uparrow$ means that there is an unguarded recursion or an unguarded occurrence of Ω.

2.2. Testing Equivalences for CCS

In this section, we show how CCS can be viewed as a particular example of the general setting of testing equivalences described in Section 1.5 of the previous chapter. The set of processes will only be closed CCS terms, i.e. terms in $CREC_\Sigma$, and the principal point which must be settled is how observers can be described. It seems reasonable to use the same language to describe both the processes and the observers. An observer can test a process by communicating with it and CCS was designed to describe communication. As in the case of transition systems, we need some additional machinery to be able to indicate that a test has been successful. Let w be a distinguished action symbol, not in A. We use w as a special action which "reports success".

Example: The term $o = \bar{a}.\bar{b}.w.NIL$ is an observer which can be used to test whether a process can perform an a-action followed by a b-action. For example the process $p = a.(b.NIL + c.NIL)$ passes this test because, when o and p are put in communication, success will be eventually reported:

$o|p \ -\tau\rightarrow \bar{b}.w.NIL|(b.NIL + c.NIL) \ -\tau\rightarrow \ w.NIL|NIL \ -w\rightarrow$ □

In order to use the general setting of 1.5 we have to specify what Processes, Observers, States and Computations are.

Processes are closed CCS terms over Σ. We will also use $\mathbf{P}$ to denote this set.

Observers are closed CCS terms over $\Sigma \cup \{w\}$. We will use $\mathbf{O}$ to denote this set and o, o', o_1... to range over it.

States Since CCS is applicative in nature the set of states will be the set of all closed CCS terms over $\Sigma \cup \{w\}$. A state s will be called successful if $s-w\rightarrow$ and divergent if $s\uparrow$.

Computations For two given processes, p and o, **Comp**(p, o) is the set of (finite or infinite) sequences $\{s_n \mid n \geq 0\}$ of terms such that

1. s_0 is p|o;
2. $s_n - \tau \rightarrow s_{n+1}$;
3. if the sequence is finite with s_m as final element then $s_m - \tau \not\rightarrow$.

Since the operational semantics of CCS essentially defines a class of transition systems, the above definitions immediately give three different preorders on $\mathbf{P}(CREC_\Sigma)$. To emphasize their import, we translate Definition 1.5.3 into this setting. The main difference between these preorders and the corresponding ones for LTS is the presence of the predicate testing for partial states.

a) p may satisfy o if $(p|o) = \varepsilon \Rightarrow q$ for some q such that $q - w \rightarrow$

b) p must satisfy o if whenever $p|o = p_0|o_0 - \tau \rightarrow p_1|o_1 - \tau \rightarrow \ldots$. is a computation then there exists $n \geq 0$ such that $o_n - w \rightarrow$ and $p_k|o_k\uparrow$ implies $o_h - w \rightarrow$ for some $h \leq k$.

We have used $\Longrightarrow$ as in Section 1.1. In fact, throughout this chapter, we will use all the notational conventions fixed there. We will also use **Dead** = $\{p \mid p\downarrow, p - \tau \not\rightarrow\}$ and **Fail** = $\{p \mid p \in$ **Dead**, $p - w \not\rightarrow\}$.

From now on, we will drop the occurrences of $\mathbf{0}$ whenever this does not lead to confusion, e.g. $\sqsubseteq_i^{\mathbf{0}}$ will be rendered as $\sqsubseteq_i$. We will adopt the usual notation for CCS terms and their operational semantics. NIL will be omitted from terms; e.g. aNIL+bNIL will be rendered as a+b. We will often revert to the graphical representation of terms, which uses only the operators μ, +, NIL. In fact, there is a close correspondence between the terms which are obtained from these operators and the synchronization trees used in the previous chapter: Ω will also be used to denote τ^ω. The precedence of the operators is given by

$$\backslash\!\mid > [S] > \mu > \mid > +.$$

The preorders $\sqsubseteq_i^{\mathbf{0}}$ induced by the previous instantiation of the predicates may satisfy and must satisfy, of Definition 1.5.4 and Proposition 1.5.2 are defined only on closed terms. However they may be extended to arbitrary terms as follows:

Definition 2.1.2

For $t, u \in REC$ $t \sqsubseteq_i u$ iff for every closed substitution ρ we have $t\rho \sqsubseteq_i u\rho$ □

By and large, the preorders $\sqsubseteq_i$ are well-behaved. For example, they are preserved by all the CCS operators except +. As an example, we prove the result for the composition operator |.

Proposition 2.2.2 $p \sqsubseteq_i q$ implies $p|r \sqsubseteq_i q|r$.

Proof: The result follows from the following remarks:

i = 3: For any $o \in \mathbf{O}$, $(p|r)$ may satisfy o if and only if p may satisfy $(r|o)$.

i = 2: For any $o \in \mathbf{O}$, $(p|r)$ must satisfy o if and only if p must satisfy $(r|o)$.

i = 1: This follows from the two previous cases. □

In general $\sqsubseteq_i$, i=1,2, are not preserved by the operator +. For example $a \sqsubseteq_2 \tau a$ but $b+a \not\sqsubseteq_2 b+\tau a$. In fact, if o denotes δw, then b + a must satisfy o whereas $(b + \tau a)|o - \rightarrow a|\delta w \in$ Dead. We have that $\sqsubseteq_3$ is preserved by all CCS operators, but for uniformity we will treat the three cases together.

In order to use our algebra of processes to prove systems properties and in general to reason about them, rather then equivalences, we need a theory of congruences. In fact, if processes p and and q are equivalent, we would like to have the possibility to replace p by q in any context, without affecting the overall behaviour. This is generally possible only for equivalence relations (preorders) which are also substitutive, i.e. are preserved by all the operators of the calculus. Indeed, we have already seen that $\sqsubseteq_2$ is not preserved by operator

+. We can define new preorders $\sqsubseteq_i^C$ which are based on $\sqsubseteq_i$ but are preserved by all CCS operators. To do this we need a notion of context.

Definition 2.2.3 A context $C[\,]$ is an expression with zero or more "holes", to be filled by terms. We write $C[t]$ for the result of inserting t into each "hole". □

Definition 2.2.4 $t \sqsubseteq_i^C u$ if for every context $C[\,]$ $C[t] \sqsubseteq_i C[u]$ □

In general, there are limited numbers of contexts which are important to determine $\sqsubseteq_i^C$; in our case $r+[\,]$ is sufficient.

Definition 2.2.5 Let $p \sqsubseteq_i^+ q$ if for every closed term r, $r+p \sqsubseteq_i r+q$. □

We can prove that $\sqsubseteq_i^C$ and $\sqsubseteq_i^+$ do coincide.

Proposition 2.2.6 If $t, u \in REC_\Sigma$ then $t \sqsubseteq_i^C u$ iff $t \sqsubseteq_i^+ u$

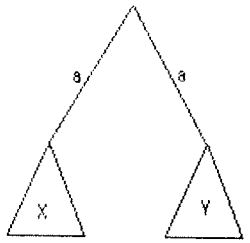
Proof: The only difficulty with this proposition is to prove that $\sqsubseteq_i^+$ is preserved by contexts involving the recursion operators. This requires some technical concepts which will be developed in Section 2.6. The proof will be posponed to that section. □

We now give some examples and counter-examples. These will mainly concern the equivalence $\simeq_1$, which in the following will be abbreviated to $\simeq$.

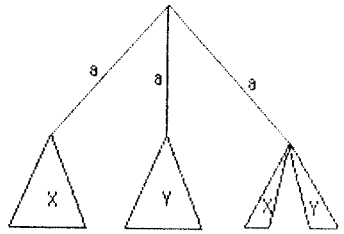
Example 1 For any $X, Y \in REC_\Sigma$

$$aX + aY \simeq^+ aX + aY + a(X+Y)$$

Using the representation of terms by trees of /Mil80/ these may be described as



p



q

The reader may like to convince himself that for any observer o we have: p may satisfy o if and only if q may satisfy o and p must satisfy o if and only if q must satisfy o. □

In the next section, we will give a set of axioms to transform terms which preserve $\simeq^+$ and we will show how to transform p into q.

Example 2 For any $X, Y, Z \in REC_\Sigma$

$$aX + a(X+Y+Z) \simeq^+ aX + a(X+Y) + a(X+Y+Z).$$ □

However, we can distinguish very similar pairs of trees.

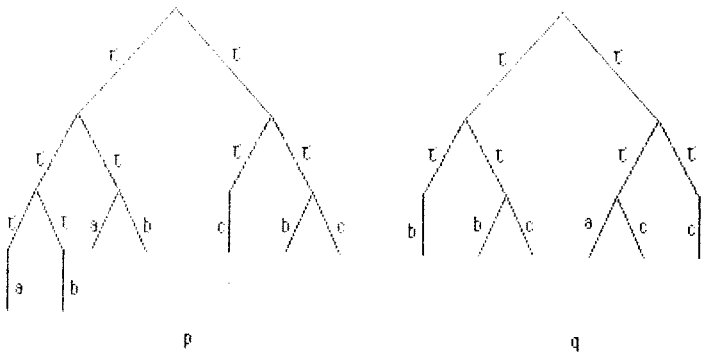
Example 3

$$p = \mu a + \mu(a + b + c) \not\simeq \mu a + \mu(a + b + c) + \mu b = q.$$

This follows since p must satisfy $\bar{\mu}\bar{a}w$ whereas $q|\,\bar{\mu}\bar{a}w \rightarrow b|\bar{a}w \in Fail$. □

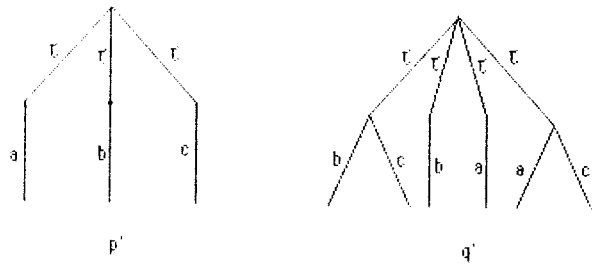
We now consider some examples with internal moves.

Example 4 Consider the two trees:



p q

It turns out that $p \not\simeq q$ although this is not immediately evident. Now consider the two trees.



p' q'

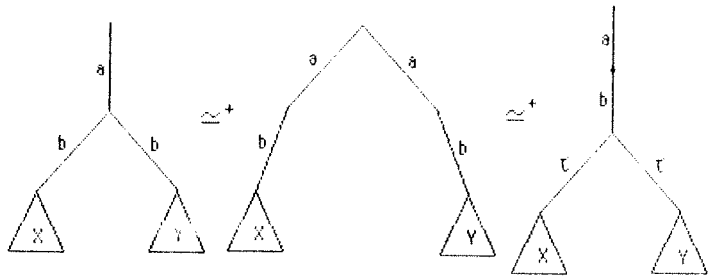
These trees have a much simpler " τ-structure" and it is relatively easy to see that they are not equivalent. For example, no matter what internal move q', makes it can always perform either a b or an a. However if p' makes an internal move to become cNIL, it can perform neither. So q' must satisfy $(\bar{a}w + \bar{b}w)$ whereas p' must satisfy $(\bar{a}w + \bar{b}w)$.

The axioms given in the next section enable us to transform any term into a term with a "τ-structure" similar to that of p', q'. We will see that p may be transformed into p' and q transformed into q', and therefore that $p' \simeq p$ and $q' \simeq q$.

Example 5

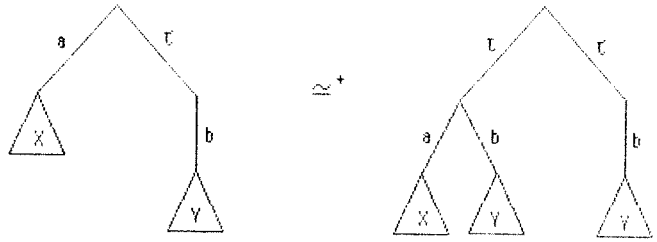
$$a(bX + bY) \simeq^+ abX + abY \simeq^+ ab(\tau X + \tau Y).$$

In terms of trees



This example show that $\simeq^+$ tends to abstract away from "when choices are made".

Example 6 $aX + \tau bY \simeq^+ \tau(aX + bY) + \tau bY$

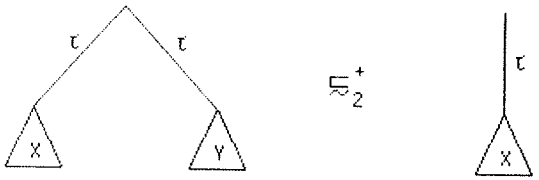


This will in fact be one of our more useful axioms. We can use it to transform terms so that they represent processes in which all choices are either purely external or purely internal.

Example 7

a) $\tau X + \tau Y \approx_2^+ \tau X$

In graphical terms this may be rendered

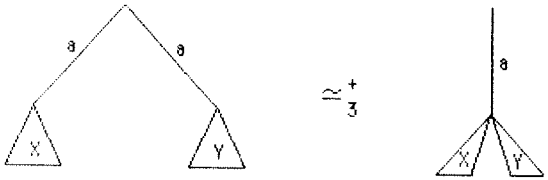


$\approx_2^+$

The presence of τ on the left side is important. For example $a+b \not\approx_2 \tau a$. This follows since $a+b$ must satisfy δw whereas $\tau a|\delta w \rightarrow a|\delta w \in$ Dead.

b) $aX + aY \simeq_3^+ a(X + Y)$

In graphical terms we have



$\simeq_3^+$

Thus the relation $\simeq_3^+$ ignores the entire tree structure of terms. We will also see that $X \simeq_3^+ \tau X$ and thus that $\simeq_3^+$ is a very weak relation.

2.3 Alternative Characterizations

In Section 1.6 of the previous chapter, after using the general ideas about testing to develop a theory of equivalence for transition systems, we gave alternative characterizations for all the generated preorders. These new characterizations were independent of the notion of experiments and were based on simple notions such as checking whether two systems can perform the same sequences of actions and whether, after any sequence, they can choose their next move among the same sets of actions. These alternative characterizations are very useful in practice since using observers to decide whether two systems are equivalent or not may be long and tedious. In fact, in order to show that two systems are not related it is sufficient to exhibit just one observer which differentiate between them. On the other side the effect of all possible observers must be considered to prove a positive statement. In the first part of this section, we prove that alternative characterizations, very similar to those defined for transition systems, can be given for the equivalences induced on CCS by the general framework of testing equivalences.

2.3.1. New tests

In spite of the striking similarities between the two instantiations of the general setting, the results of the previous chapter for alternative characterization do not carry over immediately to CCS. The main reason is the presence of partially specified terms; the slightly different notion of computation is also relevant. We have used different computations in CCS, to stress that, when considering languages with operators for interprocess communication, a natural way of testing them is to compose the processes and the observers using the appropriate parallel operator of the language itself.

However, it will be seen that we deal with underspecified terms in much the same way as for divergent terms in Chapter 1. Instead of having a predicate $\Downarrow$ over terms, which ensures

that there is no infinite sequence of internal actions, we will extend $\Downarrow$ to a predicate which not only ensures that there is no infinite τ-sequence but also that there is no finite τ-sequence which leads to an underspecified term. Because of the way successful computations have been defined, these two different predicates play the same role in the alternative characterization, and with abuse of notation we denote also this predicate over CCS closed terms by $\Downarrow$. Apart from $\Downarrow$, which is redefined below, all the other notations used (MUST, after, etc.) are borrowed from Chapter 1.

Definition 2.3.1

a) $\Downarrow$ is the least predicate such that

i) If $p\downarrow$ then $p\Downarrow$

ii) If $\forall p'$, $p-\tau\rightarrow p'$ implies $p'\Downarrow$, then $p\Downarrow$.

b) $p\Downarrow\varepsilon$ if $p\Downarrow$

$p\Downarrow as$ if $p\Downarrow\varepsilon$ and ($\forall p'$, $p=a\Rightarrow p'$ implies $p'\Downarrow s$) □

As usual we will use $p\Uparrow$ and $p\Uparrow s$ to denote *not* $p\Downarrow$ and *not* $p\Downarrow s$.

The following definition is exactly the same as that in Section 1.6.2, apart from the different meaning of the predicate $\Downarrow$. Since the definitions of testing equivalences for both CCS and LTS's are instances of the same general setting, the proof of the characterization theorem is very similar to the one of theorem 1.6.7. Rather than repeating the entire proof we will simply outline it here, mentioning only the points in which the two proofs differ. The main theorem also uses a series of lemmas whose statements and proofs are identical to Lemmas 1.6.4 - 1.6.6.

Definition 2.3.2

$p \sqsubseteq'_3 q$ if $traces(p) \subseteq traces(q)$;

$p \sqsubseteq'_2 q$ if $s \in A^*$, for all finite $L \subseteq A$, $p\Downarrow s$ implies

i. $q\Downarrow s$ and

ii. (p after s) MUST L implies (q after s) MUST L;

$p \sqsubseteq'_1 q$ if $p \sqsubseteq'_3 q$ and $p \sqsubseteq'_2 q$. □

Theorem 2.3.3

$p \sqsubseteq_i q$ if and only if $p \sqsubseteq'_i q$ for i=1,2,3.

Proof: Because of the way $\sqsubseteq_1$ and $\sqsubseteq'_1$ have been defined we only need to prove the theorem for i = 2, 3.

i = 3: For $s \in A^+$, let $o_3 = sw$. Then $s \in traces(p)$ if and only if p may satisfy o_3. The claim is an easy corollary of this fact.

i = 2: **a.** We first prove that $p \not\sqsubseteq'_2 q$ implies $p \not\sqsubseteq_2 q$.

The core of the corresponding part of the proof of Theorem 1.6.7 consists in proving that if there exist $s = a_1 \ldots a_n$ and $L \subseteq A$ such that

a. $p\Downarrow s$ and $q\Uparrow s$ or

b. $s \in traces(q)$ and $s \notin traces(p)$ or

c. (p after s) MUST L and (q after s) MUST L

than there exists an observer o such that p must satisfy o while q must satisfy o. We will have that o is equal to o^d_2, o^s_2 and $o^{\perp}_2$ respectively in case a, b and c where

$o^d_2 = \tau w + \bar{a}_1 (\tau w + \ldots \bar{a}_{n-1}(\tau w + \bar{a}_n \tau w)\ldots)$

$o^s_2 = \tau w + \bar{a}_1 (\tau w + \ldots \bar{a}_{n-1}(\tau w + \bar{a}_n)\ldots)$

$o^{\perp}_2 = \tau w + \bar{a}_1 (\tau w + \ldots \bar{a}_{n-1} (\tau w + \bar{a}_n \Sigma\{\bar{a} w \mid a \in L\})\ldots)$

b. The proof that $p \sqsubseteq'_2 q$ implies $p \sqsubseteq_2 q$ is identical to that of part b. of theorem 1.6.7.

□

We conclude by deriving a characterization of $\sqsubseteq_i^c$ similar to that for $\sqsubseteq_i$, given in the above theorem. We need the following lemmas.

Lemma 2.3.4 If $p \sqsubseteq_2 q$ then

i) $p \xrightarrow{\tau}$ implies $p \sqsubseteq_2^+ q$

ii) $\tau p \sqsubseteq_2^+ q$

iii) $p \xrightarrow{\tau}\!\!\!\!\!\!\not\;\;$, $q \xrightarrow{\tau}\!\!\!\!\!\!\not\;\;$ implies $p \sqsubseteq_2^+ q$.

Proof:

i) Suppose $r+p$ must satisfy o. Then, since $p \xrightarrow{\tau}$, we have that p must satisfy o. Therefore q must satisfy o and this implies that $r+q$ must satisfy o. It follows that $p \sqsubseteq_2^+ q$.

ii) Suppose $r+\tau p$ must satisfy o. Then, once again, we have that p must satisfy o and, as in case i), this means that $r+q$ must satisfy o.

iii) Suppose $r+p$ must satisfy o. If $p|o \xrightarrow{\tau}\!\!\!\!\!\!\not\;\;$ then, from the characterization of $\sqsubseteq_2$ given in theorem 2.3.3, it follows that $q|o \xrightarrow{\tau}\!\!\!\!\!\!\not\;\;$ and, in this case, $r+q$ must satisfy o. On the other hand if $p|o \xrightarrow{\tau}$ then p must satisfy o. Since we are assuming that $p \sqsubseteq_2 q$, it follows that q must satisfy o and therefore $r+q$ must satisfy o. □

Lemma 2.3.5 If $p \sqsubseteq_2 q$ and ($q \xrightarrow{\tau}$ implies $p \xrightarrow{\tau}$), then $p \sqsubseteq_2^+ q$.

Proof: If $p \xrightarrow{\tau}$, then the result follows from the previous lemma. If $p\!\!\not\downarrow$ then the result is trivially true. Therefore we may assume that $p\!\downarrow$, $p \xrightarrow{\tau}\!\!\!\!\!\!\not\;\;$ and $q \xrightarrow{\tau}\!\!\!\!\!\!\not\;\;$. From part iii) of the previous lemma, it follows once more that $p \sqsubseteq_2^+ q$. □

Theorem 2.3.6

i) $p \sqsubseteq_3^+ q$ if and only if traces(p) $\subseteq$ traces(q)

ii) $p \sqsubseteq_2^+ q$ if and only if

a) ($p\!\downarrow$ and $q \xrightarrow{\tau}$) implies $p \xrightarrow{\tau}$.

b) for all $s \in A^*$ and finite $L \subseteq A$, $p\!\downarrow s$ implies

i) $q\!\downarrow s$

ii) (p after s) MUST L implies (q after s) MUST L.

Proof:

i) Follows from the fact that $\sqsubseteq_3$ is preserved by all of CCS operators.

ii) We certainly have that $p \sqsubseteq_2^+ q$ and $p\!\downarrow$ and $q \xrightarrow{\tau}$ implies $p \xrightarrow{\tau}$. For, assume the antecedent and suppose $p \xrightarrow{\tau}\!\!\!\!\!\!\not\;\;$, then for any action "a" not appearing in p and q we have $a+p$ must satisfy $\bar{a}w$ whereas $a+q$ must satisfy $\bar{a}w$. This together with Theorem 2.3.3 is sufficient to prove the only if part. The if part follows from the same theorem and from Lemma 2.3.5. □

2.3.2. Minimal sets of observers

Many observers are useless for distinguishing processes. For example, if o contains no occurrence of w then, for every process p, we have $\mathcal{R}(o,p) = \{\perp\}$. This means that some observers may be ignored. In general, one can ask which is the smallest set of observers which generates the preorders $\sqsubseteq_i$. The observers used in the proof of theorem 2.3.3 suggest the specific observers needed to distinguish processes.

Let $\mathbf{O}_1 = \mathbf{O}_2 \cup \mathbf{O}_3$ where $\mathbf{O}_2, \mathbf{O}_3$ are the sets of observers generated, respectively, by the following grammars, where Σ stands for finite summation and $a, b \in A$.

a. $t ::= \Sigma\{aw \mid a \in L\} \mid \tau w + bt$

b. $t ::= aw \mid at$.

Theorem 2.3.7 $p \sqsubseteq_i q$ if and only if $p \sqsubseteq_i^{\mathbf{O}_i} q$ for $i = 1, 2, 3$.

Proof: We need to prove the claim only for $i = 2, 3$. The "only if" part is immediate since $\mathbf{O}_i \subset \mathbf{O}$, where $\mathbf{O}$ is the set of observers used to define $\sqsubseteq_i$. We only need to prove the "if" part.

i=3: The claim follows directly from the corresponding case of Theorem 2.3.3.

i=2: Notice o^d_2, o^s_2 and $o^{\perp}_2$ all belong to $\mathbf{O}_2$. We have $p \not\sqsubseteq_2 q$ implies $p \not\sqsubseteq'_2 q$ and, as shown in the proof of Theorem 2.3.3, this implies that p must satisfy o and q must satisfy o for some $o \in \mathbf{O}_2$. □

We can show that we need observers generated by all the clauses in the definition of $\mathbf{O}_2$ and $\mathbf{O}_3$. The case of $\mathbf{O}_3$ is very simple since the only observers we have are unbounded sequences of actions and these are needed in order to test processes which differ after an arbitrary number of moves. Let us consider $\mathbf{O}_2$. We can see that observers from $\mathbf{O}_2$ are indeed necessary. For example we have $a(b+c) + ad \not\simeq_2 ab + a(c+d)$ but no observer from $\mathbf{O}_3$ can distinguish between them. In fact we can show that we need both observers of the form $aw+bw$ and $\tau w+a$.

i) If we omit $\tilde{a}w + \tilde{b}w$ we have that $p = \tau a + \tau(b+c)$ and $q = \tau(a+b) + \tau c$ would be considered equivalent while we have p must satisfy $\tilde{a}w + \tilde{b}w$ and q must satisfy $\tilde{a}w + \tilde{b}w$.

ii) If we omit $\tau w+a$ we have, for $p = b$ and $q = a + b$, that $p \simeq_2 q$ while p must satisfy $\tau w+a$ and q must satisfy $\tau w+a$.

These classes of observers shed light on the nature of the basic experiments. Probably the most important thing is that all terms in $\mathbf{O}_i$ are finite, since this implies that two processes which are not testing equivalent can be distinguished by finite experiments. This fact will be very important later (Section 2.6) when we need to prove that all the relations $\simeq_i^c$ satisfy some inductive properties.

The similarities of all the proofs which allow for alternative characterizations of testing equivalences for both CCS and LTS's suggest that also for LTS's we can define minimal sets of experimenters. The only reason we did not describe these experimenters in Chapter 1 was the fact that we had not yet introduced appropriate operators to define the minimal classes.

2.4. Three proof systems for CCS

In Section 2.2 we established a number of properties of testing equivalences and congruences for CCS. In Section 2.3 we presented their alternative characterizations independent of observers. In this section, we study their equational characterization. First we will present three sets of axiom schemata, whose variables may be replaced by arbitrary agents of CCS, and then three "complete" proof systems based on these schemata. Such systems are of both theoretical and practical interest. They give additional insight into the identification induced by testing equivalences and may form the basis for automatic proof systems for CCS and related languages.

Our "complete" proof systems have a major drawback: they are not recursively enumerable and thus they are not effective. However for CCS there can never be a complete recursive equational axiomatization, i.e. an axiomatization from which all and only the valid instances of testing equivalences follow by equational reasoning. In fact, it has been pointed out that a Turing machine can be simulated in pure CCS (in chapter 1 of /Mil80/ it is shown how to implement a pushdown automaton and we know that a Turing tape can be simulated by two such automata). Moreover, this simulation may be carried out so that a Turing machine TM will diverge on a blank input tape if and only if its translation $[\![TM]\!]$ is such that $[\![TM]\!] \simeq_i^+ \Omega$, i = 1, 2 or 3. We will give complete systems which are not recursively enumerable. These are of considerable interest in themselves. For example, they show that the axioms $\mathbf{A}_i$ are complete for finite terms and if we add a sufficiently powerful form of induction we get completeness for arbitrary (closed) terms. It is the required form of induction which introduces the non-recursive enumerability.

We shall now examine the axiom systems for the three relations $\simeq_i^+$, defined in the previous sections for CCS. The basic axioms are given in Table 2.1. Most of them are given in terms of "=", and they are designed to be used in conjunction with the rules:

$X = Y$ implies $X \sqsubseteq Y$, $Y \sqsubseteq X$

$X \sqsubseteq Y$, $Y \sqsubseteq X$ implies $X = Y$.

The axioms A1-A4, S1-S3, H1-H3, C1, Ω1, Ω2 are essentially taken from /Mil80/ and /HP80/. The summation notation used in C1 is justified by the axioms A1- A4. As in /Mil80/, $\Sigma\{t_i \mid i \in \emptyset\}$ denotes NIL. The notation $t\{+\Omega\}$ is meant to indicate that the term Ω is optional as a summand of the term t. The axioms of particular interest are N1-N4, which replace the "τ-laws" of /Mil80/. Indeed the new axioms imply these τ-laws.

Let $\mathbf{A_1}$ denote the set of axioms in Table 2.1 other than E1 and F1. Let $\mathbf{A_2}$ be the set $\mathbf{A_1}$ together with E1 and $\mathbf{A_3}$ be the set $\mathbf{A_1}$ together with F1. We write $t \sqsubseteq_i u$ $(t =_i u)$ if $t \sqsubseteq u$ $(t = u)$ can be derived from the axioms $\mathbf{A_i}$. These axioms are rather low-level but we can derive various others from them. A list of some important derivable axioms is given in Table 2.2. We will now derive the new axioms and re-examine the examples of the previous section.

D1 - $\tau X + \Omega \sqsubseteq X + \Omega$ — from N4.

Conversely

$X + \Omega \sqsubseteq X + \tau X + \Omega$ — from Ω1

$\sqsubseteq \tau(X + X) + \Omega$ — from N2

$= \tau X + \Omega$ — from A1.

D2 - We use induction on the size of I:

i) $|I| = 1$.

Then $\mu X = \mu X + \mu X$ — from A1

$= \mu(\tau X + \tau X)$ — from N1

$= \mu\tau X$ — from A1

ii) $I = J \cup \{0\}$

Then $\mu(\Sigma\{\tau X_i \mid i \in I\}) = \mu(\Sigma\{\tau X_i \mid i \in J\} + \tau X_0)$

$= \mu(\tau\Sigma\{\tau X_i \mid i \in J\} + \tau X_0)$ — using induction with $\mu = \tau$

$= \mu(\Sigma\{X_i \mid i \in J\}) + \mu X_0$ — from N2

$= \Sigma\{\mu X_i \mid i \in J\} + \mu X_0$ — using induction.

$X + X = X$	A1
$X + Y = Y + X$	A2
$X + (Y + Z) = (X + Y) + Z$	A3
$X + NIL = X$	A4
$\mu X + \mu Y = \mu(\tau X + \tau Y)$	N1
$X + \tau Y \sqsubseteq \tau(X + Y)$	N2
$\mu X + \tau(\mu Y + Z) = \tau(\mu X + \mu Y + Z)$	N3
$\tau X \sqsubseteq X$	N4
$NIL[S] = NIL$	S1
$(X + Y)[S] = X[S] + Y[S]$	S2
$\mu X[S] = S(\mu)X[S]$	S3
$NIL\backslash l = NIL$	H1
$(X+Y)\backslash l = X\backslash l + Y\backslash l$	H2
$(\mu X)\backslash l = \mu(X\backslash l)$ if $\mu \notin \{l, \bar{l}\}$; NIL otherwise	H3

Let t denote $\Sigma\{\mu_i t_i \mid i \in I\}\{+\Omega\}$ and u denote $\Sigma\{\mu'_j t'_j \mid j \in J\}\{+\Omega\}$ then

$t|u = \Sigma\{\mu_i(t_i|t') \mid i \in I\} + \Sigma\{\mu'_j(t|t'_j) \mid j \in J\} +$

$\Sigma\{\tau(t_i|t'_j) \mid \mu_i = \bar{\mu}_j\}$ {Ω | Ω is a summand of t or t'} — C1

$\Omega \sqsubseteq X$	Ω1
$\Omega[S] = \Omega$	Ω2
$\Omega\backslash l = \Omega$	Ω3
$t[rec\ x.\ t/x] = rec\ x.t$	REC
$\tau X + \tau Y \sqsubseteq X$	E1
$X \sqsubseteq \tau X + \tau Y$	F1

TABLE 2.1: Axioms for CCS

$\tau X + \Omega = X + \Omega$	D1
$\mu\,\Sigma\{\tau X_i \mid i \in I\} = \Sigma\{\mu X_i \mid i \in I\}, \quad I \neq \emptyset$	D2
$\mu X + \Omega = \mu(X + \Omega) + \Omega$	D3
$X + \tau Y = \tau(X + Y) + \tau Y$	D4
$\mu X + \mu Y = \mu X + \mu Y + \mu(X + Y)$	D5
$\mu X + \mu(X + Y + Z) = \mu X + \mu(X + Y) + \mu(X + Y + Z)$	D6
$\mu\tau X = \mu X$	D7
$X + \tau(X + Y) = \tau(X + Y)$	D8
$\mu(X + \tau Y) + \mu Y = \mu(X + \tau Y)$	D9
$\mu\Sigma\{t_i + \Omega \mid i \in I\} + \Omega = \Sigma\{\mu t_i \mid i \in I\} + \Omega, \quad I \neq \emptyset$	D10
$\mu(X + \Omega) = \mu(X + \Omega) + \mu\Omega$	D11
$X + \Omega = \Omega$	E2
$\tau X + \tau Y \sqsubseteq \tau X$	E3
$X + \Omega = X$	F2
$X = \tau X$	F3

TABLE 2.2: Derivable Axioms

D3 - $\mu(X + \Omega) + \Omega \sqsubseteq \mu(X + X) + \Omega$ from Ω1

$= \mu X + \Omega$ from A1.

Conversely

$\mu X + \Omega \sqsubseteq \mu X + \mu(X + \Omega) + \Omega$ from Ω1

$= \mu(\tau X + \tau(X + \Omega)) + \Omega$ from N2

$\sqsubseteq \mu(\tau X + X + \Omega) + \Omega$ from N4

$\sqsubseteq \mu(\tau(X + X) + \Omega) + \Omega$ from N2

$= \mu(\tau X + \Omega) + \Omega$ from A1

$= \mu(X + \Omega) + \Omega$ from D1

D4 - $X + \tau Y = X + \tau Y + \tau Y$ from A1

$\sqsubseteq \tau(X + Y) + \tau Y$ from N2.

Conversely

$\tau(X + Y) + \tau Y \sqsubseteq X + Y + \tau Y$ from N4

$\sqsubseteq X + \tau Y$ from N2, A1.

D5 - This is Example 1 of the previous section

$\mu X + \mu Y = \mu(\tau X + \tau Y)$ from N1

$= \mu(\tau X + \tau(X + \tau Y))$ from D4

$= \mu(\tau X + \tau(\tau(X + Y) + \tau Y))$ from D4

$= \mu(\tau X + \tau(X + Y) + \tau Y)$ from N1

$= \mu X + \mu(X + Y) + \mu Y$ from D2.

D6 - Example 2.

i) $\mu = \tau$

$\tau X + \tau(X + Y + Z) = \tau(X + Y + Z + \tau X) + \tau(X + Y + Z)$ from D4

$= \tau(X + Y + Z + \tau X + \tau(X + Y)) + \tau(X + Y + Z)$ from D4, A1

$= \tau(\tau(X + Y + Z) + \tau X + \tau(X + Y)) + \tau(X + Y + Z)$ from D4, A1

$= \tau(X + Y + Z) + \tau X + \tau(X + Y)$ from D2, A1.

ii) $\mu = a$

$aX + a(X + Y + Z) = a(\tau X + \tau(X + Y + Z))$ from N1

$= a(\tau X + \tau(X + Y) + \tau(X + Y + Z))$ from i)

$= aX + a(X + Y) + a(X + Y + Z)$ from D2.

Note that in D5, in order to prove equality between terms which contain no occurrence of τ, we first introduce τ's, then use some τ-laws and finally we eliminate τ's. The same method is used in D6, where it is seen that D2 can be used to translate τ-properties into a-properties.

D7 - An instance of D2

D8 - $X + \tau(X + Y) = \tau(X + X + Y) + \tau(X + Y)$ from D4
$= \tau(X + Y)$ from A1.

D9 - $\mu(X + \tau Y) + \mu Y = \mu(\tau(X + \tau Y) + \tau Y)$ from N1
$= \mu(\tau(\tau(X + Y) + \tau Y) + \tau Y)$ from D4
$= \mu(\tau(X + Y) + \tau Y + \tau Y)$ from N1
$= \mu(\tau(X + Y) + \tau Y)$ from A1
$= \mu(X + \tau Y)$ from D4.

These last three derived rules are the τ-laws of /HM84/.

D10 - We use induction on the size of I. If $|I| = 1$ then D10 coincides with D3. Otherwise we can write $\Sigma\{X_i \mid i \in I\}$ as $X_0 + Y$ where $Y = \Sigma\{X_i \mid i \in J\}$. Then

$\Sigma\{\mu X_i \mid i \in I\} + \mu X_0 + \Omega = \mu(Y + \Omega) + \mu X_0 + \Omega$ by induction
$= \mu(\tau(Y + \Omega) + \tau X_0) + \Omega$ from N1
$= \mu(Y + X_0 + \Omega) + \Omega$ from D1

D11 - $\mu(X + \Omega) = \mu(X + \Omega) + \mu\Omega$

$\mu(X + \Omega) \sqsubseteq \mu(X + \tau\Omega + \tau\Omega)$ from Ω1 and A1
$\sqsubseteq \mu(\tau(X + \Omega) + \tau\Omega)$ from N2
$\sqsubseteq \mu(X + \Omega) + \mu\Omega$ from N1.

Conversely

$\mu(X + \Omega) + \mu\Omega = \mu(\tau(X + \Omega) + \tau\Omega)$ from N1
$\sqsubseteq \mu(X + \Omega + \Omega)$ from N4
$= \mu(X + \Omega)$ from A1.

E2 - This is derived from the set of axioms $\mathbf{A_2}$.

$\Omega \sqsubseteq X + \Omega$ is an instance of Ω1.

Conversely

$X + \Omega \sqsubseteq X + \tau X + \tau\Omega$ from Ω1
$= \tau X + \tau\Omega$ from D8, A1
$\sqsubseteq \Omega$ from E1.

E3 - This is again derived from $\mathbf{A_2}$.

$\tau X + \tau Y = \tau\tau X + \tau Y$ from D7
$\sqsubseteq \tau X$ from E1

F2 - A derived rule from $\mathbf{A_3}$.

$X + \Omega \sqsubseteq X$ from Ω1, A1.

Conversely

$X \sqsubseteq \tau X + \tau\Omega$ from F1
$\sqsubseteq \tau X + \Omega$ from N4
$\sqsubseteq X + \Omega$ from N4.

F3 - Another derived rule from $\mathbf{A_3}$.

$\tau X \sqsubseteq X$ is N4
$X \sqsubseteq \tau X + \tau X$ from F1
$\sqsubseteq \tau X$ from A1

Example 5 - Derivable in $\mathbf{A_1}$.

$ab X + ab Y = a(\tau b X + \tau b Y)$ from N1
$= a(bX + \tau bX + bY + \tau bY)$ from D8
$= a(\tau(bX + bY) + \tau(bX + bY))$ from N3 twice
$= a\tau(bX + bY)$ from A1
$= a(bX + bY)$ from D7
$= ab(\tau X + \tau Y)$ from N1.

$\mathbf{A_3}$ deserves some additional comments. In fact, using the derived axiom F3 in N1 we obtain $\mu X + \mu Y = \mu(X + Y)$ and indeed we have that the axioms N1 - N4, F1 may be replaced by:

$$X = \tau X$$

$$\mu X + \mu Y = \mu(X + Y)$$

$$X \sqsubseteq X + Y$$

However, our presentation has the advantage that it shows the duality between the two systems $\mathbf{A_2}$ and $\mathbf{A_3}$ and how they are obtained from $\mathbf{A_1}$.

As mentioned previously, in order to extend our axiom systems to cope with infinite terms, we need some results which allow to determine the behaviour of infinite terms as the limit of the behaviour of finite ones. In particular, we will show that it is possible to determine the behaviour of any process as the limit of the behaviour of all the finite processes which approximate it. The rest of the section will be dedicated to discussing sets of approximants and to present a set of inference rules which permit properties of processes to be derived from previously known ones.

A recursively defined arbitrary term t may be considered as a finite notation for an infinite tree. This tree is obtained by "unwinding" the recursive term via the rewrite rule:

$$\mathrm{rec}\ x.u \rightarrow u[\mathrm{rec}\ x.u/x].$$

We are interested in the domain of finite trees which approximate this unwinding of t. It may be defined as follows.

Let $<$ be the least relation between terms which satisfies:

$\Omega 1$ $\quad \Omega < X$

REC1 $\quad t[\mathrm{rec}\ x.t/x] < \mathrm{rec}\ x.t$ and

i. $t_i < u_i, 0 \le i \le k$ <u>implies</u> $op(t_1,..,t_k) < op(u_1,..,u_k)$ for every $op \in \Sigma_k$

ii. $t < t$

iii. $t < u$ and $u < r$ <u>implies</u> $t < r$.

We let $FIN(t) = \{d \mid d \in FREC_\Sigma \text{ and } d < t\}$. Since i.-iii. above amounts to say that $<$ is a congruence, the relation $<$ will be referred to as the least pre-congruence (wrt Σ) generated by the axioms $\Omega 1$, REC1.

These sets of finite approximants have been studied at length in /Gue81/, /GTW77/ and /CN76/. For example FIN(t) it is proved to be directed with respect to the relation $<$ (see e.g. /Gue81/ Proposition 3.18). The unwinding of terms may also be defined in the following way:

i) $t^0 = \Omega$

ii) $(\mathrm{rec}\ x.t)^{n+1} = t[\mathrm{rec}\ x.t)^n/x]$

iii) $op(t_1, \ldots, t_k)^{n+1} = op(t_1{}^{n+1}, \ldots, t_k{}^{n+1})$.

The next lemma relates these two sets of syntactic approximants.

<u>Lemma 2.4.1</u> If $d \in FIN(t)$ then there exists $n \ge 0$ such that $d < t^n$.

<u>Proof:</u> See /Gue81/, Theorem 3.34. □

We are now ready to give the proof systems for CCS. They will be given in terms of a set of rules of the form:

$$\frac{S}{S'}$$

where S, S' are sets of statements. These rules are to be interpreted as: if every

statement in S (the set of premises) can be derived, then any statement in S' (the set of conclusions) can be derived.

R1 (**EQUALITY**)

$$\frac{t = u}{t \sqsubseteq u,\ u \sqsubseteq t} \qquad \frac{t \sqsubseteq u,\ u \sqsubseteq t}{t = u}$$

R2 (**PARTIAL ORDER**)

$$\frac{t \sqsubseteq u,\ u \sqsubseteq r}{t \sqsubseteq r} \qquad \frac{}{t \sqsubseteq t}$$

R3 (**SUBSTITUTIVITY**)

i) $\dfrac{t \sqsubseteq u}{t\rho \sqsubseteq u\rho}$ ii) $\dfrac{t \sqsubseteq u}{\text{rec x. } t \sqsubseteq \text{rec x. } u}$

iii) $\dfrac{t_i \sqsubseteq u_i,\ 1 \leq i \leq k}{op(t_1, \ldots, t_k) \sqsubseteq op(u_1, \ldots, u_k)}$

R4 (**GENERAL INDUCTION**)

$$\frac{d \sqsubseteq u, \text{ for all } d \in FIN(t)}{t \sqsubseteq u}$$

We write $\mathbf{A} \vdash t \sqsubseteq u$ if $t \sqsubseteq u$ can be derived from the set of axioms **A** using the rules R1-R4. Note that R4 is an infinitary (non-recursively enumerable) rule since it has an infinite number of premises. Recursively enumerable proof systems can be obtained by replacing it by a finitary form of induction such as Fixpoint Induction or Scott Induction /Sto77/. Indeed these may be derived from R4. As a simple example, we derive Fixpoint Induction:

FP) $\dfrac{t[u/x] \sqsubseteq u}{\text{rec x. } t \sqsubseteq u}$

Lemma 2.4.2 If **A** contains the axioms Ω1, REC1 then FP is a derived rule in the system with rules R1-R4 and axioms **A**.

Proof: Let r denote rec x. t. We first show that $\mathbf{A} \vdash r^n \sqsubseteq u$ for every $n \geq 0$.

i) $n = 0$: $\mathbf{A} \vdash r^0 \sqsubseteq u$ follows from Ω1;

ii) $n = k+1$: We assume $\mathbf{A} \vdash r^k \sqsubseteq u$.

Then

$\mathbf{A} \vdash r^{k+1} = t[r^k/x]$

$\sqsubseteq t[u/x]$ by repeated applications of R3 and induction on k

$\sqsubseteq u$ from the premise.

Also note that if $t < u$ then $\mathbf{A} \vdash t \sqsubseteq u$ since **A** contains the axioms which generate $<$. Now let $d \in FIN(r)$. Then, from lemma 2.4.1, $\mathbf{A} \vdash d \sqsubseteq r^n$ for some n and therefore $\mathbf{A} \vdash d \sqsubseteq u$. Since this is true of every $d \in FIN(r)$ we may apply R4 to obtain $\mathbf{A} \vdash r \sqsubseteq u$. □

Lemma 2.4.3 If **A** contains the axioms Ω1 and REC1 then the rule R3 ii) is derivable.

Proof: We assume that $\mathbf{A} \vdash t \sqsubseteq u$. Let ρ be the substitution defined by

$\rho(x) = \text{rec x.u}$

$\rho(y) = y,\ y \neq x.$

Then applying the first part of R3, we get $\mathbf{A} \vdash t\rho \sqsubseteq u\rho$, i.e.

$\mathbf{A} \vdash t[\text{rec x.u}/x] \sqsubseteq u[\text{rec x.u}/x]$

$\sqsubseteq \text{rec x.u}$ using REC1

Now applying FP we get $\mathbf{A} \vdash \text{rec x. } t \sqsubseteq \text{rec x. } u$. □

We decided to include this rule as part of R3 for the sake of clarity. We now state the main theorems of the chapter.

Theorem 2.4.4 (**Soundness**)

For $i = 1, 2, 3$ $\mathbf{A}_i \vdash t \sqsubseteq u$ $(t=u)$ implies $t \sqsubseteq_i^c u$ $(t \simeq_i^c u)$ □

Theorem 2.4.5 (Completeness)

For $i = 1, 2, 3$ $t \sqsubseteq_i^C u$ $(t \simeq_i^C u)$ implies $\mathbf{A}_i \vdash t \sqsubset u$ $(t=u)$ □

The next two sections are devoted entirely to the proofs of these results.

2.5. Detailed Proofs for Finitary CCS

In this section we prove soundness of the three sets of axioms presented in the previous one and prove their completeness for a finitary subset of CCS, i.e. CCS without recursively defined terms. To deal with terms which contain the recursive operator rec x. it is natural to use some form of induction. The proof systems presented above and in particular the rule R4 will serve this purpose. The generalization to infinite terms will be discussed in Section 2.6.

2.5.1 Soundness

In this subsection we concentrate on proving the axioms of table 2.1 sound. We will mainly use the alternative characterization of $\sqsubseteq_i^+$ given in Theorem 2.3.6 and the previous lemmas. In the following we write $t \sqsubseteq_i t'$ if $t \sqsubset t'$ can be derived from the set of axioms $\mathbf{A}_i$ using the rules R1, R2 and R3 i)- iii).

Lemma 2.5.1

If $op \in \Sigma_k$ and $t_j \sqsubseteq_i^+ t'_j$ for all j, $1 \leq j \leq k$ then $op(t_1,..t_k) \sqsubseteq_i^+ op(t'_1,..t'_k)$

Proof: It is sufficient to give the proof for closed terms. Each of the operators are examined in turn. Cases NIL and Ω are trivial. We will rely on the alternative

characterization of Theorem 2.3.6 for the proofs relative to all operators apart from "|". In this case we shall exploit some distinctive features of the original definition of the various preorders. We prove the claim only for i = 2 and 3 since the case i = 1 follows automatically.

a) $p \sqsubseteq_i^+ q$ implies $p|r \sqsubseteq_i^+ q|r$ $i=1,2,3$

i=3) It is sufficient to show that $r' + (p|r)$ may satisfy o implies $r' + (q|r)$ may satisfy o. From the hypothesis $(r' + (p|r))|o =\varepsilon\Rightarrow p_1$, $p_1 \in$ Success. If this computation does not involve the subterm p|r then we will immediately have $r' + (q|r)|o =\varepsilon\Rightarrow p'_1$ for some $p'_1 \in$ Success. Otherwise $(p|r)|o=\varepsilon\Rightarrow p_1$. Therefore $p|(r|o)=\varepsilon\Rightarrow p'_1$ for some $p'_1 \in$ Success. Since $p \sqsubseteq_3 q$, the latter implies $q|(r|o) =\varepsilon\Rightarrow q'_1$ for some $q'_1 \in$ Success , because $(r|o)$ is an observer. Therefore $(q|r)|o =\varepsilon\Rightarrow q_1$ for some $q_1 \in$ Success.

i=2) It is sufficient to show that $r' + (p|r)$ must satisfy o, implies that $r' + q|r$ must satisfy o. We distinguish two cases.

i) p|r must satisfy o

This implies p must satisfy r|o and, since $p \sqsubseteq_2 q$, q must satisfy r|o.This implies q|r must satisfy o. It follows that $r' + q|r$ must satisfy o since every computation from $(r' + q|r)|o$ which starts with an action or communication from r' is also a computation from $(r' + p|r)|o$.

ii) p|r must satisfy o.

If $p|r\text{-}\tau\rightarrow$ then $r + p|r$ must satisfy o implies that p|r must satisfy o. Therefore, we may assume $p|r\text{-}\tau\not\rightarrow$ and we have $a + p$ must satisfy $\bar{a}w + r$ for any a which does not appear in p, r. Since $p \sqsubseteq_2^+ q$ then $a + q$ must satisfy $\bar{a}w + r$, i.e. $q|r\text{-}\tau\not\rightarrow$. By a simple case analysis on r' we can now establish that $r' + q|r$ must satisfy o.

b) $p \sqsubseteq_i^+ q$ implies $\mu p \sqsubseteq_i^+ \mu q$

i=3) We have to distinguish two cases: i. $\mu = \tau$ and ii. $\mu = a$ for some $a \in A$. In case i. we have that, for any process r, traces(μr) = traces(r) and in case ii) we have traces(μr) = {ε} ∪ {μs | s ∈ traces(r)}. In both cases it is not difficult to see that traces(p) ⊆ traces(q) implies traces(μp) ⊆ traces(μq) and $\mu p \sqsubseteq_3^+ \mu q$ follows from theorem 2.3.6.

i=2) We certainly have $q\text{-}\tau\rightarrow$ only if $p\text{-}\tau\rightarrow$, we are left to prove that $\mu p\Downarrow s$ implies $\mu q\Downarrow s$ and if (μp after s) MUST L then (μq after s) MUST L given that the same holds for p and q. We have $p\Downarrow s$ implies $q\Downarrow s$ and so $\mu p\Downarrow s$ implies $\mu q\Downarrow s$ for all $s \in A^*$, since $p\Downarrow s$ implies $\mu p\Downarrow \mu s$. To prove the rest we have again to distinguish two cases: $\mu = \tau$ and $\mu = a$. In case $\mu = \tau$ we have μp after s = p after s and μq after s = q after s for any s, thus the claim follows by hypothesis. If $\mu = a$ and $s = \varepsilon$, then clearly ap MUST L iff aq MUST L; if $\mu = a$ and $s = bs'$ for some $s' \in A^*$, then either ap after bs' = aq after bs' = $\emptyset$ or ap after bs' = p after s' and aq after bs' = q after s' and again the claim follows from the hypothesis.

c) $p \precsim_i^+ q$ implies $r + p \precsim_i^+ r+q$

i=3) It is sufficient to show that traces(r+p) $\subseteq$ traces (r+q). This follows since traces(p) $\subseteq$ traces(q) by hypothesis and traces(p_1+p_2) = traces(p_1) $\cup$ traces(p_2) for any pair of processes p_1, p_2.

i=2) Clearly $r+q\text{-}\tau\rightarrow$ implies $r+p\text{-}\tau\rightarrow$ or $(r+p)\Uparrow$ since $p\Downarrow$ and $q\text{-}\tau\rightarrow$ implies $p\text{-}\tau\rightarrow$. Suppose $(r+p)\Downarrow s$ then we have to prove $(r+q)\Downarrow s$ and (r+p) after s MUST L implies (r+q) after s MUST L. We have that $(r+p)\Downarrow s$ implies $r\Downarrow s$ and $p\Downarrow s$ and thus we have also $q\Downarrow s$ by hypothesis whence $(r+q)\Downarrow s$. Moreover for $s \neq \varepsilon$ we have (r+p) after s MUST L implies (r+q) after s MUST L since p_1+p_2 after as' = p_1 after as' $\cup$ p_2 after as' for any pair of processes p_1, p_2 and for any $a \in A$. We are left to consider only the case $s = \varepsilon$, i.e. we have to prove r+p MUST L implies r+q MUST L. We distinguish two cases:

i. $p\text{-}\tau\rightarrow$

then we have r+p MUST L implies p MUST L and then q MUST L by hypothesis. The latter together with r+p MUST L implies r+q MUST L.

ii. $p\text{-}\tau\not\rightarrow$

then $q\text{-}\tau\not\rightarrow$ and r+p MUST L implies r MUST L or p MUST L. In case r MUST L we certainly have r+q MUST L; in case p MUST L then q MUST L and this together with r+p MUST L implies r+q MUST L.

d) $p \precsim_i^+ q$ implies $p[S] \precsim_i^+ q[S]$

We can easily extend the bijective functions S from processes to sets of traces and to sets of actions. We would then have:

i=3) We need to prove traces(p[S]) = S(traces(p)). Since S is monotonic we have traces(p[S]) $\subseteq$ traces(q[S])

i=2) Suppose $p[S]\Downarrow s$ then $p\Downarrow S^{-1}(s)$ and $q\Downarrow S^{-1}(s)$, by hypothesis therefore $q[S]\Downarrow s$. Moreover, since $S(\tau) = \tau$ we have $q[S]\text{-}\tau\rightarrow$ implies $q\text{-}\tau\rightarrow$ implies $p\text{-}\tau\rightarrow$ implies $q[S]\text{-}\tau\rightarrow$. Suppose p[S] after s MUST L then we have p after $S^{-1}(s)$ MUST $S^{-1}(L)$. This by hypothesis implies q after $S^{-1}(s)$ MUST $S^{-1}(L)$, which in turn implies q [S] after s MUST L.

e) $p \precsim_i^+ q$ implies $p\backslash a \precsim_i^+ q\backslash a$

We only prove p\a after s MUST L implies q\a after s MUST L since the other cases are similar to the corresponding one in d). p\a after s MUST L implies either a is in s or $p=s\Rightarrow$ or p after s MUST L-{a}. In the first two cases, we have p\a after s = q\a after s = $\emptyset$ and the claim follows. In case p after s MUST L-{a} then q after s MUST L-{a} and q\a after s MUST L-{a} and q\a after s MUST L. □

Lemma 2.5.2

If $p\text{-}\mu\rightarrow r$ if and only if $q\text{-}\mu\rightarrow r$, and $p\downarrow$ if and only if $q\downarrow$ then $p \simeq_i^+ q$

Proof: Trivial □

Lemma 2.5.3

If $t \sqsubseteq_i t'$ ($t =_i t'$) is an instance of an axiom from $\mathbf{A_i}$, i=1, 2 or 3, then $t \precsim_i^+ t'$ ($t \simeq_i^+ t'$).

Proof: It is sufficient to consider closed terms. The previous lemma takes care of axioms A1-A4, S1-S3, H1-H3, C1 and REC. Ω1-Ω3 are trivial. We examine the remaining ones. For each inequality (equality) we consider, we will denote its left hand side by r_1 and its right hand side by r_2. Note that for N1-N4 we can easily prove that traces(r_1) = traces(r_2) thus we only need to prove that $r_1 \precsim_2^+ (\simeq_2^+) r_2$.

N1: $\mu p + \mu q \simeq_2^+ \mu(\tau p + \tau q)$

We certainly have $r_1 \Downarrow s$ if and only if $r_2 \Downarrow s$. On the other hand, we have $r_1 - \tau \rightarrow$ if and only if $r_2 - \tau \rightarrow$. We are only left to prove that if r_1 after s MUST L then r_2 after s MUST L. The only interesting cases are the ones in which either $s = \varepsilon$ and $\mu = \tau$ or $s = a$ for some $a \in A$, since in all the other cases, we have r_1 after $s = r_2$ after s.

i. $s = \varepsilon$ and $\mu = \tau$.

We have r_1 after ε = p after ε U q after ε U $\{\tau p + \tau q\}$ and r_2 after ε = p after ε U q after ε U $\{\tau p + \tau q, \tau(\tau p + \tau q)\}$ and the claim follows from the fact that $\tau(\tau p + \tau q)$ MUST L if and only if $\tau p + \tau q$ MUST L.

ii. $s = a, a \in A$.

r_1 after ε MUST L only if $a \in L$ and this implies r_2 after ε MUST L, the case r_2 after ε MUST L is treated similarly. We also have r_1 after $\mu = \{p, q\}$ and r_2 after $\mu = \{p, q\}$ U $\{\tau p + \tau q\}$ and the claim follows since $\tau p + \tau q$ MUST L if and only if p MUST L and q MUST L.

N2: $p + \tau q \sqsubseteq \tau(p + q)$

The only interesting part of the proof of this axioms consists in showing that r_1 after ε MUST L implies r_2 after ε MUST L. Suppose r_2 MUST L, then we either have p MUST L or q MUST L or both. In case q MUST L we have r_1 MUST L since $q \in r_1$ after ε. In case p MUST L and q MUST L we have that there exists p' such that $p - \tau \rightarrow p'$ and p' MUST L; this again suffices to prove that r_1 MUST L since $p' \in r_1$ after ε.

N3: $\mu p + \tau(\mu q + r) \simeq^+ \tau(\mu p + \mu q + r)$

Once again, the only interesting part is to prove that r_1 MUST L iff r_2 MUST L. If $\mu = \tau$ then we have r_1 MUST L only if p MUST L, q MUST L and either $r - \tau \rightarrow$ or r MUST L. From this it is not difficult to derive $\tau(\mu p + \mu q + r)$ MUST L. The converse is similar. If $\mu \in A$ then r_1 MUST L only if either r MUST L or $\mu \in L$ and $r - \tau \rightarrow$. In both these cases we can conclude r_2 MUST L. The converse is similar.

N4: $\tau p \sqsubseteq_1 p$

The proof is trivial since $r_2 \in r_1$ after ε.

E1: $\tau p + \tau q \sqsubseteq_2^+ \tau p$

Equally trivial.

F1: $p \sqsubseteq_3 \tau p + \tau q$

We certainly have traces(p) $\subseteq$ traces(p) U traces(q). □

Proposition 2.5.4 (Soundness)

a) $t \sqsubseteq_1 t'$ implies $t \sqsubseteq_1^+ t'$

b) $t =_1 t'$ implies $t \simeq_1^+ t'$

Proof: The proof follows from the Lemmas 2.5.2 and 2.5.3. The soundness of R1 and R2 being immediate. □

2.5.2 Normal Forms

The proof of completeness relies (like most of such proofs) upon reducing CCS terms to particular canonical forms. These are restricted kinds of terms which reveal the essence of testing equivalences better than the actual definitions of these equivalences. Normal forms give evidence that testing a process is an operation which may be logically divided into two phases, which alternate during the experimentation. In one phase, the control is in the hand of the process which is being experimented, in the other it is left completely to the experimenter. In fact, the canonical forms are trees whose nodes either have outgoing arcs labelled only by silent moves or only by distinct visible actions. Local and global nondeterminism, /FHLR79/, are clearly distinguished. The different kinds of canonical forms will also shed light on the different stress on divergence put by the three preorders, $\sqsubseteq_i$, and by the three powerdomain constructions of Chapter 1. In fact, for each preorder we will have a corresponding normal form.

Before defining our canonical forms, which we will call **normal forms**, we need some basic definitions. The following definition, which establishes a kind of convexity condition on sets of sets, will be used many other times throughout the thesis.

Definition 2.5.5 (Saturated sets)

If $\mathbf{L}$ is a non-empty finite set of finite sets and $A(\mathbf{L}) = \{a \mid a \in L \text{ for some } L \in \mathbf{L}\}$ then $\mathbf{L}$ is said to be **saturated** if $\forall K \subseteq A$, $(L \in \mathbf{L}$ and $L \subseteq K \subseteq A(\mathbf{L}))$ implies $K \in \mathbf{L}$. □

Note that if $\mathbf{L}$ is a saturated set then we will have:

i. $A(\mathbf{L}) \in \mathbf{L}$

ii. $X \in \mathbf{L}$ and $Y \in \mathbf{L}$ implies $X \cup Y \in \mathbf{L}$

iii. $X \in \mathbf{L}$ and $Y \in \mathbf{L}$ and $X \subseteq Z \subseteq Y$ implies $Z \in \mathbf{L}$

In fact conditions ii. and iii. are sufficient to characterize a saturated set.

We will use the notion of saturated sets in the definition of normal forms below. It turns out to be a crucial notion to get unique normal forms for equivalent agents.

Definition 2.5.6 Normal Forms (nf)

p is in **normal form** if it is in the form

a. $\Sigma\{\tau\,\Sigma\{ap(a) \mid a \in L\} \mid L \in \mathbf{L}\}$ or

b. $\Sigma\{ap(a) \mid a \in L\}$ $[+\,\Omega]$

where

i. $\mathbf{L}$ is a saturated set

ii. $[+\Omega]$ denotes that Ω is an optional summand and if Ω is a summand then $p(a)$ is a normal form in the form b. and Ω is a summand of $p(a)$ for all $a \in A(\mathbf{L})$.

iii. If Ω is not a summand then $p(a)$ is a normal form in the form a. . □

In the following we will refer to normal forms as nf and to normal forms in the form a. and b. as τ-nf and a-nf, respectively.

Examples

i) $a(a+b) + \Omega$ is not in nf because ii. is not satisfied and the subterm $(a+b)$ is not in τ-nf.

ii) $\tau a_1 \tau a + \tau a_2 \tau b$ is not in nf because $\{\{a_1\}\{a_2\}\}$ is not saturated.

There is no a-subterm corresponding to the set of labels $k=\{a_1, a_2\}$. The normal form corresponding to this term will be $\tau a_1 \tau a + \tau a_2 \tau b + \tau(a_1 \tau a + a_2 \tau b)$.

iii) $\tau a \tau b + \tau(a \tau c + a \tau b)$ is not in normal form.

If p is a normal form and for any a, $p \Longrightarrow -a\rightarrow p_1$ and $p \Longrightarrow -a\rightarrow p_2$ then p_1 and p_2 must be identical. This example violates this requirement. Because of this property, we can use a suggestive notation: for a normal form p we may let p(a) denote the unique subterm (if it exists) such that $p=a\Rightarrow p(a)$.

iv) NIL is an a-nf. In the definition we merely let $L = \emptyset$. Similarly τNIL is a τ-nf and Ω is an a-nf. On the other hand it should be noted that $\tau\Omega$ is not a normal form.

From the definition, we have that if p is in normal form and $p\overset{a}{\not\rightarrow}$ then Ω is a summand of p and p is in a-nf. Also, every normal form is a finite term and if p is a normal form and $p=a\Rightarrow p'$ then p' is also in normal form.

Together with normal forms, we have two other kinds of "canonical" forms for CCS terms. As mentioned above they stress the different approaches one can take when dealing with partially specified or divergent terms.

Definition 2.5.7 Strong normal forms (snf)

p is in **strong normal form** if it is in normal form and

i. p(a) is in strong normal form

ii. if Ω is a summand of p then p is Ω. □

Definition 2.5.8 Weak normal forms (wnf)

p is in **weak normal form** if it is in normal form and

i. p(a) is a weak normal form

ii. Ω is a summand of p

□

If we look upon terms as trees then an snf is a normal form which only contains Ω as a leaf. On the other side, a wnf is a normal form such that the arcs outgoing from every node are all labelled by distinct visible actions. Later we will see that indeed a wnf corresponds to a prefix-closed set of strings from A.

In the rest of this subsection, we will show that every finite closed term can be transformed to a normal form, a strong normal form or a weak normal form by only using axioms from $\mathbf{A_1}$, $\mathbf{A_2}$ or $\mathbf{A_3}$, respectively. The proof will be based on showing that any CCS term can be transformed into a term which only contains the operators NIL, Ω, τ, and +. Then, a series of lemmas will guarantee that if we use only these operators to combine normal forms, we can always find a normal form provably equivalent to the starting term.

Lemma 2.5.9 If p in nf then there exists an a-nf n such that $p + \Omega =_1 n$ and nf.

Proof:

a) Suppose $p =_1 \Sigma\{\tau\Sigma\{ap(a) \mid a \in L\} \mid L \in \mathbf{L}\}$

Then $p + \Omega =_1 \Sigma\{\tau\Sigma\{ap(a) + \Omega \mid a \in L\} \mid L \in \mathbf{L}\} + \Omega$ using D3

By induction there exists an a-nf n(a) such that $p(a) + \Omega =_1 n(a)$. Therefore,

$p + \Omega =_1 \Sigma\{\tau\Sigma\{an(a) \mid a \in L\} \mid L \in \mathbf{L}\} + \Omega$

$=_1 \Sigma\{an(a) \mid a \in A(\mathbf{L})\} + \Omega$ using D1 and D3

b) The proof for p in a-nf is similar. □

Lemma 2.5.10 If p, q are in nf then there exists a normal form n such that $\tau p + \tau q =_1 n$. Moreover, either n is a τ-normal form or else nfl.

Proof: We use induction on the size of p and q. There are four cases depending on what kind of normal forms p and q are.

i) p is $\Sigma\{\tau p_i \mid i \in I\}$, q is $\Sigma\{\tau p_j \mid j \in J\}$

Then $\tau p + \tau q =_1 \Sigma\{\tau p_i \mid i \in I\} + \Sigma\{\tau p_j \mid j \in J\}$ using D2.

Let r denote the right hand side. Now r may not be in normal form for various reasons. For example, it may be that $r \Rightarrow -a\rightarrow r_1$, $r \Rightarrow -a\rightarrow r_2$ and r_1, r_2 are not syntactically identical. Let N(r) be the number of such pairs. We show, by induction on N(r), that r can be transformed into an r' such that N(r') = 0. If r_1, r_2 is such a pair then the axioms A1-A4 may be used to rewrite r as

$r =_1 \tau(ar_1 + r'_1) + \tau(ar_2 + r'_2) + r'$

$=_1 ar_1 + \tau(ar_1 + r'_1) + ar_2 + \tau(ar_2 + r'_2) + r'$ using D8

$=_1 \tau(ar_1 + ar_2 + r'_1) + \tau(ar_1 + ar_2 + r'_2) + r'$ using N3 twice

$=_1 \tau(a(\tau r_1 + \tau r_2) + r'_1) + \tau(a(\tau r_1 + \tau r_2) + r'_2) + r'$ using N1 twice.

Now by induction $\tau r_1 + \tau r_2$ has a normal form n of the required form. Therefore $r =_1 \tau(an + r'_1) + \tau(an + r'_2) + r'$. If s denotes the right hand side of the latter equality, then $N(s) < N(r)$. Therefore, by induction we can now assume that r is such that N(r) = 0. Thus r can now be written as $\Sigma\{\tau\Sigma\{ar(a) \mid a \in L\} \mid L \in \mathbf{L}\}$. Moreover, by hypothesis r(a) is in τ-nf or is in a-nf and diverges. Thus if this term is not in nf it must be that $\mathbf{L}$ is not saturated. Let K be such that $L_j \subseteq K \subseteq A(\mathbf{L})$.

By induction on the size of K, we can show, that $r =_1 r + \Sigma\{an(a) \mid a \in K\}$. If $K = L_j$ then the claim is immediate. Otherwise, K can be written as $K_1 \cup \{a_0\}$, a_0 belonging to some $L_i \in \mathbf{L}$. We can assume $r =_1 r + \tau\Sigma\{an(a) \mid a \in K_1\}$ and let r_1 denote $\Sigma\{an(a) \mid a \in K_1\}$. Then

$r =_1 r + \tau r_1 + \tau(r' + a_0 n(a_0))$ using A1-A3

$=_1 r + \tau r_1 + \tau(r' + a_0 n(a_0)) + \tau(r_1 + r' + a_0 n(a_0))$ using D5

$=_1 r + \tau r_1 + \tau(r' + a_0 n(a_0)) + \tau(r_1 + r' + a_0 n(a_0)) + \tau(r_1 + a_0 n(a_0))$ using D6

$=_1 r + \tau(r_1 + a_0 n(a_0))$ as required.

Therefore, by induction, we can assume that

$r =_1 r + \tau\,\Sigma\{an(a) \mid a \in K\}$ for every such k.

Now, by systematically applying this result, r may be transformed into a term of the form $\Sigma\{\tau\,\Sigma\{an(a) \mid a \in L\} \mid L \in \mathbf{L}\}$ where $\mathbf{L}$ is saturated, and this term is in τ-nf.

ii) p is $\Sigma\{a_i p_i \mid i \in I\}$, q is $\Sigma\{\tau p_j \mid j \in J\}$

Then $\tau p + \tau q =_1 \tau p + q$ using D2

and we proceed as in part i).

iii) p is $\Sigma\{a_i p_i \mid i \in I\}$, q is $\Sigma\{a_j q_j \mid j \in J\}$

Then $\tau p + \tau q$ is an instance of i).

iv) p is $\Sigma\{ap(a) \mid a \in L\} + \Omega$. Then $\tau p + \tau q = p + q$ by D1, D3.

From Lemma 2.5.9 there exists an a-nf n such that $q + \Omega =_1 n$.

Let n be $\Omega + \Sigma\{an(a) \mid a \in K\}$. Then

$p + n =_1 \Omega + \Sigma\{ap(a) \mid a \in L-K\} + \Sigma\{an(a) \mid a \in K-L\} +$

$\Sigma\{a(\tau p(a) + \tau q(a)) \mid a \in L \cap K\}$ using N1.

By induction, there exists a normal form r such that $r(a) =_1 \tau p(a) + \tau q(a)$.

Therefore

$p + n =_1 \Omega + \Sigma\{a(p(a) + \Omega) \mid a \in L-K\} + \Sigma\{a(n(a) + \Omega) \mid a \in K-L\} +$

$\Sigma\{a(r(a) + \Omega) \mid a \in L \cap K\}$ using D3.

We now apply Lemma 2.5.9 to obtain the required normal form. □

Lemma 2.5.11 If p, q are in nf then there exists a nf n such that $p+q =_1 n$.

Proof: We use induction on the size of p, q. There are four cases, depending on the form of p and q.

a) p is $\Sigma\{ap(a) \mid a \in L\} + \Omega$

Then $p + q =_1 \tau p + \tau q$, using D1, and we can apply the previous lemma.

b) p and q are in τ-nf.

Then $p + q =_1 \tau p + \tau q$, using D2, and again we can apply the previous lemma

c) p is $\Sigma\{ap(a) \mid a \in L\}$, q is $\Sigma\{\tau q_i \mid i \in I\}$

Then $p + q =_1 \tau(p + q_1) + \tau q$ using D4.

By induction $p + q_1$ may be transformed into a normal form n and by the previous lemma $\tau n + \tau q$ may be transformed into a normal form.

d) The only remaining case is when p is $\Sigma\{ap(a) \mid a \in L\}$, q is $\Sigma\{aq(a) \mid a \in K\}$

Then $p + q =_1 \Sigma\{ap(a) \mid a \in L-K\} + \Sigma\{aq(a) \mid a \in K-L\} +$

$\Sigma\{a(\tau p(a) + \tau p(a)) \mid a \in L \cap K\}$

From the previous lemma, each $\tau p(a) + \tau q(a)$ can be transformed into an nf of the required form and the claim follows. □

Corollary 2.5.12 If p_i are in nf, $1 \le i \le k$, then there exists an nf n such that $\Sigma\{\tau p_i \mid 1 \le i \le k\} =_1 n$, and n is either in τ-nf or else nff and is in a-nf.

Proof: By induction on k.

k=1. If p_1 is in a-nf and $p_1$ff then τp_1 is already in τ-nf. Otherwise $\tau p =_1 p_1$, using D1 or D2.

$k = m+1$. $\Sigma\{\tau p_i \mid 1 \le i \le k\} =_1 \Sigma\{\tau p_i \mid 1 \le i \le k-1\} + \tau p_k$

$=_1 n' + \tau p_k$ by induction

If n' is in τ-nf we can apply Lemma 2.5.10; otherwise $n' + \tau p_k =_1 n' + p_k$, using D1 and we may apply Lemma 2.5.11. □

Corollary 2.5.13 If p_i are in nf, $1 \le i \le k$, then there exists a nf n such that $\Sigma\{p_i \mid 1 \le i \le k\} =_1 n$.

Proof: By induction on k. □

Proposition 2.5.14 For every finite closed term d there exist a normal form nf(d) such that $d =_1 nf(d)$.

Proof: By applying the rules S1-S3, H1-H3, C1 and Ω2-Ω3 we can eliminate all occurrences of |, [S] and \a. Therefore, without loss of generality we can assume that d does

not contain these operators and can be written as $\Sigma\{a_i d_i \mid i \in I\} + \Sigma\{\tau e_j \mid j \in J\}$.

Now, by induction, we can assume that each d_i, e_j is in nf. Using D7 if necessary, each $a_i d_i$ may be considered to be in nf. Thus, from Corollary 2.5.13 there exists a normal form n' such that $n' =_1 \Sigma\{a_i d_i \mid i \in I\}$. Similarly, from Corollary 2.5.12 there exists a norm form n" such that $n'' =_1 \Sigma\{\tau e_j \mid j \in J\}$. Therefore, by applying Corollary 2.5.13 again, we get a normal form n such that $d =_1 n' + n'' =_1 n$. □

This proposition enables us to derive similar results on strong and weak normal forms. We will use the notations τ-snf and a-snf in the obvious way.

Proposition 2.5.15 For every finite closed term d there exists a strong normal form snf(d) such that $d =_2 snf(d)$.

Proof: From the previous proposition, we can assume that d is in normal form.

a) d is $\Sigma\{ad(a) \mid a \in L\}[+ \Omega]$

If Ω is a summand then $d =_2 \Omega$ from E2. Otherwise, by induction, we can assume that each d(a) is in snf. Then using D7 we have :

$d =_2 \Sigma\{ad(a) \mid a \in L, d(a) \text{ in } \tau\text{-snf or } d(a)\Uparrow\} + \Sigma\{a\tau d(a) \mid a \in L, d(a) \text{ in a-nf and } d(a)\Downarrow\}$.

b) If d is in τ-nf the proof is similar. □

Proposition 2.5.16 For every finite closed term d there exists a weak normal form w such that $d =_3 wnf(d)$.

Proof: We can assume that d is in nf.

a) d is $\Sigma\{ad(a) \mid a \in L\}[+ \Omega]$

Then $d =_3 \Sigma\{ad(a) \mid a \in L\} + \Omega$ using F2

$=_3 \Sigma\{a(d(a) + \Omega) \mid a \in L\} + \Omega$ using D3

By induction, each term of the form $d(a) + \Omega$ has a weak normal form and the result thus follows.

b) d is $\Sigma\{\tau\Sigma\{ad(a) \mid a \in L\} \mid L \in \mathbf{L}\}$

Then $d =_3 d + \Omega$

$=_3 \Sigma\{an(a) \mid a \in A(\mathbf{L})\} + \Omega$ using D1

We may now apply part a). □

Weak normal forms may also be considered as prefix-closed sets of strings from A. If $s \in A^*$ then we can also use s to denote its representation as a term: the representation of ε, the empty string is Ω, and the representation of as is at where t is the term representing s. □

Lemma 2.5.17 If d is a weak normal form then there exists a prefix-closed set of strings s_i, $1 \le i \le n$, such that $d =_1 \Omega + \Sigma\{s_i \mid 1 \le i \le n\}$.

Proof We use induction on d.

i) d is Ω. Immediate, the corresponding string is ε.

ii) d is $\Sigma\{ad(a) \mid a \in L\} + \Omega$

By induction, $d(a) =_1 \Omega + \Sigma\{s_i \mid i \in I_a\}$ where $\{s_i \mid i \in I_a\}$ is prefix-closed. Therefore,

$d =_1 \Sigma\{a(\Sigma\{s_i + \Omega \mid i \in I_a\}) \mid a \in L\} + \Omega$

$=_1 \Sigma\{a(\Sigma\{s_i + \Omega \mid i \in I_a\}) \mid a \in L\} + \Sigma\{a\Omega \mid a \in L\} + \Omega$ from D11

$=_1 \Sigma\{a(\Sigma\{s_i \mid i \in I_a\}) \mid a \in L\} + \Sigma\{a\Omega \mid a \in L\} + \Omega$ from D10 □

2.5.3 Completeness

This section is entirely dedicated to proving that the sets of axioms $\mathbf{A_i}$ of Table 2.1 completely characterize the preorders $\sqsubseteq_i$. The proofs rest heavily on the existence of normal forms and on defining orderings on them which naturally relate to the preorders.

In the following it will be convenient to let NF_1, NF_2 and NF_3 denote the set of normal forms, strong normal forms and weak normal forms, respectively. Moreover, the following

notational convention will be used:

$\mathbf{L}(n)$ will stand for $\mathbf{L}$ if n is of the form $\Sigma\{\tau\Sigma\{an(a) \mid a \in L\} \mid L \in \mathbf{L}\}$

$L(n)$ will stand for L if n is of the form $\Sigma\{an(a) \mid a \in L\}$.

The orderings on normal forms will be defined in two steps using the recursive nature of the definition of the normal forms. Firstly, we will define orderings which only compare normal forms at the topmost level; we will then define new orderings which recursively extend the first ones to compare normal forms at all levels.

Definition 2.5.18 For $d, d' \in NF_j$ let $d <_i d'$ be defined by

a. $d <_3 d'$ if $Init(d) \subseteq Init(d')$

b. $d <_2 d'$ if $d\Downarrow$ implies $d'\Downarrow$ and

i. d, d' are in a-nf and $L(d) = L(d')$

ii. d, d' are in τ-nf and $\mathbf{L}(d) \supseteq \mathbf{L}(d')$

iii. d is in τ-nf, d' is in a-nf and $L(d') \in \mathbf{L}(d)$

c. $d <_1 d'$ if $d <_2 d'$ and $d <_3 d'$ □

These preorders are able to compare normal forms at the topmost level only. We now define three new ones, $\lessdot_i$, which extend $<_i$ recursively to compare normal forms at every level.

Definition 2.5.19 For d, d' NF_j let $d \lessdot_i d'$ if

i. $d <_i d'$

ii. $d(a) \lessdot_i d'(a)$ whenever both $d(a)$ and $d'(a)$ are defined. □

Our task now is to determine the relationship between the preorders on general CCS terms and these orderings on normal form. We have:

Lemma 2.5.20 For $d, d' \in NF_j$ $d \sqsubseteq_i^+ d'$ implies $d <_i d'$.

Proof: We use the alternative characterizations of $\sqsubseteq_i^+$ given in Section 2.3.1.

i=3) Trivial, since $d \sqsubseteq_3^+ d'$ implies $Traces(d) \subseteq Traces(d')$.

i=2) Certainly we have $d\Downarrow$ implies $d'\Downarrow$. Moreover, since $d \sqsubseteq_2^+ d'$ implies $d' - \tau\rightarrow$, only if $d - \tau\rightarrow$, we cannot have that d is in a-nf and d' in τ-nf. Let us consider the remaing cases:

i. d, d' in a-nf and $d \sqsubseteq_2^+ d'$ implies $L(d) \subseteq L(d')$. In fact, if there exist $a \in L(d)$ and $a \notin L(d')$ then we would have d MUST $\{a\}$ and d' MUST $\{a\}$. On the other hand, we have $L(d) \subseteq L(d')$ from Lemma 2.3.5 and the claim follows.

ii. d, d' in τ-nf and $d \sqsubseteq_2^+ d'$ implies $A(\mathbf{L}(d)) \supseteq A(\mathbf{L}(d'))$ by Lemma 2.3.5. Since $\mathbf{L}(d)$ is saturated to prove that $\mathbf{L}(d) \supseteq \mathbf{L}(d')$ it is sufficient to show that $M \in \mathbf{L}(d')$ implies that there exists $N \in \mathbf{L}(d)$ such that $M \supseteq N$. Let us suppose there exists $M_0 \in \mathbf{L}(d')$ such that for all $N \in \mathbf{L}(d)$ there exists $a \in N$ such that $a \notin M_0$. Then if we let $L = \{a \mid a \notin M_0 \text{ and } a \in A(\mathbf{L}(d))\}$, we would have d MUST L and d' MUST L.

iii. d in τ-nf and d' in a-nf and $d \sqsubseteq_2^+ d'$ implies $A(\mathbf{L}(d)) \supseteq L(d')$ again by Lemma 2.3.5. Since $\mathbf{L}(d)$ is saturated, to prove that $L(d') \in \mathbf{L}(d)$ it is sufficient to prove that there exists $N \in \mathbf{L}(d)$ such that $L(d') \supseteq N$. This proof is similar to the corresponding one of case ii.

i=1) Follows from the previous two cases. □

Lemma 2.5.21 For $d, d' \in NF_j$, $d \sqsubseteq_i^+ d'$ implies $d(a) \sqsubseteq_i^+ d'(a)$ whenever both $d(a)$ and $d'(a)$ are defined.

Proof We use the definition of $\sqsubseteq_i^+$ in terms of observers.

i=3) $d(a)$ may satisfy o

implies d may satisfy $\bar{a}o$

implies d' may satisfy $\bar{a}o$

implies $d'(a)$ may satisfy o.

1=2) $d(a)$ must satisfy o

implies d must satisfy $\bar{a}o$

implies d' must satisfy $\bar{a}o$

implies $d'(a)$ must satisfy o

This together with Theorem 2.3.9 and the fact that $d(a)\Uparrow$ implies $d(a) \sqsubseteq_2^+ d'(a)$ suffices

to prove the claim.

i=1) The claim follows from the previous two cases. □

We are now ready to extend Lemma 2.5.20 to $\lessdot_i$.

Lemma 2.5.22 For $d, d' \in NF_i$, $d \sqsubseteq_i^+ d'$ implies $d \lessdot_i d'$

Proof: The proof follows by structural induction from the previous two lemmas. □

We now perform another step toward proving completeness by showing that once we have established a relation, $\lessdot_i$, between normal forms we can just use axioms or rules from our proof system to determine whether two normal forms are related.

Lemma 2.5.23 For $d, d' \in NF_i$, $d \lessdot_i d'$ implies $d \sqsubseteq_i d'$

Proof:

i=3) By induction $d(a) \sqsubseteq_3 d'(a)$ for every $a \in Init(d)$. Using the axiom $\Omega \sqsubseteq X$ it is easy to derive $d \sqsubseteq_3 d'$.

i=2) If $d \not\equiv d'$ then $d = \Omega$ and the result follows from $\Omega 1$. We are left to consider the cases d and d' satisfy conditions i.-iii. of Definition 2.5.18. We assume that $d(a) \sqsubseteq_2 d'(a)$, for every $a \in Init(d')$ and let r denote $d[d'(a)/d(a),\ a \in Init(d')]$ (note that $Init(d') \subseteq Init(d)$). To prove the claim it is sufficient to show that whichever condition of Definition 2.5.8 is satisfied we may derive $r \sqsubseteq_2 d'$.

i. r is d'

ii. $r =_2 d' + \Sigma\{\tau\Sigma\{ad(a) \mid a \in L\} \mid L \in \mathbf{L}(d) - \mathbf{L}(d')\}$

$\sqsubseteq_2 d'$ using E3

iii. $r =_2 d' + \Sigma\{\tau\Sigma\{ad(a) \mid a \in L\} \mid L \in \mathbf{L}(d) - \mathbf{L}(d')\} + \tau.d'$

$\sqsubseteq_2 d'$ using E3 and N4

i=1) This proof is very similar to the previous one. Let r be defined as above. Since $Init(d) = Init(d')$, we have $d \sqsubseteq_1 r$. We again have to take into account that d and d' have to satisfy i. or ii. or iii. of Definition 2.5.8.

i. $r \sqsubseteq_1 d'$ using $\Omega 1$

ii. $r \sqsubseteq_1 d' + \Sigma\{\tau\Sigma\{ad'(a) \mid a \in L\} \mid L \in \mathbf{L}(d) - \mathbf{L}(d')\}$

$\sqsubseteq_1 d' + \Sigma\{ad'(a) \mid a \in A(\mathbf{L}(d) - \mathbf{L}(d'))\}$ using N4

$= d'$ using D8

The last derivable axioms is applicable since every a in k must appear in a summand of d'.

iii. Similar to the corresponding one of case i=2. Instead of E3, D8 is used, similarly to ii. above. □

Proposition 2.5.24 (Completeness)

If e and e' are two finite CCS terms then

a. $e \sqsubseteq_i^+ e'$ implies $A_i \vdash e \sqsubset e'$

b. $e \simeq_i^+ e'$ implies $A_i \vdash e = e'$

Proof This follows from the existence of normal forms for e and e' (Propositions 2.5.14, 2.5.15, 2.5.16) and from the previous two lemmas. □

2.6 Extending the proofs to infinite terms

In the last section we have proved that all the axioms of Table 2.1 are consistent with respect to testing equivalences and are complete for the finitary version of CCS. There are relatively standard methods for extending completeness results to the full calculus. These methods depend on showing that the behaviour of infinite processes is completely determined by the behaviour of their finite approximants.

In Section 2.4 we have defined the set of finite terms which approximate CCS terms. A recursively defined term is approximated by its syntactically finite unwindings. In this

section, we establish a relationship between the semantics of the finite unwindings of terms and the semantics of the infinite terms. This will suffice to prove that the properties which are necessary for the extension hold.

Similarly to the finitary case, the proofs will be based on the existence of some kind of normal forms. Due to the fact that we also have to consider infinite terms, we cannot prove that any CCS term can be transformed to any of the normal forms defined in the previous section. We will use head normal form (hnf), instead; i.e. we will use particular terms which resemble the previous normal forms up to a certain depth. Since we only need normal forms when comparing infinite terms with finite ones, this will be sufficient.

Definition 2.6.1 Head normal form (hnf)

p is in **head normal form** if it is of the form

a. $\Sigma\{ap(a) \mid a \in L\}$ or

b. $\Sigma\{\tau\Sigma\{ap(a) \mid a \in L\} \mid L \in \mathbf{L}\}$ where $\mathbf{L}$ is saturated. □

We will use τ-hnf and a-hnf in the obvious way. Note that if p is in head normal form then $p\Downarrow$. Therefore, for example, Ω is not in hnf. We can prove that for any *convergent* CCS term t a head normal form exists which can be proved (by $\mathbf{A_1}$, R1, R2) to be equivalent to it. Head normal forms must be treated in much the same way as normal forms and use the same lemmas. Rather than giving the entire proofs of the lemmas, we merely state the required results.

Lemma 4.6.2 If p_i are in head normal form, $1 \le i \le k$, then two head normal forms h_1, h_2 exist such that

i. $\Sigma\{\tau p_i \mid 1 \le i \le k\} =_1 h_1$

ii. $\Sigma\{p_i \mid 1 \le i \le k\} =_1 h_2$.

Proof: This is similar to Lemmas 2.5.10 and 2.5.11 and Corollaries 2.5.12 and 2.5.13. □

Proposition 2.6.3 If $p\Downarrow$ then a head normal form hnf(p) exists such that p = hnf(p).

Proof: We use induction on the size of $\{p' \mid p=\tau\Rightarrow p'\}$. Thus, we may assume the result for every p' such that $p-\tau\rightarrow p'$. If $p\Downarrow$ then $p\downarrow$ and we will now use induction on the proof of this.

i) ap'. By definition this is in hnf.

ii) $\tau p'$. Then $p'\downarrow$. By induction p' has a hnf, hnf(p'). If hnf(p') is in a-hnf then τhnf(p') is in τ-hnf. Otherwise

p = τhnf(p')

= hnf(p') using D2

iii) p_1+p_2. Then $p_1\downarrow$, $p_2\downarrow$ and thus, by induction, both hnf(p_1) and hnf(p_2) exist. Therefore

p_1+p_2 = hnf(p_1) + hnf(p_2)

= h, for some hnf h, using the previous lemma.

iv) p'[S]. Then $p'\downarrow$ and thus, by induction, hnf(p') exists. We may now use S1-S3 to transform hnf(p')[S] into hnf.

v) $p\backslash l$. Similar to iv) use H1-H3 instead of S1-S3.

vi) rec x. t. Then t[rec x. t/x]$\downarrow$ and by induction there exists a hnf h such that h = t[rec x. t/x]. The result now follows since rec x.t = t[rec x. t/x], is an axiom in $\mathbf{A_1}$.

vii) $p_1|p_2$. Then $p_1\downarrow$, $p_2\downarrow$ and, by induction, hnf(p_1), hnf(p_2) exist. There are three cases, depending on their form.

a) hnf(p_1) is $\Sigma\{ap(a) \mid a \in L\}$, hnf($p_2$) is $\Sigma\{bq(b) \mid b \in K\}$

Then applying C1 we get

$$p_1|p_2 = \Sigma\{a(p(a) \mid hnf(p_2)) \mid a \in L\} + \Sigma\{b(hnf(p_1) \mid q(b)) \mid b \in K\} + \Sigma\{\tau\, a(p(a) \mid q(b)) \mid a \in L, b \in K \text{ and } a = \bar{b}\}$$

Since $p_1|p_2 -\tau\rightarrow p(a)|q(b)$ whenever $a = \bar{b}$ we may assume, by induction, that to each of these terms corresponds a hnf. The result now follows from Lemma 2.6.2

b) hnf(p_1) is in a-hnf, hnf(p_2) is in τ-hnf.

c) Both hnf(p_1) and hnf(p_2) are in τ-hnf.

These cases are similar to a). □

We may now concentrate on inferring properties of infinite terms from the corresponding properties of their finite approximations. All the results will be based on the fact that all preorders $\sqsubseteq_i^+$ enjoy some inductive properties.

Proposition 2.6.4 For every $p, q \in CREC_\Sigma$ we have $p < q$ implies $p \sqsubseteq_i^+ q$

Proof: From Proposition 2.5.4 we have that $\sqsubseteq_i^+$ satisfy $\Omega 1$ and REC1 and the three implications in the definition of $<$. Since $<$ is the least of such relations, it follows that $p < q$ implies $p \sqsubseteq_i^+ q$. □

Proposition 2.6.5 For every finite closed term d there exists a finite number of experiments $\mathbf{O}(d,p)$ such that $d \sqsubseteq_i^{\mathbf{O}(d,p)} p$ if and only if $d \sqsubseteq_i^+ p$.

Proof: The if part is trivial. We prove the only if part.

From Theorem 2.3.7, we know that a set of specific observers $\mathbf{O}_i$ is necessary to define $\sqsubseteq_i^+$. Let Act(d, p) be the set of actions used in the definitions of d and p. This set is finite. Now, let $A(d, p) = Act(d, p) \cup \overline{Act}(d, p) \cup \{a\}$ where $a, \bar{a} \notin Act(d, p)$, let $|d|$ denote the length of the longest sequence of actions d may perform and let $\mathbf{O}_i(d, p)$ be the set of observers in $\mathbf{O}_i$ which may only perform actions from A(d, p) and such that the length of the sequences of actions they may perform is smaller than or equal to $|d|+1$. Now, because of the way computations are defined, it is not difficult to see that for any observer o which is not in $\mathbf{O}_i(d, p)$ we cannot have d may satisfy o and p may $\not$satisfy o or d must satisfy o and p must satisfy o. In fact, if $|o| > |d|+1$ then d may $\not$satisfy o, moreover d must satisfy o only if for all $s \in traces(o)$ of length $|d|$ we have $\bar{s} \notin traces(d)$. But this would imply that $\bar{s} \notin traces(p)$ and thus that p must satisfy o. If o contains some actions which are not in A(d, p) then we may substitute all of them with a and nothing changes. This is sufficient to prove that observers

from $\mathbf{O}_i(d, p)$ suffice to determine whether $d \sqsubseteq_i^+ p$. Act(d, p) finite and Theorem 2.3.7 allow us to conclude that $\mathbf{O}_i(d, p)$ is finite. □

Lemma 2.6.6 $d \sqsubset_i p$ implies there exists $d' < p$ such that $d \sqsubset_i d'$.

Proof: By induction on the number of times the rule $t[rec\ x.t/x] \sqsubseteq_i rec\ x.t$ is used in the proof of $d \sqsubset_i p$. □

Lemma 2.6.7

a) $p\text{-}\mu\rightarrow q$ and $d' < q$ implies there exists d with $d < p$ such that $d\text{-}\mu\rightarrow d'$

b) $p\text{-}\mu\rightarrow q$ and $q < r'$ implies there exists r with $p < r$ such that $r\text{-}\mu\rightarrow r'$

c) $p\text{-}t\rightarrow q$, with $t \in A_\chi^*$, and $d' < q$ implies there exists d with $d < p$ such that $d\text{-}t\rightarrow d'$.

Proof:

a) and b) can be proved by induction on the proof that $p\text{-}\mu\rightarrow q$.

c) will be proved by induction on t. If $t = \varepsilon$ then the required d is Ω. If $t = t'\mu$ then there exists p', p" such that $p\text{-}t'\rightarrow p'\text{-}\mu\rightarrow q$. Since $\Omega < q$, from part a) we have there exists $e < p'$ such that $e\text{-}a\rightarrow\Omega$. The claim follows from the inductive hypothesis. □

Proposition 2.6.8

a) p may satisfy o implies d may satisfy o for some $d < p$

b) p must satisfy o implies d must satisfy o for some $d < p$

Proof:

a) Follows from the previous lemma. Since p may satisfy o only if there exists s such that $p=s\Rightarrow$, $o=\bar{s}\Rightarrow o'\text{-}w\rightarrow$.

b) Suppose p must satisfy o. Consider the computation tree from o|p where the leaves are labelled by terms o'|p' such that $o'\text{-}w\rightarrow$. Every term has a finite number of successors and by Konig Lemma this tree is finite. We use induction on the size of the tree. Furthermore because of lemma 2.6.6, it is sufficient to show that there exists a d such that $d \sqsubset_1 p$ and d must satisfy o. If $p\Uparrow$ then $o\text{-}w\rightarrow$ and the required d is Ω. Thus, we may assume $p\Downarrow$ and therefore

in hnf. Furthermore, we may assume $o\text{-}w\!\!\rightarrow$. There are two cases according to the form of p.

i) $p = \Sigma\{ap(a) \mid a \in L\}$. Let L' denote the set of $a \in L$ such that $o\text{-}\tau\!\!\rightarrow o_1 \ldots\ldots \text{-}\tau\!\!\rightarrow o_n\text{-}\bar{a}\!\!\rightarrow$, where $o_k\text{-}w\!\!\rightarrow$, for $1 \leq k \leq n$. If $L' = \emptyset$ then the required d is $\Sigma\{a\Omega \mid a \in L\}$. Thus, we may assume $L' \neq \emptyset$. For each $a \in L'$, let $D(a)$ denote the set of o' such that $o\text{-}\tau\!\!\rightarrow o_1\text{-}\tau\!\!\rightarrow \ldots\text{-}\tau\!\!\rightarrow o_n\text{-}\bar{a}\!\!\rightarrow o'$ where $o_k\text{-}w\!\!\rightarrow$, for $1 \leq k \leq n$. Then, for every $o' \in D(a)$, $p(a)$ must satisfy o'. By induction on the size of the computation tree there exists a finite term $d(a, o')$ such that $d(a, o')$ must satisfy o' and $d(a, o') \sqsubseteq_1 p(a)$. The required d is $\Sigma\{ad(a, o') \mid a \in L', o' \in D(a)\} + \Sigma\{a\Omega \mid a \notin L'\}$

ii) $p =_1 \Sigma\{\tau\Sigma\{ap(a) \mid a \in L\} \mid L \in \mathbf{L}\}$. Let p_L denote $\Sigma\{ap(a) \mid a \in L\}$. Then for every $L \in \mathbf{L}$ we have p_L must satisfy o. By induction on the size of the computation tree there exists d_L such that d_L must satisfy o, and $d_L \sqsubseteq_1 p_L$. The required d is $\Sigma\{\tau d_L \mid L \in \mathbf{L}\}$ □

Proposition 2.6.9 If $d \sqsubseteq_i^+ q$ for all $d < p$ then $p \sqsubseteq_i^+ q$.

Proof We consider the case i=2. Suppose $d \sqsubseteq_2^+ q$ for every $d < p$ and p must satisfy o then, from the previous proposition, we have d must satisfy o for some $d < p$ and thus also q must satisfy o. Because of theorem 2.3.6, to prove $p \sqsubseteq_2^+ q$ we are only left to prove that if $d\text{-}\tau\!\!\rightarrow$ for every $d < p$ then $p\text{-}\tau\!\!\rightarrow$. This follows from part b. of Lemma 2.6.7. The proof for $\sqsubseteq_3^+$ is similar and that for $\sqsubseteq_1^+$ follows from $\sqsubseteq_2^+$ and $\sqsubseteq_3^+$. □

Proposition 2.6.10 If $d \sqsubseteq_i^+ q$ then $\exists \bar{d} < q$ such that $d \sqsubseteq_i^+ \bar{d}$.

Proof From Proposition 2.6.5, we have that $d \sqsubseteq_i^+ q$ if and only if $d \sqsubseteq_i^{\mathbf{O}(d, q)+} q$. If i = 2 to prove the claim it is sufficient to show that there exist $\bar{d}$ such that d must satisfy o implies $\bar{d}$ must satisfy o for every $o \in \mathbf{O}_2(d, q)$ and $d\text{-}\tau\!\!\rightarrow$ implies $\bar{d}\text{-}\tau\!\!\rightarrow$. From Proposition 2.6.8 b) we have that for every $o_i \in \mathbf{O}_2(d, q)$, $i > 0$, there exists $d_i < q$ such that q must satisfy o implies d_i must satisfy o. Moreover, from Lemma 2.6.7 a), we have $q\text{-}\tau\!\!\rightarrow$ implies $\exists d_0 < q$ such that $d_0\text{-}\tau\!\!\rightarrow$. Since $(Fin(q), <)$ is directed, $\mathbf{O}(d, q)$ is finite and $<$ preserves $\sqsubseteq_i^+$ we have that there exist $d'' = \bigsqcup d_i$ such that $d'' < q$ and q must satisfy o implies d'' must satisfy o for every $o \in \mathbf{O}_2(d, q)$ and $q\text{-}\tau\!\!\rightarrow$ implies $d''\text{-}\tau\!\!\rightarrow$. Since, by hypothesis d must satisfy o implies q must satisfy o and $d\text{-}\tau\!\!\rightarrow$ implies $q\text{-}\tau\!\!\rightarrow$, the claim follows for $\bar{d} = d''$ when i = 2. We also can find d''' such that $d \sqsubseteq_3^+ d'''$ and, if we let $d' = \bigsqcup\{d'', d'''\}$, we also have $d \sqsubseteq_1^+ d'$. □

We can now prove soundness of R4 and extend the completeness results to infinitary CCS.

Proposition 2.6.12 The rules R3ii. and R4 of the proof system of Table 2.3 are sound.

Proof: The soundness of R4 follows from Proposition 2.6.9. The soundness of R3ii. from R4 (Lemma 2.4.3). □

Proposition 2.6.13 (Theorem 2.2.6)

$t \sqsubseteq_i^+ t'$ if and only if $t \sqsubseteq_i^c t'$

Proof: If $t \sqsubseteq_i^c t'$ then obviously $t \sqsubseteq_i^+ t'$. Conversely suppose $t \sqsubseteq_i^+ t'$. Then, for any context $\mathbf{C}[\ ]$, we can apply R3 i.-iii. to prove that $\mathbf{C}[t] \sqsubseteq_i^+ \mathbf{C}[t']$. This can be proven by structural induction on $\mathbf{C}[\ ]$. It follows that $\mathbf{C}[t] \sqsubseteq_i \mathbf{C}[t']$ and therefore $t \sqsubseteq_i^+ t'$. □

Proposition 2.6.14 (Theorem 2.4.4) - Soundness -

$\mathbf{A}_i \vdash t \sqsubseteq t'$ $(t = t')$ implies $t \sqsubseteq_i^c t'$ $(t \simeq_i^c t')$

Proof Proposition 2.6.12 justifies R3 ii. and R4. The remaining rules and axioms were proved sound in Proposition 2.5.4. □

Proposition 2.6.15 (Theorem 2.4.5) - Completeness -

If p and q are two closed terms then $p \sqsubseteq_i^c q$ implies $\mathbf{A}_i \vdash p \sqsubseteq q$

Proof We know $p \sqsubseteq_i^c q$. To apply R4 to derive $p \sqsubseteq q$ it is sufficient to show that $\mathbf{A}_i \vdash d \sqsubseteq q$ for every d in $Fin(p)$. Now $d < p$ and thus, by Proposition 2.6.6, $d \sqsubseteq_i^c p$ and also $d \sqsubseteq_i^c q$. Proposition 2.6.10 implies there exists $d' < q$ such that $d \sqsubseteq_i^c d'$. On the other side the completeness theorem for finite terms (Proposition 2.5.24) implies $\mathbf{A}_i \vdash d \sqsubseteq d'$. Since $d' < q$ implies $d' \sqsubseteq q$ we have $\mathbf{A}_i \vdash d \sqsubseteq q$. We can now apply R4 and the claim follows. □

3. FULLY ABSTRACT MODELS FOR CCS

3.0. Introduction

In the first two chapters we have seen how the behaviour of nondeterministic communicating systems can be described in terms of the sequences of actions they may perform. Moreover, we have shown how to use various notions of behavioural equivalences to abstract from unwanted details. We have called this approach to systems semantics **two level operational semantics**, as it first allows the behaviour of systems to be described in terms of abstract machines (labelled transition system) and then makes it possible to forget their internal structure by identifying all the machines which exhibit the same external behaviour. This approach to semantics has the advantage of being very close to our operational view of systems and of not enforcing any prior decision on the aspects to be ignored. At the same time, it offers the possibility of having different models which depend on the operationally-defined congruence relations over systems.

Another widely used approach to semantics is the denotational one. In this case, an element of a predefined semantic domain is associated to every system. This approach has the undisputed advantage of mathematical tractability and offers the possibility of easily proving systems properties. In fact, to prove that two systems are equivalent it is sufficient to show that they are associated to the same element of the semantic domain. In some cases, this approach may be found to be too abstract and may make it difficult to check whether the semantics we obtain respect our intuitions about systems behaviour. In order to appreciate the full extent of the impact of denotational semantics on systems behaviour it is necessary to compare it with other, more concrete, semantics. Excellent touchstones are operationally

based semantics or complete axiomatic characterizations of the induced equivalences.

In this chapter, we will outline some general techniques for denotational semantics and discuss criteria, which can be used to examine the relationships between denotational and operational semantics. In the next chapter, we will show how the study of complete axiomatic characterizations of the equivalence induced by a denotational semantics permits us to detect anomalies of the chosen denotations and of the semantic domains used. The paradigm which seems to emerge, at least when considering communicating systems, is that it is important to have as many different approaches to semantics as possible and to check one against the others. We are not (yet?) in a situation which allows us to choose a single approach and follow it in every circumstance.

When considering the relationships between denotational and operational semantics for a particular class of systems, a denotational semantics will be **sound with respect to** an operational one if the equivalence relation induced by the denotational semantics on the systems of the class under consideration is a subset of the equivalence relation induced by the operational semantics. If the converse holds, then a denotational semantics will be **complete with respect to** the operational one. Obviously it is not difficult to find a denotational semantics which is sound and another which is complete. In fact the identity function (which distinguishes all syntactically different systems) is sound with respect to every operational semantics. On the other hand, the constant function which maps every system into the one point domain (identifying all systems) is complete with respect to every operational semantics. It is by querying whether a denotational semantics is both sound and complete **(fully abstract)** with respect to an operational semantics that we can check whether the two semantics are in complete agreement.

We will propose three new models (semantic domains) which offer three denotational semantics for CCS. The three denotational semantics will be proved fully abstract with respect to the three operational semantics induced by the testing preorders discussed in the previous chapter. The semantic models will be constructed in a very abstract way (completion

by ideals of the partial orders generated by the complete set of axioms of the previous chapter) but, afterwards, we will be able to show that they can be associated with collections of special kinds of trees (**Representation Trees**) which are very similar to the Acceptance Trees of /DeN84/ and to the Asynchronous Nondeterministic Trees of /Hen84/.

Roughly speaking the representation tree associated with a particular process contains the following information:

1. The sequences of communications or actions which may be performed by the process.
2. A finite set of communications which represents the possible future of the process after every sequence of actions or communications. In general this will depend on internal nondeterministic choices and is characterized by the set of communications the process must accept at that stage of its progress.

The rest of the chapter is organized as follows. In the first section we state all definitions and basic results from domain theory and algebraic semantics which will be used throughout the chapter. In the second section we show how the equational characterizations of the testing preorders of chapter 2 can be used to build semantic domains for CCS which give fully abstract denotational semantics for the language. Finally, in the third section, we present the "concrete" models based on Representation Trees and show that they are isomorphic to the equationally generated models. This result is obtained by exploiting the close relationship between Representation Trees and the normal forms used to prove the completeness theorems of the previous chapter.

3.1. Algebraic Relations and Fully Abstract Models

In this section, we give some of the definitions which will be used in this and the next two chapters. We will assume familiarity with many notions which are at the basis of the denotational approach to semantics, as described in /GTWW77/ and /Gue81/. In fact, most of the definitions and propositions reported below are borrowed either from the above references or from /TWW79/.

Basic Definitions

- A set D, equipped with a relation $\sqsubseteq$, is a **partial order** (po) if $\sqsubseteq$ is a reflexive, antisymmetric and transitive relation. It will be denoted by $(D, \sqsubseteq)$.

- If $(D, \sqsubseteq)$ is a po, $X \subseteq D$ and $y \in D$, then y is an **upper bound** of X if for all $x \in X$ we have $x \sqsubseteq y$. Moreover, if for all $z \in D$, $(\forall x, x \sqsubseteq z)$ implies $y \sqsubseteq z$ then y is called the **least upper bound** (lub) of X and denoted by $\bigsqcup X$.

- If $(D, \sqsubseteq)$ is a po then a subset X of D is **directed** if for all $x, y \in X$ there exists $z \in X$ such that $x \sqsubseteq z$ and $y \sqsubseteq z$.

- If $(D, \sqsubseteq)$ is a po then a subset X of D is an **ideal** if it is directed and for all $d_1, d_2 \in D$ $d_1 \sqsubseteq d_2$ and $d_2 \in X$ implies $d_1 \in X$.

- A po, $(D, \sqsubseteq)$, is a **complete partial order** (cpo) if there exists $\perp \in D$ such that $\perp \sqsubseteq x$ for all $x \in D$, and if for all the directed subsets of D there exists a lub l such that $l \in D$.

- An element e of a cpo, $(D, \sqsubseteq)$, is **finite** (or **isolated**) if for all directed sets X in D we have that $e \sqsubseteq \bigsqcup X$ implies there exists $x \in X$ such that $e \sqsubseteq x$. We will write **FIN**(D) for the set of the isolated elements of the cpo D.

- A subset B of a cpo $(D, \sqsubseteq)$, is said to be a **base** of D if every $e \in B$ is isolated and for every $x \in D$ we have that there exists an E, which is a directed subset of B, such that $x = \bigsqcup E$.

- A cpo $(D, \sqsubseteq)$ is **algebraic** if it has a base.

- Given two cpo's $(D_1, \sqsubseteq_1)$ and $(D_2, \sqsubseteq_2)$ a function $f: D_1 \rightarrow D_2$ is **monotonic** if for all $x, y \in D$ we have $x \sqsubseteq_1 y$ implies $f(x) \sqsubseteq_2 f(y)$, and **continuous** if for all directed sets X of D_1 we have $f(\bigsqcup_1 X) = \bigsqcup_2 \{f(x) \mid x \in X\}$. Moreover a function which preserves the least elements ($f(t_1, \ldots, \bot, \ldots, t_n) = \bot$) is called **strict**.

These are most of the basic notions from domain theory that we will be using to give the denotational semantics of our languages. The approach we will follow is that suggested in /GTWW77/ and /Gue81/ and is based on continuous Σ-algebras. We assume familiarity with these algebras and with related notions. Let us now give some of their basic definitions.

- Let Σ be a fixed set of function symbols, (**signature**), each of which has a given arity. A **Σ-algebra** is a pair (A, Σ_A) in which A is a set (called the **carrier**) and Σ_A is a set of functions $\{f_A \mid f \in \Sigma\}$ such that if the arity of f is k then f_A is a function from $A^k \rightarrow A$.
We will often abbreviate (A, Σ_A) to A, if Σ_A is evident from the context.

- If Σ is a signature then T_Σ (the **term algebra** for Σ) is the least set of strings which contains all the function symbols of arity 0 and all the terms of the form $f(t_1, \ldots, t_k)$ where f is an element of Σ of arity k and $t_1, \ldots, t_k$ are strings in T_Σ.

- Any signature Σ can be extended by a set of variables X, to obtain the new signature $\Sigma(X)$, $T_\Sigma(X)$ is used to denote the term algebra for the new signature.

- If (A, Σ_A) and (B, Σ_B) are two Σ-algebras then a function $h: A \rightarrow B$ is a **Σ-homomorphism** if for every f in Σ_A we have $h(f_A(t_1, \ldots, t_k)) = f_B(h(t_1), \ldots, h(t_k))$.

- Two Σ-algebras (A, Σ_A) and (B, Σ_B) are **isomorphic** as Σ-algebras if and only if there exist two Σ-homomorphisms, $h: A \rightarrow B$ and $g: B \rightarrow A$, such that $h \circ g = id_B$ and $g \circ h = id_A$.

- If $(D, \sqsubseteq)$ is a po with a minimum element $\bot$, (D, Σ_D) is a Σ-algebra and every f_D in Σ_D is monotonic w.r.t. $\sqsubseteq$, then $(D, \Sigma_A, \sqsubseteq)$ is an **ordered Σ-Algebra** (Σ-po).

- A **homomorphism of ordered Σ-algebras** is a Σ-homomorphism which is monotonic and strict.

- If $(D, \sqsubseteq)$ is a cpo, (D, Σ_D) is a Σ-algebra and every f_D in Σ_D is continuous w.r.t. $\sqsubseteq$, then $(D, \Sigma_D, \sqsubseteq)$ is a **continuous Σ-Algebra** (Σ-cpo)

- A Σ-cpo, $(D, \Sigma_D, \sqsubseteq)$, is **algebraic** if $(D, \sqsubseteq)$ is an algebraic cpo.

- A **homomorphism of continuous Σ-algebras** is a continuous Σ-homomorphism.

- A Σ-algebra (Σ-po, Σ-cpo) I is **initial** in the class **C** of Σ-algebra (of Σ-po, of Σ-cpo) if for every $A \in \mathbf{C}$ there exists a unique (monotonic, continuous) Σ-homomorphism $h: I \rightarrow A$.

- **CT_Σ** is the Σ-po with carrier $T_{\Sigma \cup \{\bot\}}$ ordered by the smallest partial order relation satisfying the following conditions:
  a. $\bot \sqsubset t$ for every $t \in T_\Sigma$
  b. if $f \in \Sigma$, the arity of f is k and $t_i \sqsubset t'_i$ for all $1 \leq i \leq k$ then $f(t_1, \ldots, t_k) \sqsubset f(t'_1, \ldots, t'_k)$

- If $(A, \Sigma_A, \sqsubseteq)$ is a Σ-po, a **Σ-preorder over A** is a relation, $<$, over A which is reflexive, transitive, compatible with $\sqsubseteq$ (for every a, a' in A, $a \sqsubseteq a'$ implies $a < a'$) and function preserving (for every a, a' in A, $a < a'$ implies $f_A(a) < f_A(a')$ for every $f_A \in \Sigma$).

Given any Σ-po, A, and any Σ-preorder over A, $<$, we can factorize A via $<$ to obtain a new Σ-po, $(A/<, <, \Sigma_{A/<})$, where $A/<$ denotes the set of equivalence class over A generated by $<$, and, if $[a]$ is used to denote the representative of the equivalence class which includes a, we have that $[a] < [a']$ if $a < a'$ and $f_{A/<}([a_1], \ldots, [a_k]) = [f_A(a_1, \ldots, a_k)]$.

Theorem 3.1.1 $(A/<, <, \Sigma_{A/<})$ is a Σ-po and the injection mapping $A \rightarrow A/<$ is a Σ-po homomorphism. □

Theorem 3.1.2 The Σ-po $T_\Sigma/<$ is initial in the class $\mathbf{C}[<]$ □

We will be interested in instances of this theorem in which the Σ-preorder, $<$, is generated syntactically by sets of inequations (E). Such a preorder is denoted by $<_E$.

Given a set of inequations, a set of deduction rules can be defined which allows the inequations in E to be used to derive statements of the form $t < t'$ where t, t' are terms in $T_\Sigma(X)$. The set of deduction rules is similar to that already used in the previous chapter to prove that certain sets of inequations completely characterize the testing preorders defined on CCS. It is called **DED(E)** in /Hen85/ and consists of the following rules:

1. (**reflexivity**)

   $t < t$ is always true

2. (**transitivity**)

   $t < t'$ and $t' < t''$ implies $t < t''$

3. (**substitutivity**)

   $t_1 < t'_1, \ldots, t_k < t'_k$ implies $f(t_1, \ldots, t_k) < f(t'_1, \ldots, t'_k)$
   for every f in Σ of arity k

4. (**instantiation**)

   $t < t'$ implies $t\rho < t'\rho$ for every substitution ρ

5. (**inequations**)

   $t < t'$ is true for every inequation $t < t'$ in E.

We are now ready to show how a Σ-cpo can be used to give a meaning to all the terms of a language defined via a BNF.

Given the set of recursive terms over Σ (a set of operators), we can define REC_Σ, ranged over by t, by the following schema:

$t ::= x \mid op(t_1, \ldots, t_k), op \in \Sigma^k \mid rec\ x.\ t$ with $\Sigma = U\{\Sigma^k \mid k \geq 0\}$.

This definition was already given in Section 2.1, it is reported here for the sake of clarity. The same conventions of 2.1 connected with this syntax apply. We let $<$ be the least precongruence over REC_Σ generated by the axioms:

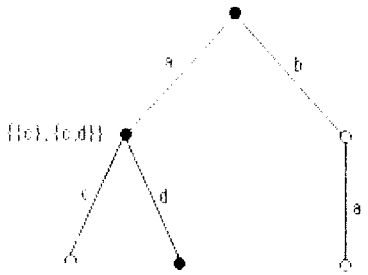
$\Omega 1$. $\Omega < X$

REC1. $t[rec\ x.\ t/x] < rec\ x.\ t$.

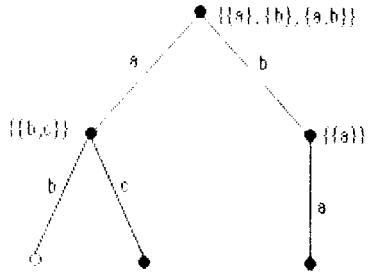
As in Section 2.4, $FREC_\Sigma$ denotes the set of finite terms, i.e. the set of terms which do not contain any subterm of the form rec x. t, and we let $Fin(p) = \{d \mid d < p, d \in FREC_\Sigma\}$.

Courcelle and Nivat /Niv75/, /CN76/ and the ADJ group /GTWW77/ have proposed to use Σ-cpo's to interpret a language defined as above. This idea arises from the general techniques of denotational semantics (/Sco76/). In general, the meaning of the various syntactic constructs is defined via semantic functions, in terms of the meanings of their components. Now, the semantic functions can be seen as homomorphisms between algebras. In fact, any context-free definition of the syntax of a programming language yields a complete Σ-algebra $(CT_\Sigma(X))$; given any other Σ-cpo, initiality of $CT_\Sigma(X)$ guarantees the existence of the semantic homomorphism. We will take an only slightly different approach, in which given any Σ-cpo as semantic domain, an element of the semantic domain is explicitly associated with all t in REC_Σ, via a function $\mathcal{M}$.

We need some additional notations. Given any Σ-cpo $(I, <_I, \Sigma)$, whose least element is denoted by $\perp_I$, let ENV_I denote the set of I-environments which consists of a set of mappings from X (the set of variables) to I. An element ENV_I will be denoted by e_I. If $g \in I$ then $e_I[g/x]$ is the environment which coincides with e_I except that at x, where it maps x to g. (Whenever I is

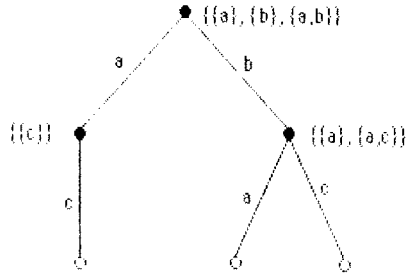


1) representation of $p_1 = a.(\tau.c.\Omega + \tau.(c.\Omega + d)) + b.(a.\Omega + \Omega)$



2) representation of $p_1 = \tau.a.\tau.(b\Omega + c) + \tau.b.\tau.a + \tau.(a.\tau.(b.\Omega + c) + b.\tau.a)$

and $p'_1 = \tau.a.(b.\Omega + c) + \tau.b.a$

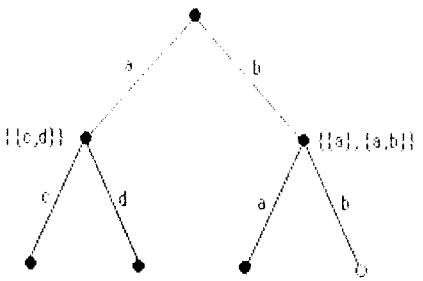


3) representation of $p_3 = \tau.a.r + \tau.b.q + \tau.(a.r + b.q)$

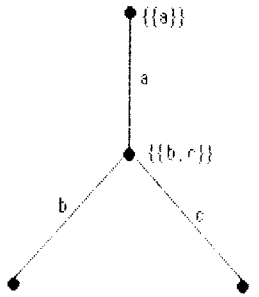
and $p'_3 = \tau.a.c.\Omega + \tau.b.q$

where $r = \tau.c.\Omega$ and $q = \tau.a.\Omega + \tau.(a.\Omega + c.\Omega)$

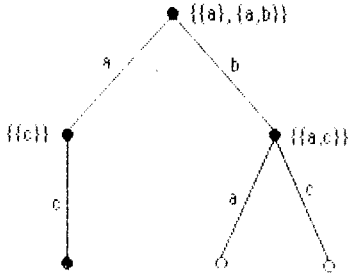
Table 3.2



1) representation of $q_1 = a.\tau.(c + d) + b.(\tau.a + \tau.(a + b\Omega))$



2) representation of $q_2 = a.\tau.(b + c)$



3) representation of $q_3 = \tau.a.\tau.c + \tau.(a.\tau.c + b.\tau.(a.\Omega + c.\Omega))$

and $q'_3 = \tau.a.c + \tau.(a.c + b.(a.\Omega + c.\Omega))$

Table 3.3

Proposition 3.3.5 For i = 1, 2, 3

a. $(RT_i, <_i)$ is a complete partial order;

b. $(RT_i, <_i)$ is an algebraic complete partial order whose finite elements are all the trees from RT_i with a finite number of nodes.

Proof:

- We first prove **part a.** for every i in {1, 2, 3}.

i = 3. $<_3$ is certainly a partial order since $\subseteq$ is a partial order. The least element is "o" (the trivial tree consisting only of an open node). To define the lub of any directed subset of RT_3 it is sufficient to define a prefix-closed set of strings. Given any directed $D \subseteq RT_3$ we may define an upper bound t by:

Nodes(t)= U {Nodes(d) | d ∈ D}.

It is trivial to check that this is also the lub of D. In fact, because of the way Nodes(t) has been defined, if we have that there exists t' ∈ RT_3 such that Nodes(t') ⊂ Nodes(t) we must also have that there exists d ∈ D and s ∈ Nodes(d) such that s ∉ Nodes(t').

i = 2. The least element of RT_2 is again the trivial tree "o". The relation $<_2$ is easily checked to be reflexive, transitive and antisymmetric, being defined in terms of set inclusions and equalities. It remains to prove that any directed set D of trees from RT_2 has a least upper bound belonging to RT_2. Given any directed $D \subseteq RT_2$, we can define a new representation tree t by defining Nodes(t), Cnodes(t) and $\mathcal{A}(t, s)$ as follows:

1. Nodes(t) = {s | s ∈ Nodes(d) for almost all d in D};
2. CNodes(t) = {s | s ∈ Nodes(t) and s ∈ CNodes(d) for some d in D};
3. s ∈ CNodes(t) implies $\mathcal{A}(t, s) = \cap\{\mathcal{A}(d, s) \mid d \in D \text{ and } s \in CNodes(d)\}$.

Now we have that this definition of t implies:

(*) $\mathcal{A}(t, s) = \mathcal{A}(d, s)$ for some d ∈ D and

(**) Succ(t, s) = Succ(d, s) for some d ∈ D and, since D is directed,

(***) $\forall s \in Nodes(t)\ \exists d_s$ such that $\mathcal{A}(t, s) = \mathcal{A}(d_s, s)$ and $Succ(t, s) = Succ(d_s, s)$.

This is sufficient to conclude that t ∈ RT_2. It remains to prove that t is also the lub of D. First we prove that t is an upper bound (d $<_2$ t ∀d ∈ D), i.e. that d and t satisfy part ii. of definition 3.3.4. Indeed, given any d ∈ D, d(s) closed implies either s ∉ Nodes(t) or s ∈ CNodes(t). This is sufficient to satisfy Condition 1 of 3.3.4 ii. Moreover, because of the way $\mathcal{A}(t, s)$ is defined we will certainly have $\mathcal{A}(t, s) \subseteq \mathcal{A}(d, s)$ for every d ∈ D. With regards to acceptance sets the case $\mathcal{A}(t, s) = \emptyset$ remains to be considered. In this case, because of (*) -(***), we have there exists d' ∈ D such that $\mathcal{A}(d, s) = \emptyset$ and $\mathcal{A}(t, s) = \mathcal{A}(d', s)$. Then since D is directed, we have that for every d ∈ D either $\mathcal{A}(d, s) = \emptyset$ and Succ(t, s) = Succ(d, s) or $\mathcal{A}(d, s) \neq \emptyset$ and Succ(d', s) ∈ $\mathcal{A}(d, s)$. It can thus be concluded that t is an upper bound. Because of (*)-(***), it is also not difficult to prove that t is the least upper bound. In fact, it is trivial to show that for any t' ≠ t such that t' $<_2$ t we can find d ∈ D such that we have $\not(d <_2 t')$.

i = 1. The least element is "o", and $<_1$ is a partial order since both $<_2$ and $<_3$ are partial orders and the union of two partial orders gives a partial order. Given any directed subset D of RT_1, we can define its lub as follows:

1. Nodes(t) = U {Nodes(d) | d ∈ D};
2. CNodes(t) = U { CNodes(d) | d ∈ D};
3. s ∈ CNodes(t) implies $\mathcal{A}(t, s) = \cap\{\mathcal{A}(d, s) \mid d \in D \text{ and } s \in CNodes(d)\}$.

Simple calculations suffice to prove that t is an element of RT_1 and that this is indeed the lub of D. In fact, from the remark following Definition 3.3.4 and from reasonings similar to those in case i = 2, it is not difficult to conclude:

i. Succ(t, s) = U {Succ(d, s) | d ∈ D and s ∈ Nodes(d)};
ii. if t(s) is closed then for all but a finite number of d in D we have $\mathcal{A}(t, s) = \mathcal{A}(d, s)$ and Succ(t, s) = Succ(d, s).

- We can now prove **part b**.

We will only prove that $(RT_2, \lhd_2)$ is algebraic since cases i = 1 and i = 3 can be treated in much the same way. We must prove that:

1) Given any $D \subseteq RT_2$, for every $d \in FRT_2$, $d \lhd_2 \bigsqcup D$ implies $\exists \bar{d} \in D$ such that $d \lhd_2 \bar{d}$ and

2) For every $t \in RT_2$ there exists $D \subseteq FRT_2$, such that $t = \bigsqcup D$.

1) Suppose $d \lhd_2 \bigsqcup D$. Then (*) - (***) imply that for every $s \in Nodes(d)$ there exists $d'_s \in D$ such that $s \in Nodes(d'_s)$ and for every $s \in CNodes(d)$ there exists $d''_s \in D$ such that $\mathcal{A}(d''_s, s) \subseteq \mathcal{A}(d, s)$. Since the sets $\{s \mid s \in Nodes(d)\}$ and $\{s \mid s \in CNodes(d)\}$ are finite by hypothesis and $\mathcal{A}(d, s)$ is finite by definition the claim follows with $\bar{d} = \bigsqcup\{d'_s, d''_s \mid s \in Nodes(d)\}$.

2) Given any t, the set of finite elements which approximates it can be defined as follows:

$t = \bigsqcup\{t_n \mid n \geq 0\}$ where:

$Nodes(t_n) = \{s \mid s \in Nodes(t) \text{ and } length(s) \leq n\}$

$CNodes(t_n) = \{s \mid s \in CNodes(t) \text{ and } length(s) \leq n\}$

$s \in CNodes(t_n)$ implies $\mathcal{A}(t_n, s) = \mathcal{A}(t, s)$. □

In order to use Rt_i's as a model for CCS we must also define operations on them in correspondence of every CCS operator. In the following, these operations are defined indirectly, by exploiting the strong correspondence between RT_i and the normal forms of Section 2.5.2. The same correspondence will be used to prove that the models of CCS based on RT_i are fully abstract with respect to the precongruences $\sqsubseteq_i^c$. Since $(RT_i, \lhd_i)$ are all algebraic cpo's, it is sufficient to define the operations on their bases, FRT_i.

Firstly, we concentrate on relating Representation Trees and CCS normal forms. We will let NF_1, NF_2 and NF_3 denote the set of normal forms, strong normal forms and weak normal forms, respectively.

Definition 3.3.6 Let Φ be a function from NF_i to FRT_i by structural induction as follows:

1. if $n = \Sigma\{an(a) \mid a \in L\}[+\Omega]$

then $\Phi(n)$ will be the following representation tree t

i. $Nodes(t) = \{\varepsilon\} \cup \{as \mid a \in L, s \in Nodes(\Phi(n(a)))\}$

ii. $CNodes(t) = \emptyset$ if Ω is a summand

$\{\varepsilon\} \cup \{as \mid a \in L, s \in CNodes(\Phi(n(a)))\}$ otherwise

iii. $\mathcal{A}(t, \varepsilon) = \emptyset$

$\mathcal{A}(t, as) = \mathcal{A}(\Phi(n(a)), s)$ if $as \in CNodes(t)$

2. if $n = \Sigma\{\tau.\Sigma\{an(a) \mid a \in L\} \mid L \in \mathbf{L}\}$

then $\Phi(n)$ is the following representation tree t

i. $Nodes(t) = Nodes(\Phi(\Sigma\{an(a) \mid a \in A(\mathbf{L})\})$

ii. $CNodes(t) = CNodes(\Phi(\Sigma\{an(a) \mid a \in A(\mathbf{L})\}))$

iii. $\mathcal{A}(t, \varepsilon) = \mathbf{L}$

$\mathcal{A}(t, as) = \mathcal{A}(\Phi(n(a)), s)$ if $as \in CNodes(t)$ □

Lemma 3.3.7

a. $n \in NF_i$ implies $\Phi(n) \in FRT_i$

b. For all $n, m \in NF_i$ $n \ll_i m$ implies $\Phi(n) \lhd_i \Phi(m)$

Proof:

a. The proof follows trivially by structural induction.

b. The proof is again by structural induction. Let us assume $\Phi(n(a)) \lhd_i \Phi(m(a))$, whenever n(a) and m(a) are defined. Because of the way $\ll_i$ is defined we only need to prove that $n \lhd_i m$ implies:

i = 3) $Succ(\Phi(n), \varepsilon) \subseteq Succ(\Phi(m), \varepsilon)$

i = 2) $\varepsilon \in Cnodes(\Phi(n))$ implies $\varepsilon \in Cnodes(\Phi(m))$;

$\mathcal{A}(\Phi(n), \varepsilon) \supseteq \mathcal{A}(\Phi(m), \varepsilon)$;

$\mathcal{A}(\Phi(m), \varepsilon) - \emptyset$ implies either $\mathcal{A}(\Phi(n), \varepsilon) = \emptyset$ and $Succ(\Phi(n), \varepsilon) = Succ(\Phi(m), \varepsilon)$

or $Succ(\Phi(m), \varepsilon) \in \mathcal{A}(\Phi(n), \varepsilon)$.

This is easily derived from the definition of $\lessdot_i$ of Section 2.5. □

Definition 3.3.8

Let Ψ be a function from FRT_1 to NF_1 defined by structural induction as follows

1. If t is an acceptance tree and $\mathcal{A}(t, \varepsilon) = \emptyset$ then

$$\Psi(t) = \Sigma\{a\Psi(t(\varepsilon, a)) \mid a \in Succ(t, \varepsilon)\} \quad [\ + \Omega]$$

where Ω is a summand only if $\varepsilon \in ONodes(t)$.

2. If t is an acceptance tree and $\mathcal{A}(t, \varepsilon) \neq \emptyset$ then

$$\Psi(t) = \Sigma\{\tau\, \Sigma\{a\Psi(t(\varepsilon, a)) \mid a \in L\} \mid L \in \mathcal{A}(t, \varepsilon)\}.$$ □

Lemma 3.3.9

a. $t \in FRT_i$ implies $\Psi(n) \in NF_i$

b. For all $t, t' \in FRT_i$ $t \lessdot_i t'$ implies $\Psi(t) \ll_i \Psi(t')$

Proof: Both a. and b. can be proved by structural induction using the same techniques as in the proof of Lemma 3.3.7. □

We can now define operators for both normal forms and Representation Trees.

Definition 3.3.10 For every $op \in \Sigma$ let

1. $op_i: NF_i^k \to NF_i$, $i = 1, 2, 3$ be defined by

   a) $op_1(n_1, \ldots n_k) = nf(\ op(n_1, \ldots n_k))$;

   b) $op_2(n_1, \ldots n_k) = snf(\ op(n_1, \ldots n_k))$;

   c) $op_3(n_1, \ldots n_k) = wnf(\ op(n_1, \ldots n_k))$.

2. $op'_i: FRT_i^K \to FRT_i$, $i = 1, 2, 3$ be defined by

$$op'_i(t_1, \ldots t_k) = \Phi(op_i(\Psi(t_1), \ldots \Psi(t_k)).$$ □

Proposition 3.3.11 op_i and op'_i of Definition 3.3.10 are monotonic.

Proof:

a. We need to show that

$$n_j \ll_i m_j \text{ for every } 1 \leq j \leq k \text{ implies } op_i(n_1, \ldots, n_k) \ll_i op_i(m_1, \ldots, m_k).$$

From Lemma 2.5.23 we have $n_j \ll_i m_j$ implies $n_j \sqsubseteq_i m_j$ and this implies $op_i(n_1, \ldots, n_k) \sqsubseteq_i op_i(m_1, \ldots, m_k)$ by R3 iii. of Table 2.3. Proposition 2.5.4 a) makes it possible to infer $op_i(n_1, \ldots, n_k) \sqsubseteq_i^c op_i(m_1, \ldots, m_k)$ and the claim follows from Lemma 2.5.22.

b. From part a) and from Lemma 3.3.9, we have that all the functions used to define op'_i are monotonic and the claim follows since the composition of two monotonic functions is a monotonic function. □

Proposition 3.3.12 $(NF_i, \ll_i)$ is a partial order and when equipped with the operators op_i is isomorphic as a Σ-po to $(FRT_i, \lessdot_i)$ with the operators op'_i.

Proof: $(NF_i, \ll_i)$ can be easily proved to be a partial order. It remains to show that there exists an isomorphism pair between the two Σ-po's, NF_i and FRT_i. This pair will be given by Φ and Ψ. In fact we can prove, by using structural induction, that the two functions are inverses. We now prove that both Φ and Ψ preserve the operators:

a. $\Phi(op_i(n_1, \ldots, n_k)) = op'_i(\Phi(n_1), \ldots, \Phi(n_k))$ and

b. $\Psi(op'_i(t_1, \ldots, t_k)) = op_i(\Psi(t_1), \ldots, \Psi(t_k))$.

From the definitions of op'_i we now have that

$op'_i(\Phi(n_1), \ldots, \Phi(n_k)) = \Phi(op_i(\Psi(\Phi(n_1)), \ldots, \Psi(\Phi(n_k)))$ and a. follows from the fact that Φ and Ψ are inverses. On the other side we also have $\Psi(op'_i(t_1, \ldots t_k)) = \Psi(\Phi(op_i(\Psi(t_1), \ldots \Psi(t_k)))$ and b. follows from the same reason. This together with Proposition 3.3.11 is sufficient to prove that Φ and Ψ are two Σ-homomorphisms. The claimed isomorphism follows from the fact that Φ and Ψ are inverses. □

Let us now turn our attention to the relationships between NF_i and the Σ-partial order induced on $FREC_\Sigma$ by the axioms $\mathbf{A}_i$ of Table 2.1.

Proposition 3.3.13 $(NF_i, \ll_i)$ equipped with the operators op_i is isomorphic as Σ-po to $(FREC_\Sigma/\sqsubseteq_i, \sqsubseteq_i)$ with the operators op $\in \Sigma$.

Proof: Let $id: NF_i \to FREC_\Sigma$ be the identity function and $\varepsilon_i: FREC_\Sigma \to NF_i$ be defined as $\varepsilon_1 = nf$, $\varepsilon_2 = snf$ and $\varepsilon_3 = wnf$. Obviously, id preserves both the ordering and the operators. Moreover, the results of Section 2.5 about the existence of normal forms, (Lemma 2.5.22 and Proposition 2.5.24) imply that ε_i is well defined and monotonic. We also have

$\varepsilon_i(op(d_1, \ldots, d_k) =_i op_i(\varepsilon_i(d_1), \ldots, \varepsilon_i(d_k))$,

since $\varepsilon_i(d_j) =_i d_j$ and $\sqsubseteq_i$ is preserved by every operator op in Σ. These two results suffice to conclude that also ε_i preserves the ordering and the operators. Finally, since, for every normal form n and for every CCS term d, we also have $\varepsilon_i \circ id(n) = n$ and $id \circ \varepsilon_i(d) =_i d$, the claim follows. □

We are now ready to state and prove the final results of this chapter. We will let $\mathbf{I}_i$ denote the ideal completion of the Σ-po generated by the axioms $\mathbf{A}_i$.

Theorem 3.3.14 For i = 1, 2, 3,

RT_i is a Σ-cpo which is isomorphic to $\mathbf{I}_i$ and therefore **fully abstract w.r.t.** $\approx_i^c$.

Proof: We have already shown that RT_i is an algebraic cpo with FRT_i as finite elements. To define the continuous functions $op_i: RT_i^k \to RT_i$ it is sufficient to define them for the finite elements. We have already done this and we can thus consider RT_i as a Σ-cpo. Moreover an algebraic Σ-cpo is uniquely determined, up to isomorphism, by the Σ-po induced on its finite elements. From Proposition 3.3.12 and Proposition 3.3.13 we may conclude that $\mathbf{I}_i$ is isomorphic to RT_i. Fully Abstractness follows from Theorem 3.2.2. □

The weakness of this representation theorem is that we have not given a natural definition of the CCS operators which apply directly to the trees in RT_i. We have defined them indirectly by defining them on normal forms and using the isomorphism between normal forms and the

finite trees in RT_i. However, the definitions of most of the operators may be found in /DeN84/. We have not reported them here because they turn out to be not very "natural". We consider the definitions via normal forms at least as intuitive as the direct definitions given in /DeN84/. The main reason for this is that, because of the heterogeneity between the way we treat the roots of the trees and the rest of their nodes, when defining every binary operation we must distinguish various cases, depending on whether the operands have empty or non-empty acceptance set associated with their root.

However, in the final chapter of this thesis, we will discuss a slight variation of RT_2 which makes it possible to overcome the above mentioned difficulties and to equip Representation Trees with "natural" operations. We will then be able to show that RT_2 is a subdomain of the new model and that also this can be used to give a denotational semantics to CCS.

4. MODELS AND AXIOMS FOR A THEORY OF CSP

4.0. Introduction

Since its appearance CSP /Hoe78/ has played the role of a test language for many proposed theories of concurrency by providing very interesting control structures. Various formal semantics for CSP have been proposed. The so-called **trace semantics**, introduced in /Hoe81/, associates with every process the set of its possible sequences of actions. It turns out to be suitable for reasoning about potential communication sequences but insensitive to deadlock. The β-semantics of /FHLR79/ attributes a meaning to each CSP process in two steps: firstly each process of a system is given its own semantics; the semantics of the system is then obtained by using a binding operator β which takes into account the concurrent interactions among the various processes. This method applies only to a subset of CSP. Other, more operational, lines have also been followed: /Plo83/ gives a structured operational semantics for CSP, /HLP81/ and /AZ82/ gives its semantics by translation into CCS.

Brookes, Hoare and Roscoe, /BHR84/, /Bro83b/ and /Ros82/, suggest a new approach to define a mathematical model for CSP. Their proposal is basically denotational but strongly related to operational behaviours. Following the lead of CCS, they introduce a notion of process based on the elementary concepts of event and transition. Starting from a set of elementary actions and a few basic processes they define a small set of combinators which allow larger processes to be built from smaller ones.

Their semantics is given by associating a so-called **refusal set** to every process. Each refusal set consists of a set of **failures**. A failure is a pair $\langle s, V \rangle$, where s is a finite sequence of visible actions in which the process may have been engaged up to a certain moment and V is the set of actions that the process can reject in the next step. The semantics of the various combinators is given by defining the transformations they induce on the domain of refusal sets. The association of a process with a refusal set is not one-to-one; the same refusal set can be associated to more than one syntactic process. A notion of equivalence is then introduced as follows: two processes are equivalent if and only if they have the same refusal set as denotation.

In the first chapter (Section 1.4), we discussed an operationally defined equivalence also based on possible traces and possible failures (*failures equivalence*). It was shown that this equivalence coincided with one of the three testing equivalences discussed in Section 1.6 for a large class of transition systems. It is tempting to try to understand whether Representation Trees, the domains obtained starting from testing equivalences, are related in some way to Refusal Sets, the domain from which failure equivalence was derived.

In this chapter, we begin investigations in this direction. The congruence on processes induced by the denotational model (the Refusal Sets model) is characterized algebraically and a sound and complete set of axioms for finite terms is defined to form the basis of the theory. This allows a deeper understanding of the model and throws lights on problems connected with the choices for the denotation of particular processes. In particular, the algebraic characterization sheds light on difficulties the chosen denotational semantics has in capturing the operational intuitions of the interaction between completely specified processes and the primitive process used to represent both divergence and under-specification. These difficulties lead to the definition of a new semantic domain obtained by imposing an additional constraint on the domain of refusal sets. The subdomain obtained in this way allows us to overcome the difficulties mentioned above and brings us closer to understanding the

interrelations between Refusal Sets and Representation Trees. In fact, their precise relationships will be studied in the next chapter.

The rest of this chapter is organized as follows. In Section 4.1 the syntax of a version, which does not contain recursively defined processes, of the language for a Theory of Communicating Sequential Processes (TCSP) is introduced; the refusal set model is then presented and used to give the semantics of the language following the same general framework discussed in the first section of the previous chapter; finally, the preorder naturally induced by the denotational semantics is discussed. In Section 4.2, it is proved that this preorder can be characterized as the smallest relation satisfying a particular set of axioms: A set of axioms is defined and proved consistent and complete with respect to the refusal set semantics of /BHR84/. The section ends with a discussion about certain inadequacies of the semantics which are evidenced by the axiomatic characterization, and with a discussion about the consequences of using refusal sets as the semantic domain for TCSP. In Section 4.3, a new semantic domain is presented and a more compact representation for refusal sets is proposed. A new semantics of the whole language for TCSP is defined in Section 4.4. A consistent and complete axiom system which characterizes the precongruence induced by the new semantics is presented in section 4.5 together with a new fully abstract model which naturally extends to infinite processes in the same way as in Section 2.6.

4.1 TCSP and Refusal Sets

In this section we review the definition of the various operators of a Theory for Communicating Sequential Processes (TCSP) and their denotational semantics. All the operators are described in /BHR84/, where their definition and use are motivated. At first we will consider only a subset of the operators; the reasons for the exclusion of others (rec x. _

and ||) will be discussed later. In the second part of the chapter we will consider all the operators.

Let us start by giving an informal description of all the operators of the theory:

1. **Inaction** Stop represents the process which never does anything

2. **Undefined** CHAOS represents the wholly arbitrary process which can exhibit every possible behaviour; this is the most nondeterministic process.

3. **Action** If A is a set of elementary actions and $a \in A$ and P is a process then $a \rightarrow P$ represents the process which can perform an a-action and then behave like P.

4. **External Choice** If P and Q are processes then $P \Box Q$ is a process which behaves like P or Q. The environment (the other processes interacting with it), can completely determine which subprocess would be in fact used.

5. **Internal Choice** If P and Q are processes then $P \sqcap Q$ is the process representing their nondeterministic composition. $P \sqcap Q$ behaves like either P or Q and the environment has no control over the choice.

6. **Parallel Composition** If P and Q are processes then $P \| Q$ is a process which can perform a particular action only if P and Q together can perform it. This gives a very tight form of parallelism.

7. **Interleaving** If P and Q are processes then $P \| Q$ is a process which can perform any sequence of actions obtained by arbitrary interleaving of the sequences of actions which can be performed by the two component processes.

8. **Hiding** If P is a process and a is an elementary action then P/a is a process which behaves exactly the same as P apart from the fact that all a-actions performed by P are invisible to the environment. These actions can occur without the other processes having any control over them.

The behaviour of processes built using the above operators is specified by associating a refusal set to every process. A refusal set provides a means to describe the traces a process can execute and at the same time the elementary actions it will be able to refuse to execute after each trace. A refusal set F is an element of $POW(A^* \times POW(A))$, which satisfies the following conditions:

1. $\langle s, V\rangle \in F$ **implies** V is finite.
2. $\langle \varepsilon, \{\}\rangle \in F$, where ε represents the empty string and $\{\}$ the empty set.
3. $\langle st, \{\}\rangle \in F$ **implies** $\langle s, \{\}\rangle \in F$.
4. $V \subseteq W$ and $\langle s, W\rangle \in F$ **implies** $\langle s, V\rangle \in F$.
5. Let $U = \{a \mid \langle sa, \{\}\rangle \in F\}$ and let W be a finite subset of $(A - U)$, then $\langle s, V\rangle \in F$ **implies** $\langle s, V \cup W\rangle \in F$.

Most of the conditions a refusal set is required to satisfy are intuitive. It must be non-empty (2.); its set of traces has to be prefix-closed (3.); its set of refusals must be downward closed (4.); and finally any refusal set must be such that if it does not contain any trace of the form "sa" then it must contain a refusal associated with s which contains "a", i.e. $\langle sa, \{\}\rangle$ not in F and $\langle s, V\rangle$ in F implies $\langle s, V \cup \{a\}\rangle \in F$, (5.). Condition 1. is less intuitive than the others: in section 4.4 we will show that this condition is not strictly necessary, it is been kept here for homogeneity with the results of /BHR84/. In the following we will use **F** to denote the subset of $POW(A^* \times POW(A))$ whose elements satisfy conditions 1. - 5.. Below we present syntax and semantics of **FTCSP (a finitary subset of TCSP)**. Following the same style of sections 2.1 and 3.1, the set of finite processes ranged over by P, Q, R ... is defined by a BNF. Below a denotes elements of a countable set A of elementary actions.

$$P := Stop \mid CHAOS \mid a \rightarrow P \mid P_1 \sqcap P_2 \mid P_1 \,\square\, P_2 \mid P_1 \,|\!|\, P_2 \mid P/a$$

We will use $\square \{P_i \mid i \in I\}$ to denote $P_{i(1)} \square \ldots \square P_{i(n)}$ where $I = \{i(1) \ldots i(n)\}$. If $I = \{\}$ then $\square \{P_i \mid i \in I\}$ will denote Stop. $\sqcap \{P_i \mid i \in I\}$ will be used to denote $P_{i(1)} \sqcap \ldots \sqcap P_{i(n)}$ where $I = \{i(1) \ldots i(n)\}$ is non-empty. The absence of brackets will be justified by axioms which will be introduced later; both $\square$ and $\sqcap$ will in fact be proved associative. The precedence of the operators is given by: $/a > a\rightarrow > |\!|\, > \sqcap > \square$.

This is for the syntax. To give the semantic function $\mathbf{F}$ for TCSP as in /BHR84/ we can use the general framework discussed in the previous chapter($\mathbf{F} = \mathcal{M}_{\mathbf{F}}$). To do so we need to prove that **F** is a cpo and to define a set of continuous operators over **F** which correspond to the operators in the syntax of FTCSP. Indeed, in this case, since we have only syntactically finite terms and no recursion operator, it would be sufficient to prove that **F** is a partial order and that its operators are monotonic . However, since completeness of **F** and continuity of its operators will be needed later, we state here the general results which are proved in /BHR84/.

We will now present the relevant operations on refusal sets. We will have an $op_{\mathbf{F}}$ for every operation op in the syntax of FTCSP. In the definition of $/_{\mathbf{F}}$ we use s/a to denote the string obtained from s when all the occurrences of a are removed.

$$Stop_{\mathbf{F}} = \{\langle \varepsilon, V\rangle \mid V \subseteq A, V \text{ finite}\}$$

$$CHAOS_{\mathbf{F}} = \{\langle s, V\rangle \mid s \in A^* \text{ and } V \subseteq A, V \text{ finite}\}$$

$$a \rightarrow_{\mathbf{F}} F = \{\langle \varepsilon, V\rangle \mid V \subseteq A - \{a\}, V \text{ finite}\} \cup \{\langle as, W\rangle \mid \langle s, W\rangle \in F\}$$

$$F_1 \,\square_{\mathbf{F}}\, F_2 = \{\langle \varepsilon, V\rangle \mid \langle \varepsilon, V\rangle \in F_1 \cap F_2\} \cup \{\langle s, W\rangle \mid s \in A^+ \text{ and } \langle s, W\rangle \in F_1 \cup F_2\}$$

$$F_1 \sqcap_{\mathbf{F}} F_2 = F_1 \cup F_2$$

$$F_1 \,|\!|_{\mathbf{F}}\, F_2 = \{\langle s, V \cup W\rangle \mid \langle s, V\rangle \in F_1 \text{ and } \langle s, W\rangle \in F_2\}$$

$$F/_{\mathbf{F}} a = \{\langle s/a, V\rangle \mid \langle s, V \cup \{a\}\rangle \in F\} \cup \{\langle (s/a)t, V\rangle \mid \langle t, V\rangle \in CHAOS \text{ and } \forall n\, \langle sa^n, \{\}\rangle \in F\})$$

Moreover, it should be noted that in the definition of $/_F a$ the second member of the main union operator is present only to cover the cases in which CHAOS is a subprocess of F and F can offer a sequence of a's and then behave like CHAOS.

It can be proved, see /Bro83/, that all the above defined operators are closed w.r.t. F. Moreover, from /Bro83/ and /BHR84/, we have that F ordered by reverse inclusion enjoys the properties which are needed if it is to be used as a model for FTCSP.

Proposition 4.1.1

Let $\Sigma_{FTCSP} = \{CHAOS, Stop, a\rightarrow, /a, \sqcap, \Box, \mathbf{I}\}$ then $(F, \supseteq)$ is a Σ_{FTCSP}-cpo. □

This proposition allows us to give a denotational semantics for FTCSP, following the approach discussed in Section 3.1 and taking as semantic function $\mathcal{M}_F$, the function $\mathbf{F}$. The proposed semantics induces a natural equivalence relation between FTCSP terms:

$P \simeq Q$ if and only if $\mathbf{F}[P] = \mathbf{F}[Q]$.

As for testing equivalence, we can break the natural equivalence induced by the denotational semantics into two preorders. We will have

$P \sqsubseteq Q$ if and only if $\mathbf{F}[P] \supseteq \mathbf{F}[Q]$

meaning intuitively that P is considered less defined than Q when P behaves more nondeterministically than Q; i.e. the external environment can control the progression of P in a smaller number of circumstances.

Let $\mathbf{C}[\ ]$ denote an FTCSP context, as for CCS in definition 2.2.3. It is well known that, because of the way the semantic function $\mathbf{F}$ has been defined, $P \sqsubseteq Q$ implies $\mathbf{C}[P] \sqsubseteq \mathbf{C}[Q]$ for every context $\mathbf{C}[\ ]$; i.e. $\sqsubseteq$ is a precongruence on FTCSP.

Proposition 4.1.2 $\sqsubseteq$ is preserved by all the operators in FTCSP. □

The next section will be completely dedicated to defining the axioms system which completely characterize the preorder $\sqsubseteq$. The following definitions, which allow the selection of particular subparts of a given refusals set F, will be useful.

Definition 4.1.3

i. $Init(F) = \{ a \mid \langle a, \{\}\rangle \in F \}$.

ii. $Traces(F) = \{ s \mid \langle s, \{\}\rangle \in F \}$

iii. $Refusals(F) = \{ V \mid \langle \varepsilon, V\rangle \in F \}$ □

4.2. A Complete Set of Axioms for a Theory of CSP

4.2.1. The axiom system

In this section we examine the axiom system corresponding to the preorder defined in the previous section. For every operator in the language we have a corresponding set of axioms. We have, moreover, some axioms showing the interrelationships between certain pairs of operators. The basic axioms are given in Table 4.1. Most of them are given in term of " = " and are intended to be read in conjunction with the rules:

$X = Y$ implies $X \sqsubseteq Y$ and $Y \sqsubseteq X$

$X \sqsubseteq Y$ and $Y \sqsubseteq X$ implies $X = Y$

Some of the axioms are stated in /BHR84/ others have been worked out by stressing the similarities of refusal sets with the models in /DH84/ and /Miln83/; in fact some of the

axioms are reminiscent of those in chapter 2, in spite of the very different set of operators used. H4 is a new one and is particularly relevant since it permits the hiding operator to be "pushed inside" terms and eventually to be eliminated.

If **A** is the set of axioms C1, E1-E4, I1-I4, D1-D4, H1-H5, PC1-PC5 we have:

Proposition 4.2.1 (**Soundness**)

The set of axioms **A** is **sound** for $\sqsubseteq$, i.e. if $P \sqsubseteq Q$ ($P = Q$) is an instance of an axiom from **A** then $P \sqsubseteq Q$ ($P \simeq Q$).

Proof: The proof consists in determining the refusal sets of P and Q, whatsoever, and comparing them. We will just outline the proof of some axioms which are not stated in /BHR84/ and /Bro83b/.

H4 . $(a \rightarrow X \,\Box\, Y)/a = X/a \sqcap (X \,\Box\, Y)/a$

We assume H3. It is then sufficient to prove $(a \rightarrow X \,\Box\, Y)/a = (X \sqcap (X \,\Box\, Y))/a$

$\mathcal{F}[a \rightarrow X \,\Box\, Y] = \{\langle as, T\rangle \mid \langle s,T\rangle \in \mathcal{F}[X]\} \cup \{\langle \varepsilon, W\rangle \mid \langle \varepsilon, W\rangle \in \mathcal{F}[Y] \text{ and } a \notin W\}$

$\cup \{\langle s, V\rangle \mid s \in A^+ \text{ and } \langle s, V\rangle \in \mathcal{F}[Y]\}$

$\mathcal{F}[X \sqcap (X \,\Box\, Y)] = \{\langle \varepsilon, T\rangle \mid \langle \varepsilon,T\rangle \in \mathcal{F}[X]\} \cup \{\langle s, V\rangle \mid s \in A^+ \text{ and } \langle s, V\rangle \in \mathcal{F}[Y] \cup \mathcal{F}[X]\}$

we thus have

$\mathcal{F}[(a \rightarrow X \,\Box\, Y)/a] = \{\langle as/a, T \cup \{a\}\rangle \mid \langle s,T\rangle \in \mathcal{F}[X]\} \cup$

$\{\langle s/a, V \cup \{a\}\rangle \mid s \in A^+ \text{ and } \langle s, V\rangle \in \mathcal{F}[Y]\} \cup$

$\{\langle (s/a)t, V\rangle \mid \langle t, V\rangle \in \mathcal{F}[CHAOS] \text{ and } \forall n \langle sa^n, \{\}\rangle \in \mathcal{F}[X] \cup \mathcal{F}[Y]\}$

$= \mathcal{F}[(X \sqcap (X \,\Box\, Y))/a]$

Equality is obtained by simply applying the definition of $/_F a$ to both the left and the right handside. It should be noted that, because of the way $/_F a$ is defined, the second set of refusals in the definition of $\mathcal{F}[(a \rightarrow X \,\Box\, Y)]$ does not make any contribution to $\mathcal{F}[(a \rightarrow X \,\Box\, Y)/a]$

Undefined

C1 $CHAOS \sqsubseteq X$

External Choice

E1 $X \,\Box\, X = X$

E2 $X \,\Box\, Y = Y \,\Box\, X$

E3 $X \,\Box\, (Y \,\Box\, Z) = (X \,\Box\, Y) \,\Box\, Z$

E4 $X \,\Box\, Stop = X$

Internal Choice

I1 $X \sqcap X = X$

I2 $X \sqcap Y = Y \sqcap X$

I3 $X \sqcap (Y \sqcap Z) = (X \sqcap Y) \sqcap Z$

I4 $X \sqcap Y \sqsubseteq X$

Distributive Laws

D1 $X \sqcap (Y \,\Box\, Z) = (X \sqcap Y) \,\Box\, (X \sqcap Z)$

D2 $X \,\Box\, (Y \sqcap Z) = (X \,\Box\, Y) \sqcap (X \,\Box\, Z)$

D3 $(a \rightarrow X) \sqcap (a \rightarrow Y) = a \rightarrow (X \sqcap Y)$

D4 $(a \rightarrow X) \,\Box\, (a \rightarrow Y) = a \rightarrow (X \sqcap Y)$

Hiding

H1 $Stop/a = Stop$

H2 $CHAOS/a = CHAOS$

H3 $(X \sqcap Y)/a = X/a \sqcap Y/a$

H4 $(a \rightarrow X \,\Box\, Y)/a = X/a \sqcap (X \,\Box\, Y)/a$

H5 $\Box\{b_i \rightarrow X_i \mid i \in I\}/a = \Box\{b_i \rightarrow X_i/a \mid i \in I\}$ if $\forall i \in I \; b_i \neq a$

Parallel Composition

If $P = \Box\{a_i \rightarrow P_i \mid i \in I\}$ and $Q = \Box\{b_j \rightarrow Q_j \mid j \in J\}$ then:

PC1 $P \,\|\, Q = \Box\{a_k \rightarrow (P_k \,\|\, Q_h) \mid a_k = b_h \text{ and } k \in I, h \in J\}$

PC2 $(P \,\Box\, CHAOS) \,\|\, Q = \Box\{a_k \rightarrow Stop \mid a_k = b_h \text{ and } k \in I, h \in J\} \sqcap \Box\{b_j \rightarrow (Q_j \,\|\, CHAOS) \mid j \in J\}$

PC3 $(P \,\Box\, CHAOS) \,\|\, (Q \,\Box\, CHAOS) = \Box\{a_k \rightarrow CHAOS \mid a_k = b_h \text{ and } k \in I, h \in J\} \,\Box\, CHAOS$

PC4 $X \,\|\, Y = Y \,\|\, X$

PC5 $(X \sqcap Y) \,\|\, Z = (X \,\|\, Z) \sqcap (Y \,\|\, Z)$

Table 4.1: An axiom system for finitary TCSP

PC2 $(P \,\square\, CHAOS) \,\|\, Q = \square\{a_k \rightarrow Stop \mid a_k = b_h \text{ and } k \in I, h \in J\} \sqcap \square\{b_j \rightarrow (Q_j \,\|\, CHAOS) \mid j \in J\}$

We have $\mathcal{F}[\![P \,\square\, CHAOS]\!] = \{\langle \varepsilon, V\rangle \mid V \subseteq A\text{-}Init(P)\} \cup \{\langle s, W\rangle \mid s \in A^+, W \subseteq A\}$, hence

$\mathcal{F}[\![lhs]\!] = \{\langle \varepsilon, V \cup W\rangle \mid V \subseteq A\text{-}Init(P) \text{ and } W \subseteq A\text{-}Init(Q)\} \cup$
$\{\langle s, Z\rangle \mid s \neq \varepsilon \text{ and } s \in traces(Q) \text{ and } Z \subseteq A\}$

Note also that, for any process R, $\mathcal{F}[\![R \,\|\, CHAOS]\!] = \{\langle s, W\rangle \mid s \in traces(R), W \subseteq A\}$, hence

$\mathcal{F}[\![rhs]\!] = \{\langle \varepsilon, V\rangle \mid V \subseteq A\text{-}(Init(P) \cap Init(Q))\} \cup \{\langle a, W\rangle \mid a \in (Init(P) \cap Init(Q)), W \subseteq A\}$
$\cup \{\langle \varepsilon, W\rangle \mid W \subseteq A\text{-}Init(Q)\} \cup \{\langle s, Z\rangle \mid s \neq \varepsilon \text{ and } s \in traces(Q) \text{ and } Z \subseteq A\}$

The result follows because in $\mathcal{F}[\![rhs]\!]$ the second and third operands of the union are included respectively in the fourth and first, and because for all sets A, B, C such that $B \subseteq A$ and $C \subseteq A$ we have that $\{V \cup W \mid V \subseteq A\text{-}B \text{ and } W \subseteq A\text{-}C\} = \{Z \mid Z \subseteq A\text{-}(B \cap C)\}$ □

Many additional equalities can be derived from the equations in Table 4.1, by using the rules given in page 114. We can use them because F is a Σ-cpo, /BHR84/. The new equalities will be very useful when proving the completeness theorem. They also give additional indications of the semantics of the various operators. A list of derivable axioms is given in Table 4.2. We now show how the axioms in Table 4.2 can be derived using those in Table 4.1.

DER1 $X \sqcap CHAOS \sqsubseteq CHAOS$ — by I4
$CHAOS \sqsubseteq X \sqcap CHAOS$ — by C1

DER2 $(X \sqcap Y) \,\square\, X = (X \,\square\, X) \sqcap (X \,\square\, Y)$ — by D1
$= X \sqcap (X \,\square\, Y)$ — by E1

DER3 $X \sqcap Y = (X \sqcap Y) \,\square\, (X \sqcap Y)$ — by E1
$= ((X \sqcap Y) \,\square\, X) \sqcap ((X \sqcap Y) \,\square\, Y)$ — by D2
$= ((X \,\square\, X) \sqcap (Y \,\square\, X) \sqcap (X \,\square\, Y) \sqcap (Y \,\square\, Y)$ — by D2, I3
$= X \sqcap Y \sqcap (X \,\square\, Y)$ — by E1, I1, I2

DER4 The proof is exactly symmetrical to that for DER3, i. e. we only need to interchange $\square$ and $\sqcap$.

DER1 $X \sqcap CHAOS = CHAOS$

DER2 $(X \sqcap Y) \,\square\, X = (X \,\square\, Y) \sqcap X$

DER3 $X \sqcap Y = X \sqcap Y \sqcap (X \,\square\, Y)$

DER4 $X \,\square\, Y = X \,\square\, Y \,\square\, (X \sqcap Y)$

DER5 $X \sqcap (X \,\square\, Y \,\square\, Z) = X \sqcap (X \,\square\, Y) \sqcap (X \,\square\, Y \,\square\, Z)$

DER6 $X \,\square\, (X \sqcap Y \sqcap Z) = X \,\square\, (X \sqcap Y) \,\square\, (X \sqcap Y \sqcap Z)$

DER7 $(a \rightarrow X_1 \,\square\, Y) \sqcap (a \rightarrow X_2 \,\square\, Z) = (a \rightarrow X_1 \,\square\, a \rightarrow X_2 \,\square\, Y) \sqcap (a \rightarrow X_1 \,\square\, a \rightarrow X_2 \,\square\, Z)$

DER8 $a \rightarrow \sqcap\{X_i \mid i \in I\} = \sqcap\{a \rightarrow X_i \mid i \in I\}$

DER9 $X \sqcap Y \sqsubseteq X \,\square\, Y$

DER10 $a \rightarrow X \,\square\, CHAOS = a \rightarrow CHAOS \,\square\, CHAOS$

DER11 $\square\{a_i \rightarrow X_i \mid i \in I\} \,\square\, CHAOS = \square\{a_i \rightarrow CHAOS \mid i \in I\} \,\square\, CHAOS$

DER12 $CHAOS \,\|\, a \rightarrow X = Stop \sqcap a \rightarrow (X \,\|\, CHAOS)$

Table 4.2: Derivable Axioms

DER5 $X \sqcap (X \,\Box\, Y) \sqcap (X \,\Box\, Y \,\Box\, Z)$

$= (X \,\Box\, (X \sqcap Y)) \sqcap (X \,\Box\, Y \,\Box\, Z)$ by DER2

$= X \sqcap (X \,\Box\, Y \Box\, Z) \Box (X \sqcap Y) \sqcap (X \,\Box\, Y \,\Box\, Z)$ by D1

$= (X \sqcap X) \,\Box\, (X \sqcap Y) \,\Box\, (X \sqcap Z) \,\Box\, (X \sqcap Y \sqcap Z) \,\Box\, (X \sqcap Y \sqcap Y) \,\Box\, (X \sqcap Y \sqcap Z)$ by D1

$= (X \sqcap X) \,\Box\, (X \sqcap Y) \,\Box\, (X \sqcap Z) \,\Box\, (X \sqcap Y \sqcap Z)$ by E1, I1

$= (X \sqcap X) \,\Box\, (X \sqcap Y) \,\Box\, (X \sqcap Z)$ by E1, DER4, D1

$= X \sqcap (X \,\Box\, Y \,\Box\, Z)$ by D1

DER6 The proof is symmetric DER5.

DER7 The proof is too long and tedious to be reported here; it uses

$a \rightarrow X_1 \sqcap (a \rightarrow X_2 \,\Box\, Z) = a \rightarrow (X_1 \sqcap X_2) \,\Box\, (a \rightarrow (X_1 \sqcap X_2) \sqcap (a \rightarrow X_2 \,\Box\, Z)$

and relies heavily on D1 - D4. It has been shown to the author by M. Hennessy. The interested reader is referred to /Hen83/.

DER8 is proved by induction on the size of I, using D3 as the basis.

DER9 $X \sqcap Y \sqsubseteq X \sqcap Y \sqcap (X \,\Box\, Y)$ by DER3

$\sqsubseteq X \,\Box\, Y$ by I4.

DER10 $(a \rightarrow CHAOS) \,\Box\, CHAOS \sqsubseteq (a \rightarrow X) \,\Box\, CHAOS$ by C1

$(a \rightarrow X) \,\Box\, CHAOS \sqsubseteq (a \rightarrow X) \,\Box\, (a \rightarrow CHAOS) \,\Box\, CHAOS$ by E1, C1

$\sqsubseteq (a \rightarrow (X \sqcap CHAOS)) \,\Box\, CHAOS$ by D4

$\sqsubseteq (a \rightarrow CHAOS) \,\Box\, CHAOS$ by DER1.

DER11 is proved by induction on the size of I, using DER10 as basis.

DER12 is an instance of PC2, where P = Stop and Q = $a \rightarrow X$. In fact from E4 we have

Stop $\,\Box\,$ CHAOS = CHAOS

The derived axioms DER10 and DER12 which describe the interaction of CHAOS (the process which can exhibit the maximum of nondeterminism) with other processes, deserve some comment. We must first understand the operational intuition behind CHAOS and its role

in defining the semantics of FTCSP. CHAOS is used to denote the least defined process, which may take an infinite amount of time to reply to any request from other processes. Moreover, it is used to denote processes which can perform an infinite sequence of invisible (concealed) actions, i.e. processes which are divergent in the sense defined in Chapters 1 and 2. For instance, in the general language for a theory of CSP we have $\mathbf{f}[(\text{rec } x.\ a \rightarrow x)/a] = CHAOS_F$. If we give this operational interpretation of the process CHAOS then some instances of the derivable axioms DER10 and DER12 will not correspond to the traditional view of systems behaviour and systems reaction to external experiments. The operational implication of the above mentioned derived axioms is discussed in detail below.

DER10 states that a completely specified process P in alternative with the process CHAOS behaves in the same way as an agent which refuses the same experiments which CHAOS would refuse except that it will always accept an a-experiment when this is proposed by the environment. This can also be interpreted as meaning the when the environment is offered a choice between a completely specified process P and a divergent process the only actions of P which will be seen by the environment are its initial actions. This differs from the traditional operational view of diverging processes and from the traditional way of handling them. Usually **divergence** is

1) considered **catastrophic** or

2) considered **under-specification** or

3) **ignored**

and the expected axioms would be

1) $a \rightarrow P \,\Box\, CHAOS = CHAOS$

2) $a \rightarrow P \,\Box\, CHAOS = (a \rightarrow P \,\Box\, CHAOS) \,\Box\, CHAOS$

3) $a \rightarrow P \,\Box\, CHAOS = a \rightarrow P$.

DER12, on the contrary, states that the behaviour of the process resulting from the parallel composition of any process with the process CHAOS, is modelled by refusal sets in

such a way that it does not completely take into account the possibility of divergence. In fact, while the left side of DER12 can perform an infinite sequence of internal moves from the very beginning, its right side cannot perform any initial silent move. We think that the possibility of performing infinite silent actions from the beginning should be taken into account since, as we have seen, CHAOS can be used to model infinite chattering (infinite internal exchange of messages) and ⫴ is defined in such a way that if we have P⫴Q then P does not need to synchronize with Q to perform an invisible action.

The difficulties in coping with CHAOS also prevented us from axiomatizing ⫴, the operator used in /BHR84/ to model the interleaving of the actions of two processes. While an axiomatization has been found which allows us to remove all the occurrences of the operators ⫴ and /a from every process, we have not been able to find a process expressed only in term of □, Π, a→, CHAOS and Stop with the same denotation as P⫴CHAOS. I.e. we could not find another process which can be denoted by a refusal set which has the same traces of CHAOS but can refuse only those actions which P can refuse. This is already evident if we take P = a→Stop. In fact ⫴ cannot be eliminated from a→Stop ⫴ CHAOS, since we are unable to write a term which does not contain ⫴ but denotes a process which is able to accept (not to reject) an a-experiment after every s ∈ A.

All these problems seem to suggest that, although appealing, it is either inappropriate to model divergent processes as processes with the maximum of nondeterminism or more care is needed when modelling their interactions with other processes. In the second half of this chapter we will present a way to overcome the above mentioned difficulties by slightly changing the semantic domain.

4.2.2 The completeness theorem

This section is entirely dedicated to proving that the set of axioms in Table 4.1 completely characterizes the preorder ⊑. In this case too, the completeness proof rests heavily on the existence of **normal forms** for processes. As for CCS normal forms, the definition of FTCSP normal forms (throughout the chapter they will simply be called normal forms) will make use of saturated sets as described in Definition 2.5.5.

Through the rest of the section we use "⊑" to denote the least TCSP-precongruence generated by the axioms in **A**. The related congruence is denoted by " = ".

Definition 4.2.2 P is in **normal form** if it is of the form:

Π { □ { a→P(a) | a ∈ L } | L ∈ **L** } [□CHAOS] where

- **L** is a non-empty saturated set;
- P(a) are in normal form;
- [□CHAOS] denotes that CHAOS is an optional summand and if CHAOS is a summand then for all a ∈ A(**L**) we have that P(a) is CHAOS. □

Some examples will clarify this definition. In the examples, the subprocesses of the kind a→Stop will be rendered as a for reasons of simplicity.

1. a→(b □ c)□ CHAOS is not in normal form because P(a) is different from CHAOS

2. $(a_1 \to b) \sqcap (a_2 \to c)$ is not in normal form because $\{\{a_1\}, \{a_2\}\}$ is not saturated, i.e. there is no subterm corresponding to the set $\{a_1, a_2\}$, the corresponding normal form is $a_1 \to b \sqcap a_2 \to c \sqcap (a_1 \to b \,\Box\, a_2 \to c)$.

3. a→b Π (a→c □ d→b) is not in normal form because in every normal form, for every action a, there is at most one P such that a→P is a summand of a normal form. The corresponding normal form is: a→(b Π c) Π (a→(b Π c) □ d→b).

4. Stop and Stop $\Box$ CHAOS are in normal form.

It is possible to prove that every process can be reduced to an equivalent one which is in normal form by using axioms from **A**. The proof follows the same pattern as the corresponding proofs for CCS, in Chapter 2. In this case the proof will be somewhat easier since, because of the absence of internal moves, there is no need to differentiate between τ-normal forms and a-normal forms. When reducing terms to normal forms, the axioms for the external choice operator, $\Box$, will play exactly the same role as the axioms for the nondeterministic CCS operator, +, while the axioms for the internal choice operator, Π, will play the same role as CCS's τ-laws; roughly speaking $p \Pi q$ corresponds to $\tau.p + \tau.q$. In CCS axiomatization there is no correspondent of the distributivity laws but, taking into account the latter similarity, some of the ones of TCSP can be seen as instances of τ-laws.

The close correspondence between the axioms for CCS and TCSP suggests that there is also a correspondence between the two languages. In particular, it is possible to establish the relationship between the sets of their normal forms, which do not contain any divergent subterm, i.e. between normal forms which do not contain Ω or CHAOS.

Let **NF** denote the set of FTCSP normal forms which do not contain CHAOS and **SNF** denote the set of CCS strong normal forms which do not contain Ω (**SNF** are a subset of both NF_1 and NF_2 of section 3.3). We can define two functions:

tr: **NF** $\rightarrow$ **SNF**

rt: **SNF** $\rightarrow$ **NF**

which map CCS normal forms into FTCSP normal forms and viceversa.

i. $tr(Stop) = NIL$

ii. $tr(\Box\{a \rightarrow n(a) \mid a \in L\}) = \Sigma\{a.tr(n(a)) \mid a \in L\}$

iii. $tr(\Pi\{P_i \mid i \in I\}) = \Sigma\{\tau.tr(P_i) \mid i \in I\}$

and

i. $rt(NIL) = Stop$

ii. $rt(\Sigma\{a.n(a) \mid a \in L\}) = \Box\{a \rightarrow rt(n(a)) \mid a \in L\}$

iii. $rt(\Sigma\{\tau.P_i \mid i \in I\}) = \Pi\{rt(P_i) \mid i \in I\}$

Given these definitions, it is not difficult to see that we have:

1. $rt(tr(nf)) = nf$ and

2. $tr(rt(snf)) = snf'$ with $snf' \approx snf$

($\approx$ being the observational equivalence of /Mil80).

Thanks to this relation and to the similarities of the two sets of axioms, in the following, instead of giving the detailed proofs for the theorems which establish the existence of normal forms for FTCSP, we will simply outline the proofs and stress the similarities with the corresponding ones of Chapter 2. We feel that, in order to understand the nature of such proofs, this approach is more helpful than repeating the details of every calculation. The interested reader can find the detailed proofs in /DeN83/.

Lemma 4.2.3 If P is in normal form then

1. $P \Pi CHAOS$ has a normal form

2. $P \Box CHAOS$ has a normal form.

Proof:

1. $P \Pi CHAOS = CHAOS = Stop \Box CHAOS$ by C1 and E4.

2. $P \Box CHAOS = \Pi\{(\Box\{a \rightarrow P(a) \mid a \in L\}) \mid L \in \mathbf{L}\} \Box CHAOS$

$= \Pi\{(\Box\{a \rightarrow P(a) \mid a \in L\}) \Box CHAOS \mid L \in \mathbf{L}\}$ by D2

$= \Pi\{(\Box\{a \rightarrow CHAOS \mid a \in L\}) \Box CHAOS \mid L \in \mathbf{L}\}$ by DER11

$= \Pi\{(\Box\{a \rightarrow CHAOS \mid a \in L\}) \mid L \in \mathbf{L}\} \Box CHAOS$ by D2 $\Box$

The above lemma corresponds to Lemma 2.5.9 whereas that which follows corresponds to Lemma 2.5.10.

Lemma 4.2.4 If P and Q are in normal form then there is a normal form R such that $R = P \sqcap Q$.

Proof: The proof is similar to the corresponding one of Chapter 2.

$P \sqcap Q = S = \prod \{ (\ \Box \{a \rightarrow S_i \mid a \in L, i \in I\}) \mid L \in \mathbf{L}_1 \cup \mathbf{L}_2 \}$

S may not be in normal form for a number of reasons. We will show that it is possible to reduce S to a normal form, whatever the reason.

1. S is not in normal form because there are pairs $a \rightarrow S_1$, $a \rightarrow S_2$ which are summands of S and S_1 is syntactically different from S_2, i.e. S is of the form

$S = (a \rightarrow S_1 \Box S'_1) \sqcap (a \rightarrow S_2 \Box S'_2) \sqcap S'$.

The proof that S can be reduced to a normal form uses DER7 instead of D8 and N3, and uses D4 instead of N1.

2. Another reason why S may not be in normal form is that the set $\mathbf{L}_1 \cup \mathbf{L}_2$, from now on denoted by $\mathbf{L}$, is not saturated; i.e. $\exists K$ such that $L \subseteq K \subseteq A(\mathbf{L})$ for some $L \in \mathbf{L}$ and $\Box \{a \rightarrow S(a) \mid a \in K\}$ is not a summand of S. We can prove, by induction on K, that $S = S \sqcap \Box \{a \rightarrow S(a) \mid a \in K\}$. K may be written as $K_1 \cup \{a_0\}$ for some $a_0 \in L, L \in \mathbf{L}$. We assume that

$S = S \sqcap S_1$ where S_1 denotes $\Box \{a \rightarrow S(a) \mid a \in K_1\}$

and we prove that

$S = S \sqcap (\ \Box \{a \rightarrow P(a) \mid a \in K_1 \cup \{a_0\}\})$

by using I1 - I3 instead of A1 - A3, DER3 instead of D5 and finally DER6 instead of D6.

3. The last case to consider is when P or Q have CHAOS as a summand. We have

$(P \Box CHAOS) \sqcap Q = (P \sqcap Q) \Box (CHAOS \sqcap Q)$ by D1, I2

$= (P \sqcap Q) \Box CHAOS$ by DER1

the latter has normal form because of cases 1. and 2. above and Proposition 4.2.3.

□

Corollary 4.2.5 If P_i are in normal form then there exists a normal form R such that $R = \prod \{P_i \mid i \in \{1,\ldots,n\}\}$.

Proof: By induction on n, as for Corollary 2.5.12. □

Proposition 4.2.6 If P and Q are in normal form then there exists a normal form R such that $R = P \Box Q$.

Proof: This proof is slightly different from the one of the corresponding lemma 2.5.11 and uses heavily the distributivity laws D1 - D4. Let S denote $P \Box Q$

$S = (\prod \{ \Box \{a \rightarrow P(a) \mid a \in L\} \mid L \in \mathbf{L}\}) \Box (\prod \{ \Box \{b \rightarrow Q(b) \mid b \in K\} \mid K \in \mathbf{K}\})\ [\Box CHAOS]$

$= \prod \{(\ \Box \{a \rightarrow P(a) \mid a \in L\}) \Box (\prod \{ \Box \{b \rightarrow Q(b) \mid b \in K\} \mid K \in \mathbf{K}\}) \mid L \in \mathbf{L}\}\ [\Box CHAOS]$

$= \prod \{\prod \{(\ \Box \{a \rightarrow P(a) \mid a \in L\}) \Box (\{\Box \{b \rightarrow Q(b) \mid b \in K\}) \mid K \in \mathbf{K}\} \mid L \in \mathbf{L}\}\ [\Box CHAOS]$

(All the transformations can be obtained by repeated use of the distributivity axioms)

We can now find a normal form

R_1 for $\Box \{a \rightarrow P(a) \mid a \in L\} \Box \Box \{b \rightarrow Q(b) \mid b \in K\}$ for each $k \in \mathbf{K}, L \in \mathbf{L}$,

applying transformations similar to the ones used in part 1 and 3 of the proof of Lemma 4.2.4. By applying Corollary 4.2.5 twice and from Proposition 4.2.3 we can transform S into a normal form R. □

Corollary 4.2.7 If P_i are in normal form then there exists a normal form R such that $R = \Box \{P_i \mid i \in \{1,\ldots,n\}\}$.

Proof: By induction on n. □

Proposition 4.2.8 For every process $P \in TCSP$ there exists a normal form $nf(P)$ such that $nf(P) = P$.

Proof: Using the axioms PC1-PC5, H1-H5 and C1 it is possible to reduce P to a term containing only the operators $\Box$, $\sqcap$, $a\rightarrow$, and the basic processes CHAOS and Stop. It is then

possible to prove by structural induction on P, mainly using D2, that P can be reduced to a **standard form** $\Pi\{ \Box \{a \to Q_i \mid a \in L, i \in I\} \mid L \in \mathbf{L}\} [\,] \mathrm{CHAOS}]$. By induction on P we may assume that each Q_i is in normal form. From corollary 4.2.7 there exists a normal form $n = \Box \{a \to Q_a \mid a \in L\}$ for all $L \in \mathbf{L}$, and thus from proposition 4.2.5 there exists $nf(P) = \Pi \{ \Box \{a \to Q_a \mid a \in L\} \mid L \in \mathbf{L}\} [\,] \mathrm{CHAOS}]$. □

Normal forms and the same techniques of Section 2.5.3 will now be used to prove the completeness theorem.

Definition 4.2.9 If n and m are normal forms and

$n = \Pi \{ \Box \{ a \to n(a) \mid a \in N \} \mid N \in \mathbf{L}(n) \} [\,] \mathrm{CHAOS}\,]$

$m = \Pi \{ \Box \{ b \to m(b) \mid b \in M \} \mid M \in \mathbf{L}(m) \} [\,] \mathrm{CHAOS}\,]$ then

$n < m$ if

1. for all $M \in \mathbf{L}(m)$ there exists $N \in \mathbf{L}(n)$ such that $M \supseteq N$
2. if CHAOS is not a summand of n then it is not a summand of m and $\mathbf{L}(n) \supseteq \mathbf{L}(m)$. □

Definition 4.2.10 If n and m are as in the previous definition then

$n \lessdot m$ if

1. $n < m$ and
2. $n(a) \lessdot m(a)$ whenever both are defined. □

Lemma 4.2.11 If m and n are normal forms and CHAOS is not a summand of n then $n \sqsubseteq m$ implies $\mathrm{Init}(n) \supseteq \mathrm{Init}(m)$.

Proof: Suppose there exists $a \in A$ such that $a \in \mathrm{Init}(m)$ and $a \notin \mathrm{Init}(n)$.

Then we have $\langle a, \{\}\rangle \in \mathbf{F}[\![m]\!]$ and

$\langle a, \{\}\rangle \notin \mathbf{F}[\![n]\!]$ which contradicts $\mathbf{F}[\![n]\!] \supseteq \mathbf{F}[\![m]\!]$. □

Lemma 4.2.12 If n and m are normal forms then $n \sqsubseteq m$ implies $n < m$.

Proof: We have to prove that n and m satisfy Conditions 1. and 2. of Definition 4.2.9.

1. Suppose there exists $M_0 \in \mathbf{L}(m)$ such that for all $N \in \mathbf{L}(n)$ there exists $a \in N$ such that $a \notin M_0$. If we choose $X = \{ a \mid a \notin M_0 \text{ and } a \in A(\mathbf{L}(n))\}$ we have that $\{\langle \varepsilon, X\rangle \in \mathbf{F}[\![m]\!]\}$ while $\langle \varepsilon, X\rangle \notin \mathbf{F}[\![n]\!]$, which contradicts $\mathbf{F}[\![n]\!] \supseteq \mathbf{F}[\![m]\!]$.

2. We have to prove that if CHAOS is not a summand of n then

   a. CHAOS is not a summand of m and

   b. $\mathbf{L}(n) \supseteq \mathbf{L}(m)$]

a. Suppose CHAOS is a summand of m, then $\forall a \in A$ such that $a \notin \mathrm{Init}(n)$ we have $\langle a, \{\}\rangle \in \mathbf{F}[\![m]\!]$ while $\langle a, \{\}\rangle \notin \mathbf{F}[\![n]\!]$.

b. From Lemma 4.2.11, we have $A(\mathbf{L}(n)) \supseteq A(\mathbf{L}(m))$. Moreover $\mathbf{L}(n)$ is saturated. Thus in order to prove the claim it is sufficient to show that $M \in \mathbf{L}(m)$ implies that there exists $N \in \mathbf{L}(n)$ such that $M \supseteq N$. The proof is now similar to that for case 1. □

Lemma 4.2.13 If n and m are normal forms then $n \sqsubseteq m$ implies $n(a) \sqsubseteq m(a)$ whenever both exist.

Proof: If CHAOS is a summand of n then $n(a) = \mathrm{CHAOS}$ and the claim is verified by C1. If CHAOS is not a summand of n we prove that for all $a \in \mathrm{Init}(m)$, $\langle s, V\rangle \in \mathbf{F}[\![m(a)]\!]$ implies $\langle s, V\rangle \in \mathbf{F}[\![n(a)]\!]$. Note that n(a) always exists because of Lemma 4.2.11. We have that $\langle s, V\rangle \in \mathbf{F}[\![m(a)]\!]$ implies $\langle as, V\rangle \in \mathbf{F}[\![m]\!]$ because of the structure of normal forms, by hypothesis, this in turn implies $\langle as, V\rangle \in \mathbf{F}[\![n]\!]$. Since, if n is in normal form there exists a unique n(a) such that $a \to n(a)$, we have that $\langle s, V\rangle \in \mathbf{F}[\![n(a)]\!]$. □

Lemma 4.2.14 If n and m are normal forms then $n \sqsubseteq m$ implies $n \lessdot m$.

Proof: From the previous lemma we can assume $n(a) \sqsubseteq m(a)$ whenever both are defined. By induction we have $n(a) \lessdot m(a)$; the result then follows from Lemma 4.2.12.

□

Lemma 4.2.15 If n and m are normal forms then $n \not\preccurlyeq m$ implies $n \sqsubseteq m$.

Proof: We have to distinguish two cases:

1. CHAOS is a summand of n.

In this case we have that each n(a) is CHAOS. Let $\mathbf{K} = \{ V \mid V \in \mathbf{L}(n)$ and not $\exists M \in \mathbf{L}(m)$ such that $V \subseteq M \}$. Using axioms E2, E3, I2 and I3 many times m can be rearranged in such a way that we have, for some $M_2 \in FPOW(A(\mathbf{L}(m)))$:

$$m = \prod\{(\ \Box\{a \to m(a) \mid a \in M_1\}) \ \Box\ (\Box\{b \to m(b) \mid b \in M_2\}) \mid M_1 \in \mathbf{L}(n) - \mathbf{K}\} \ \Box CHAOS$$

$$\supseteq \prod\{(\ \Box\{a \to m(a) \mid a \in M_1\}) \Box CHAOS \mid M_1 \in \mathbf{L}(n) - \mathbf{K}\} \ \Box CHAOS \quad \text{by C1}$$

$$\supseteq \prod\{(\ \Box\{a \to CHAOS \mid a \in M_1\}) \mid M_1 \in \mathbf{L}(n) - \mathbf{K}\} \sqcap$$
$$\prod\{(\ \Box\{b \to CHAOS \mid b \in K\}) \mid K \in \mathbf{K}\} \Box CHAOS \quad \text{by C1}$$

$$= n \quad \text{by I2, I}$$

2. CHAOS is not a summand of n.

By induction we may assume that $n(a) \sqsubseteq m(a)$ whenever both are defined. Moreover we also have that CHAOS is not a summand of m and from Lemma 4.2.12 and I1, I2 we have

$$n = \prod\{(\ \Box\{a \to n(a) \mid a \in L\}) \mid L \in \mathbf{L}(m)\} \sqcap$$
$$\prod\{(\ \Box\{b \to n(b) \mid b \in K\}) \mid K \in \mathbf{L}(n) - \mathbf{L}(m)\}$$

$$\sqsubseteq \prod\{(\Box\{a \to m(a) \mid a \in L\}) \mid L \in \mathbf{L}(m)\} \sqcap$$
$$\prod\{(\ \Box\{b \to n(b) \mid b \in K\}) \mid K \in \mathbf{L}(n) - \mathbf{L}(m)\}$$

because $n(a) \sqsubseteq m(a)$ by induction

$$\sqsubseteq \prod\{(\ \Box\{a \to m(a) \mid a \in L\}) \mid L \in \mathbf{L}(m)\} \quad \text{by I4} \qquad \Box$$

We are now ready to state the main theorems.

Theorem 4.2.16 (Completeness)

For every $P, Q \in FTCSP$, $P \precsim Q$ implies $\mathbf{A} \vdash P \sqsubseteq Q$, ($P \sqsubseteq Q$ can be derived using only equations from **A**).

Proof: This proof follows directly from the existence of normal forms for P and Q (Proposition 4.2.8) and from Lemmas 4.2.14 and 4.2.15. $\Box$

Theorem 4.2.17 $\precsim$ is the least precongruence on TCSP generated by the axioms **A**.

Proof: This proof follows from the fact that $\precsim$ is a precongruence (Proposition 4.1.4), that the set of axioms **A** is sound (Proposition 4.2.1) and from the previous theorem. $\Box$

4.3. Bounded Refusal Sets: A New Domain for Processes

The algebraic characterization of the congruence induced by refusal sets has shed light on difficulties the chosen semantics for TCSP has in capturing the operational intuition of the interactions between completely specified processes and CHAOS, the fully undefined one; particular examples have been discussed at the end of section 4.2.1. We have also seen how problems connected with these difficulties prevented the axiomatization of the operator $\|$ (interleaving) and the extension of the axiom system to infinite, recursively defined processes.

In this section we impose an additional condition on the domain of refusal sets, thus getting a new domain for processes which enforces new constraints on the semantic equations and obliges us to give a new semantics to $\|$, $\Box$ and $\|$. We will show that the new domain enjoys important inductive properties which allow to extend the axiomatic characterization to infinite processes.

Many of the above mentioned problems seem to be caused by refusal sets which have traces with an infinite number of successors. If we are interested in processes which exhibit only bounded nondeterminism we may take a more consistent approach with respect to "unbounded" refusal sets. We assume that any process P which can perform a sequence of actions s with an

infinite number of possible next actions may spend all its time to decide the next move. Given this assumption, if P has a traces s with an infinite number of successors then the behaviour of P after it has performed the trace s may be considered identical to the behaviour of a process which may perform an infinite number of internal, invisible actions (e.g. (rec x. a→x)/a) and thus refuse any action proposed by the external environment. Indeed the denotation for (rec x. a→x)/a) chosen in /BHR81/ is the same as the one for CHAOS. We impose a constraint on refusal sets which forces the denotation of processes or subprocesses with an infinite number of initial moves to coincide with that of CHAOS.

The refusal set domain ordered by the superset relation $(F, \supseteq)$ of Section 4.1 is a subset of $POW(A^* \times POW(A))$ such that any of its elements satisfies condition 1.-5. of Section 4.1. To define the new domain of bounded refusal sets we impose an additional condition.

Definition 4.3.1 Let $\mathbf{B}$ be a subset of F such that every $F \in \mathbf{B}$ satisfies the following condition:

6. If $\{a \mid \langle sa, \{\}\rangle \in F\}$ is infinite then $\{\langle st, X\rangle \mid t \in A^*, X \subseteq A\} \subseteq F$. □

In the next section we will use $\mathbf{B}$ to give the semantics of a more complete version of TCSP. First, we prove that $\mathbf{B}$, ordered by reverse inclusion, enjoys particular topological properties (continuity, algebraicity, etc.). We prove this by stressing the similarities between sets of refusals (downward closed set of finite sets) and the sets of their maximals.

First, we prove some general properties of downward closed sets of finite sets which will be useful in the rest of the chapter. Let $\mathbf{K}$ be a set of sets.

Definition 4.3.2

i. $\mathbf{K}$ is **downward closed** if $X \in \mathbf{K}$ and $Y \subseteq X$ implies $Y \in \mathbf{K}$.

ii. An element X of $\mathbf{K}$ is said to be **maximal** if $Y \in \mathbf{K}$ and $Y \supseteq X$ implies $Y = X$.

iii. $\mathbf{K}$ is a **chain** if $X, Y \in \mathbf{K}$ implies $X \subseteq Y$ or $Y \subseteq X$. □

Given any downward closed set of finite sets, ordered by inclusion, we can define a set of maximals which uniquely determines it. We need two functions on downward closed sets and one relation on sets of maximals.

Definition 4.3.3

- Completion($\mathbf{K}$) = $\{X \mid X = \bigcup\{X_i \mid i \in I\}$ and $\{X_i \mid i \in I\}$ is a chain of elements of $\mathbf{K}\}$
- Maximals($\mathbf{K}$) = $\{M \mid M \in \mathbf{K}$ and M is a maximal$\}$
- If $\mathbf{M}_1$ and $\mathbf{M}_2$ are two sets of maximals then $\mathbf{M}_1 \sqsubset \mathbf{M}_2$ if and only if for all $M_1 \in \mathbf{M}_1$ there exists $M_2 \in \mathbf{M}_2$ such that $M_1 \subseteq M_2$. □

Given any downward closed set $\mathbf{K}$, Completion($\mathbf{K}$) adds to it all the limit points of its chains. On the other hand side Maximals($\mathbf{K}$) removes from $\mathbf{K}$ all the sets N which are not maximals.

We are now ready to prove that any downward closed set of finite sets is completely determined by the set of its maximals and that the inclusion relation on downward closed sets is completely determined by $\sqsubset$.

Lemma 4.3.4 If $\mathbf{K}_1$ and $\mathbf{K}_2$ are downward closed sets of finite sets and

$\mathbf{M}_i$ = Maximals(Completion($\mathbf{K}_i$)) with i = 1,2;

$\mathbf{N}_1 = \{X \mid X \in FPOW(M)$ for some $M \in \mathbf{M}_1\}$ then

i. $\mathbf{K}_1 = \mathbf{N}_1$

ii. $\mathbf{K}_1 \subseteq \mathbf{K}_2$ if and only if $\mathbf{M}_1 \sqsubset \mathbf{M}_2$.

Proof: 1. We will prove that $\mathbf{K}_1 \subseteq \mathbf{N}_1$ and viceversa.

a. If $X \in \mathbf{K}$ then certainly there exists a chain of elements of $\mathbf{K}$ whose limit is X (in fact X is the limit of the trivial chain $\{X\}$). This, by Zorn Lemma(*), implies that there exists a

---

(*) Every non-empty partially ordered set X, in which every chain has an upper bound, has a maximal element, /Hal60/.

maximal $M \supseteq X$, i.e we have that there exists $M \in \mathbf{M}_1$ such that $X \in FPOW(M)$.

b. If $X \in \mathbf{N}_1$ then, by hypothesis, there exists a chain of elements of $\mathbf{K}_1$, $\{K_i \mid i \geq 0\}$ whose limit L is bigger than X. Since X is finite we have that there exists $n \geq 0$ such that K_n is an element of the chain whose limit is L and $X \subseteq K_n$; the latter together with the fact that $\mathbf{K}_1$ is downward closed implies $X \in \mathbf{K}_1$.

2. We prove first *not*($\mathbf{K}_1 \subseteq \mathbf{K}_2$) implies $\mathbf{M}_1 \not\sqsubset \mathbf{M}_2$ and then that

$\mathbf{K}_1 \subseteq \mathbf{K}_2$ implies $\mathbf{M}_1 \sqsubset \mathbf{M}_2$.

a. Suppose there exists a set X such that $X \in \mathbf{K}_1$ and $X \notin \mathbf{K}_2$. From the proof in part 1a. we know there exists $M \supseteq X$ such that $M \in \mathbf{M}_1$. Moreover since $\mathbf{K}_2$ is downward closed $X \notin \mathbf{K}_2$ implies that for all finite $Y \supseteq X$ we have that $Y \notin \mathbf{K}_2$; by reasonings similar to those in part 1b. we can deduce that for all (finite or infinite) Y, $Y \supseteq X$ we have $Y \notin$ Completion($\mathbf{K}_2$). This together with the existence of $M \in \mathbf{M}_1$ which is larger than X implies there exists $M_1 \in \mathbf{M}_1$ such that for no $M_2 \in \mathbf{M}_2$ we have $M_1 \subseteq M_2$; i.e. $\mathbf{M}_1 \not\sqsubset \mathbf{M}_2$.

b. By definition we have that $\mathbf{K}_1 \subseteq \mathbf{K}_2$ implies Completion($\mathbf{K}_1$) $\subseteq$ Completion($\mathbf{K}_2$) and this, in turn, by simple reasoning, implies $\mathbf{M}_1 \sqsubset \mathbf{M}_2$. □

This lemma offers the possibility of describing any refusal set in terms of the set of its traces and the sets of maximals of its refusals and to characterize the ordering between two refusal sets in terms of the relation between the sets of their maximals. This is expressed in the next characterization theorem. Note that this theorem holds for general refusal sets since we never used condition 6. in the proof of Lemma 4.3.4. Before stating the theorem we need some additional definitions:

Definition 4.3.5 If F is an element of $\mathbf{B}$, we have:

- F↑s (read F diverges on s) iff $\{\langle st, X\rangle \mid t \in A^*, X \subseteq A\} \subseteq F$;
- F↓s (read F converges on s) iff *not*(F↑s);
- $\mathbf{R}(s, F)$ = Maximal (Completion (Refusals (s, F))). □

It should be noted that, since $\mathbf{B} \subseteq \mathbf{F}$, we have that the definitions of Refusals, Init and Traces of section 4.1 still hold for elements of $\mathbf{B}$.

Theorem 4.3.6 (Characterization Theorem)

a. Every refusal set F is uniquely determined by

- Traces(F) - the non empty set of its traces; and
- $\mathbf{R}(s, F)$ - the set of the maximals of the refusals associated with every trace.

b. Given two refusal sets F_1 and F_2 we have

- $F_1 \subseteq F_2$ if and only if
  1. Traces(F_1) $\subseteq$ Traces(F_2) and
  2. $s \in$ Traces(F_1) implies $\mathbf{R}(s, F_1) \sqsubset \mathbf{R}(s, F_2)$.

Proof: The proof follows directly from Lemma 4.3.4. □

We will now use the last theorem to prove that the domain of bounded refusal sets enjoys important inductive properties.

Definition 4.3.7 A subset B of any set A is said to be **cofinite** w.r.t. A if and only if A - B is finite. □

Proposition 4.3.8 If $F \in \mathbf{B}$ then for any $s \in$ Traces(F) we have that Refusals(s, F) is a non-empty downward closed set of finite subsets of A (the set of elementary actions).

Proof: Follows directly from Conditions 1., 2. and 4. of Section 4.1. □

Proposition 4.3.9 If $F \in \mathbf{B}$ and $s \in$ Traces(F) then $\mathbf{R}(s, F)$ is a finite set of cofinite subsets of A.

Proof: We have to distinguish two cases: i) F↑s and ii) F↓s.

In case i) we have that $\mathbf{R}(s, F) = \{A\}$ by condition 6.

In case ii) because of condition 5. of section 4.1 we have that

$X \in$ Refusals(s, F), Y finite and $Y \subseteq$ A-Succ(s, F) implies $X \cup Y \in$ Refusals(s, F)

i.e.

$X \in$ Refusals(s, F) implies $\{ X \cup Y \mid Y \in FPOW (A\text{-}Succ(s, F))\} \subseteq$ Refusals(s, F)

and this implies that for all $R \in \mathbf{R}(s, F)$ we have $A - Succ(s, F) \subseteq R \subseteq A$. Since F↓s, by condition 6. we have that Succ(s, F) is finite and this in turn implies that every single $R \in \mathbf{R}(s, F)$ is cofinite w.r.t. A and that $\mathbf{R}(s, F)$ is finite. □

This proposition is particularly important since it shows that only a finite amount of information is needed to talk about refusals sets. This holds even if we remove the condition that all the refusals associated to a process have to be finite. This fact will be very useful to study the relationships between Refusals Sets and Representation Trees. In fact, we will have that both these domains are determined by a prefix closed set of sequences of actions (a deterministic tree) and a set of actions associated to every sequence which describes the future after that sequence. The refusals associated with every sequence (cofinite sets of sets) in the domain of Refusal Sets will be just the complements with respect to A of the acceptance sets (finite sets of sets) associated to the same sequence in the domain of Representation Trees.

We are now ready to prove the main theorem of the section, it relies on the above proposition and on the characterization theorem.

Theorem 4.3.12

1. ($\mathbf{B}$, $\supseteq$) is a complete partial order;
2. ($\mathbf{B}$, $\supseteq$) is an algebraic complete partial order whose finite elements are all the bounded refusal sets with a finite set of nondiverging traces; i.e. such that if f is a finite element then {s | s ∈ traces(f) and f↓s} is finite.

Proof:

1. It is not difficult to prove that CHAOS is the least element and that the relation $\supseteq$ is transitive, reflexive and antisymmetric. We are left to prove that every directed set has a least upper bound which belongs to $\mathbf{B}$. Let D be a directed set of bounded refusal sets; we can

define a new refusal set F by defining a set of traces and by associating to every trace a set of refusals as follows:

i) $Traces(F) = \cap\{ Traces(d) \mid d \in D\}$

ii) $s \in Traces(F)$ implies $Refusals(s, F) = \cap\{ Refusals(s, d) \mid d \in D\}$

Since Condition 6. implies that either the set of successors of any trace is finite or it is equal to A we have that:

(*) For all $s \in$ traces(F) there exists $d \in D$ such that Succ(s, F) = Succ(s, d).

Moreover since Theorem 4.3.7 and Proposition 4.3.11 imply that for any $s \in$ Traces(F), Refusals(s, F) can be uniquely determined by a finite set of cofinite sets of elementary actions we have that for any $s \in$ Traces(F)

(**) For all but a finite number of $d \in D$ Refusals(s, F) = Refusals(s, d).

Conditions (*) and (**) are sufficient to conclude that the refusal set F, determined by i) and ii) is indeed an element of $\mathbf{B}$ and is also the l.u.b. of D.

2. Let FIN denote the largest subset of $\mathbf{B}$ such that $f \in$ FIN implies {s | s ∈ Traces(f) and f↓s} is finite; let D be a directed set of bounded refusal sets. To prove that $\mathbf{B}$ is algebraic we will prove that $f \in$ FIN and $f \supseteq \cap D$ implies there exists $d \in D$ such that $f \supseteq d$ (i.e. that FIN is a set of finite elements) and moreover that any $F \in \mathbf{B}$ can be expressed as the limit of a directed subset of FIN (i.e. that FIN is the base of $\mathbf{B}$).

2a. Because of the representation theorem, to prove the claim it is sufficient to show that there exists $d \in D$ such that for every $s \in$ Traces(f) we have $Succ(s, f) \supseteq Succ(s, d)$ and $\mathbf{R}(s, d) \sqsubset \mathbf{R}(s, f)$. From (*) and (**) in the first part of the proof, we have that $F = \cap D$ and $s \in$ Traces(F) implies there exists $d \in D$ such that Succ(s, F) = Succ(s, d) and Refusals(s, F) = Refusals(s, d). Because of this we have that $f \supseteq F$ implies that for every $s \in$ Traces(f), such that f↓s, there exists $d_s \in D$ such that $Succ(s, f) \supseteq Succ(s, d_s)$ and $\mathbf{R}(s, d_s) \sqsubset \mathbf{R}(s, f)$. Since the set of sequences s, such that f converges on s (f↓s), is finite and D is directed we have $e = \cap \{d_s \mid s \in Traces(f), f\downarrow s\}$ implies $e \in D$. The fact that for every $s \in$ traces(f), f↓s implies $Succ(s, f) = A \supseteq Succ(s, e)$ and $\mathbf{R}(s, e) \sqsubset \mathbf{R}(s, f) = \{A\}$ suffice to conclude that $f \supseteq e$

and to prove the claim.

2b. We are left to prove that FIN is a base for $\mathbf{B}$, i.e. that for every $F \in \mathbf{B}$ there exists $D \subseteq FIN$ such that F is the least upper bound of D. Given any $F \in \mathbf{B}$ we can define a chain of bounded refusal sets as follows:

Let $F^n = \{ \langle s, X\rangle \mid \langle s, X\rangle \in F \text{ and } lenght(s) < n\} \cup$

$\{ \langle st, X\rangle \mid \langle s, \{\}\rangle \in F, lenght(s) = n, t \in A^* \text{ and } X \in FPOW(A)\}$.

It is easy to check that $F^n \in FIN$ for every $n \in \mathbf{N}$ and that F is the least upper bound of the chain $F^0 \supseteq F^1 \supseteq F^2 \supseteq \ldots\ldots F^n \supseteq \ldots\ldots$ □

As mentioned in section 4.1, in /BHR84/ it is proved that also $(F, \supseteq)$ is a complete partial order. However there nothing is said about its algebraicity. It can be easily shown that, in case A is infinite, no finite element of $(F, \supseteq)$ is denotable by a TCSP term. The main reason for this is that, since CHAOS is the bottom element, all the isolated elements have an infinite (cofinite) number of initial moves whilst TCSP allows only to describe terms which either can perform all the actions from A as initial moves or only a finite subset of them. The result about cofiniteness of initial moves of isolated elements is mainly due to the fact that there exists an infinite chain of refusal sets with an infinite number of initial moves between the bottom element CHAOS and every refusal set with a finite number of initials. The formal proof of this result is not reported here since it is not important to the economy of our work. However, this fact is very important since it suggests that in order to find the right semantic domain for TCSP it is worthwile to try to restrict the general domain of refusal sets.

While the above mentioned feature of F stresses the differences between Bounded Refusal Sets and general Refusal Sets, the next theorem emphasizes their similarities. The theorem will be useful for deriving results about $\mathbf{B}$ from known results about F.

Theorem 4.3.11 $(\mathbf{B}, \supseteq)$ is a sub-cpo of $(F, \supseteq)$, i.e. the elements of $\mathbf{B}$ are a subset of the elements of F, the orderings are compatible and the limits in F of directed subsets of $\mathbf{B}$ are preserved by $(\mathbf{B}, \supseteq)$.

Proof: The claim follows since $\mathbf{B}$ is obtained by imposing an additional constraint on elements of F and the orderings on $\mathbf{B}$ and F are the same. □

4.4. TCSP and Bounded Refusal Sets

We are now ready to use $\mathbf{B}$ as semantic domain for TCSP. We consider an extended version of the language discussed in the first part of this chapter. The new version includes the interleaving operator and recursively defined processes.

The language is essentially a set of recursively defined terms over a set of operators and may be defined using the same general schema of section 3.1, with $\Sigma_{TCSP} = \Sigma_{FTCSP} \cup \{\|\}$. The set of extended TCSP processes ranged over by T, U, V ... and denoted in the rest of this chapter by REC_Σ is then defined by the following BNF-like schema:

$T ::= x \mid Stop \mid CHAOS \mid a \rightarrow T \mid T_1 \sqcap T_2 \mid T_1 \Box T_2 \mid T/a \mid T_1 \|_I T_2 \mid T_1 \| T_2 \mid rec\ x.\ T.$

We have the usual notions connected with this syntax. In the sequel $T_1 \Box T_2 \Box \ldots\ldots \Box T_n$ and $T_1 \sqcap T_2 \sqcap \ldots\ldots \sqcap T_n$ will be abbreviated as in section 4.1, and the precedence of the operators will be the following: $/a > a\rightarrow > \|_I > \| > \sqcap > \Box$.

We can use the general techniques discussed in Section 1 of the previous chapter to give a denotational semantics to this version of TCSP. In the previous section we have proved that $\mathbf{B}$ is indeed a complete partial order, in this section we will define the continuous operations on bounded refusals sets we need. We will use the continuous operations on F defined in section 4.1 to derive the operations on $\mathbf{B}$. This will enable us to borrow results from /BHR84/ to prove the continuity of the new operators. First, we need to define $\|_F$, since its definition was not given in Section 4.1, where we considered the version of the language which did not contain

the operator $\|$. The wanted definition can be easily derived from the corresponding one of /BHR84/.

Let *merge* denote the following function from pair of traces to traces:

$merge(\varepsilon, \varepsilon) = \varepsilon$

$merge(as, bt) = a\,merge(s, bt) \cup b\,merge(as, t)$ then

$F_1 \|_F F_2 = \{\langle s, V\rangle \mid s = merge(s_1, s_2), \langle s_1, V\rangle \in F_1 \text{ and } \langle s_2, V\rangle \in F_2\}$

We can now define the operations on **B** corresponding to the various operators of the language for a theory of CSP. We will let $CHAOS_S$ denote $\{\langle st, V\rangle \mid t \in A^*, V \in FPOW(A)\}$ and Σ_D denote $\{Stop_D, CHAOS_D, a\rightarrow_D, \Box_D, \sqcap_D, /_D a, \|_D, \|_D\}$ with **D** equal consistently to **F** or **B**.

Definition 4.4.1

Let $STRICT(op_F)(x_1, \ldots x_k) = op_F(x_1, \ldots x_k) \cup \{CHAOS_S \mid s \in A^*, x_i\uparrow s \text{ for } 1 \le i \le k\}$ whenever $x_i \in \mathbf{B}$ for every i, $1 \le i \le k$,

then for every $op_F \in \Sigma_F$ we can define a corresponding op_B as follows:

$op_B(x_1, \ldots x_k) = STRICT(op_F)(x_1, \ldots x_k)$. □

We need to prove that the various op_B's are continuous functions over **B**. Since **B** is a subdomain of **F** it is sufficient to prove that the various op_B's are continuous functions over **F** and that if all their arguments are elements of **B** so is their result. To do this we will prove that the two members of the union in the definition of op_B are continuous functions.

Lemma 4.4.2 Let f be a function from F^K to $Pow(A^* \times POW(A))$ defined as follows: $f(x_1,\ldots,x_k) = \cup\{CHAOS_S \mid s \in A^*, x_i\uparrow s \text{ for } 1 \le i \le k\}$ then f is continuous w.r.t. reverse inclusion.

Proof: It is sufficient to prove that f is continuous with respect to the single x_i's, i.e. that $g(x) = \cup\{CHAOS_S \mid s \in A^*, l_j\uparrow s \text{ for } j \ne i \text{ and } 1 \le i \le k\} \cup \cup\{CHAOS_S \mid s \in A^*, x\uparrow s\}$ for a fixed i and for fixed l_j. Since the first member of the central union is a constant and

moreover the union of two sets is a continuous w.r.t. the chosen ordering, we need only to prove that $h(x) = \cup\{CHAOS_S \mid x\uparrow s\}$ is continuous.

We have to prove $\sqcap_h \cup\{CHAOS_S \mid x_n\uparrow s\} = \cup\{CHAOS_S \mid (\sqcap_h x_n)\uparrow s\}$. We need first of all to prove $\sqcap_h \cup\{CHAOS_S \mid x_n\uparrow s\} = \cup\sqcap_h\{CHAOS_S \mid x_n\uparrow s\}$

We will prove that lhs $\supseteq$ rhs and viceversa. The first inclusion follows purely by set theoretical reasonings. To prove that rhs $\supseteq$ lhs, suppose that the inclusion does not hold, i.e. that there exists $\langle t, V\rangle$ such that:

a. $\langle t, V\rangle \in \sqcap_h \cup\{CHAOS_S \mid x_n\uparrow s\}$ and

b. $\langle t, V\rangle \notin \cup\sqcap_h\{CHAOS_S \mid x_n\uparrow s\}$.

We have that a. implies that for all $n \ge 0$ there exists a trace s such that $t = su$, for some string u, and $x_n\uparrow s$ and so we have that for all $n \ge 0$, $x_n\uparrow t$ by condition 6. On the other hand b. implies that for all s such that $t = su$, for some string u, there exists $n \ge 0$ such that $x_n\downarrow s$ and we have that there exists $n \ge 0$ such that $x_n\downarrow t$. In conclusion we have that a. implies $\forall n.\ x_n\uparrow t$ while b. implies $\exists n.\ x_n\downarrow t\rangle$ which is a contradiction.

We are left to prove $\cup\sqcap_h\{CHAOS_S \mid x_n\uparrow s\} = \cup\{CHAOS_S \mid (\sqcap_h x_n)\uparrow s\}$, this follows directly from the fact that $(\sqcap_h x_n)\uparrow s$ if and only if for all $n \ge 0$ $x_n\uparrow s$. □

Lemma 4.4.3 If $x_i \in \mathbf{B}$ for $1 \le i \le k$ then $op_B(x_1, \ldots x_k) \in \mathbf{B}$ for every $op_B \in \Sigma_B$.

Proof: It is very easy to verify, by structural induction and by induction on the length of s, that $op_F(x_1, \ldots x_k)$ has a trace s with an infinite number of successors if and only if $x_i\uparrow s$ for some $1 \le i \le k$. This and the fact that $x_i\uparrow s$ implies $op_B \supseteq \{\langle st, V\rangle \mid t \in A^*, V \in FPOW(A)\}$ are sufficient to prove the claim. □

Theorem 4.4.4 Let op_B be any operation from Σ_B, then op_B is continuous over **B**.

Proof From /BHR84/ we know that all $op_F \in \Sigma_F$ are continuous over **F**; moreover we have that $\cup$ is continuous and that $op_B(x_1, \ldots x_k) \in \mathbf{F}$. This together with Lemma 4.4.2 implies that every op_B is continuous over **F**, Lemma 4.4.3 and Theorem 4.3.13 imply that every op_B is continuous over **B** as well. □

Let ENV denote a function from X to $\mathbf{B}$, and e range over ENV then we may define $\mathfrak{B}: REC_\Sigma \to (ENV \to \mathbf{B})$ as follows:

$\mathfrak{B}[\![x]\!]e = e(x)$

$\mathfrak{B}[\![Stop]\!]e = \{\langle \varepsilon, V\rangle \mid V \in FPOW(A)\}$

$\mathfrak{B}[\![CHAOS]\!]e = \{\langle s, V\rangle \mid s \in A^* \text{ and } V \in FPOW(A)\}$

$\mathfrak{B}[\![a \to T]\!]e = \{\langle \varepsilon, V\rangle \mid V \in FPOW(A-\{a\})\} \cup \{\langle as, W\rangle \mid \langle s, W\rangle \in \mathfrak{B}[\![T]\!]e\}$

$\mathfrak{B}[\![T \,\square\, U]\!]e = \{\langle \varepsilon, V\rangle \mid \langle \varepsilon, V\rangle \in \mathfrak{B}[\![T]\!]e \cap \mathfrak{B}[\![U]\!]e\} \cup$
$\{\langle s, W\rangle \mid s \in A^+ \text{ and } \langle s, W\rangle \in \mathfrak{B}[\![T]\!]e \cup \mathfrak{B}[\![U]\!]e\} \cup$
$\{\langle st, V\rangle \mid t \in A^*, X \in FPOW(A) \text{ and } (\mathfrak{B}[\![T]\!]e)/s \text{ or } (\mathfrak{B}[\![U]\!]e)/s\}$

$\mathfrak{B}[\![T \sqcap U]\!]e = \mathfrak{B}[\![T]\!]e \cup \mathfrak{B}[\![U]\!]e$

$\mathfrak{B}[\![T \,\mathbf{|}\, U]\!]e = \{\langle s, V \cup W\rangle \mid \langle s, V\rangle \in \mathfrak{B}[\![T]\!]e \text{ and } \langle s, W\rangle \in \mathfrak{B}[\![U]\!]e\} \cup$
$\{\langle st, V\rangle \mid t \in A^*, X \in FPOW(A) \text{ and } (\mathfrak{B}[\![T]\!]e)/s \text{ or } (\mathfrak{B}[\![U]\!]e)/s\}$

$\mathfrak{B}[\![T/a]\!]e = \{\langle s/a, V\rangle \mid \langle s, V \cup \{a\}\rangle \in \mathfrak{B}[\![T]\!]e\} \cup$
$\{\langle st, V\rangle \mid t \in A^*, X \in FPOW(A) \text{ and } (\mathfrak{B}[\![T]\!]e)/s \text{ or } \forall n \geq 0\ \langle sa^n, \{\}\rangle \in \mathfrak{B}[\![T]\!]e\}$

$\mathfrak{B}[\![T \,\|\, U]\!]e = \{\langle s, V\rangle \mid s = merge(s_1, s_2), \langle s_1, V\rangle \in \mathfrak{B}[\![T]\!]e \text{ and } \langle s_2, V\rangle \in \mathfrak{B}[\![U]\!]e\} \cup$
$\{\langle st, V\rangle \mid t \in A^*, X \in FPOW(A) \text{ and } (\mathbf{B}[\![T]\!]e)/s \text{ or } (\mathbf{B}[\![U]\!]e)/s\}$

$\mathfrak{B}[\![rec\ x.\ T]\!]e = \mathbf{fix}\,\lambda d.\ \mathfrak{B}[\![T]\!]e[d/x]$

Table 4.3: The new Semantic Function for TCSP

Part 1. of theorem 4.3.12 and the last theorem guarantee that $\mathbf{B}$ together with $\Sigma_{\mathbf{B}}$ is a Σ-cpo and allow us to use it as a semantic domain for the language for a theory of CSP, by using the general framework of section 3.1. For the sake of clarity all the semantic equations for the extended version of TCSP are presented in Table 4.3. . The new semantic equations are very similar to those derivable in section 4.1 and explicitely defined in /BHR84/ and /Bro83b/; the only difference arises from the different way divergence is handled.

4.5. A Complete Proof System for TCSP

The denotational semantics of table 4.3 induces a new preorder on REC_Σ:

$T_1 \mathbin{\mathfrak{C}} T_2$ if and only if for all $e \in ENV$ $\mathfrak{B}[\![T_1]\!]e \supseteq \mathfrak{B}[\![T_2]\!]e$

As for $\mathfrak{C}$, defined in section 4.1, and for the same reasons we have:

Proposition 4.5.1 $\mathfrak{C}$ is preserved by all the operators in Σ. □

Similarly to $\mathfrak{C}$, the new precongruence $\mathfrak{C}$, can be characterized in terms of a set of equations over Σ. In fact, the set of axioms $\mathbf{A}^*$ of Table 4.4 characterize completely $\mathfrak{C}$, when restricted to closed finite terms ($CFREC_\Sigma$). Since the operations on $\mathbf{B}$ and $\mathbf{F}$ coincide when restricted to fully specified refusals sets, we have that the two axioms systems $\mathbf{A}$ and $\mathbf{A}^*$ are very similar. In fact, apart from the axioms for $\mathbf{|}$ (INT1-INT4), which were not given in $\mathbf{A}$, $\mathbf{A}^*$ has only two additional axioms, E$\emptyset$ and PC$\emptyset$, which enforce stricteness of $\square$ and $\mathbf{|}$ and offer a solution to the problems discussed at the end of section 4.3. The axioms which were not in Table 4.1 (**A**) will be in boldface. Please note that the axioms of $\mathbf{A}$, which do not appear in Table 4.4 (PC2, PC3), are easily derivable in $\mathbf{A}^*$ by using E$\emptyset$ and PC$\emptyset$.

Undefined

C1 $CHAOS \sqsubseteq X$

External Choice

E1 $X \,\square\, X = X$

E2 $X \,\square\, Y = Y \,\square\, X$

E3 $X \,\square\, (Y \,\square\, Z) = (X \,\square\, Y) \,\square\, Z$

E4 $X \,\square\, Stop = X$

E5 $X \,\square\, CHAOS = CHAOS$

Internal Choice

I1 $X \sqcap X = X$

I2 $X \sqcap Y = Y \sqcap X$

I3 $X \sqcap (Y \sqcap Z) = (X \sqcap Y) \sqcap Z$

I4 $X \sqcap Y \sqsubseteq X$

Distributive Laws

D1 $X \sqcap (Y \,\square\, Z) = (X \sqcap Y) \,\square\, (X \sqcap Z)$

D2 $X \,\square\, (Y \sqcap Z) = (X \,\square\, Y) \sqcap (X \,\square\, Z)$

D3 $(a \rightarrow X) \sqcap (a \rightarrow Y) = a \rightarrow (X \sqcap Y)$

D4 $(a \rightarrow X) \,\square\, (a \rightarrow Y) = a \rightarrow (X \sqcap Y)$

Hiding

H1 $Stop/a = Stop$

H2 $CHAOS/a = CHAOS$

H3 $(X \sqcap Y)/a = X/a \sqcap Y/a$

H4 $(a \rightarrow X \,\square\, Y)/a = X/a \sqcap (X \,\square\, Y)/a$

H5 $\square \{b_i \rightarrow X_i \mid i \in I\}/a = \square \{b_i \rightarrow X_i/a \mid i \in I\}$ if $\forall i \in I \; b_i \neq a$

If $P = \square \{ a_i \rightarrow P_i \mid i \in I \}$ and $Q = \square \{ b_j \rightarrow Q_j \mid j \in J \}$ then:

Parallel Composition

PCØ $P \,\|\, CHAOS = CHAOS$

PC1 $P \,\|\, Q = \square \{a_k \rightarrow (P_k \,\|\, Q_k) \mid a_k = b_h \text{ and } k \in I, h \in J\}$

PC4 $X \,\|\, Y = Y \,\|\, X$

PC5 $(X \sqcap Y) \,\|\, Z = (X \,\|\, Z) \sqcap (Y \,\|\, Z)$

Interleaving

INT1 $P \,\|\, Q = \square \{a_i \rightarrow (P_i \,\|\, Q) \mid i \in I\} \,\square\, \square \{ b_j \rightarrow (P \,\|\, Q_j) \mid j \in J\}$

INT2 $X \,\|\, CHAOS = CHAOS$

INT3 $X \,\|\, Y = Y \,\|\, X$

INT4 $(X \sqcap Y) \,\|\, Z = (X \,\|\, Z) \sqcap (Y \,\|\, Z)$

Table 4.4: The New Axiom System

The rest of the section is dedicated to proving that **A'** is sound and complete with respect to $\lesssim$. All the proofs rely on the ones given in section 4.2.2 for **A** and $\sqsubseteq$ and follow the same patterns; most of them will be only outlined. In this case we have that the proofs are simpler than the ones for **A** since there is no anomaly for terms with CHAOS as a subterm. Moreover the normal form we obtain are much closer to CCS strong normal form of Chapter 2; this allows us to extend the functions rt and tr of section 4.2.2 to divergent normal forms. In fact the similarity of the roles played by CHAOS and Ω allows to consider also terms with underpecified subterms and to define:

iv. $tr(\Omega) = CHAOS$ and

iv. $rt(CHAOS) = \Omega$

Proposition 4.5.2 The set of axiom **A'** is sound for $\lesssim$.

Proof: From definition 4.4.1 we have that op_F and op_B coincide in case for every i, $1 \leq i \leq k$, and for every s, $s \in Traces(x_i)$, we have $x_i \downarrow s$. Moreover we have that all equations of Table 4.1 are such that their left hand side diverges on a particular trace s if and only if their right hand side diverges on s as well. These two facts are sufficient to conclude that all the axioms for $\sqsubseteq$ are valid axioms for $\lesssim$ too. We are left only to prove that EØ, PCØ and INT1-INT4 are sound. For all of them it is a matter of simple calculations to prove that the refusal set of the left hand side and the one of the right hand side coincide. □

The proof of completeness relies again on the existence of normal forms.

Definition 4.5.3 If P is in $FREC_\Sigma$ then P is in **strict normal form** if it is of the form:

$\sqcap \{ \,\square \{a \rightarrow P(a) \mid a \in L\} \mid L \in \mathbf{L} \,\} [\square CHAOS]$ where

- **L** is a nonempty saturated set;
- P(a) are in normal form;
- $[\square CHAOS]$ denotes that CHAOS is an optional summand, and if CHAOS is a summand then P = CHAOS. □

It should be noted that in this case the structure of the strict normal forms is simpler than the one of the normal forms of Section 4.4. It is possible to prove that every process can be reduced to an equivalent one which is in strict normal form by using axioms from **A'**.

Proposition 4.5.4 For every $P \in CFREC_\Sigma$ there exist a strict normal form *snf* (P) such that $P = snf(P)$.

Proof: Proposition 4.2.8 guarantees that P can be reduced to a normal form. It is not difficult to see that by suitable applications of EØ every normal form can be reduced to a strict normal form. □

Next we prove that $\sqsubseteq$ and $\lesssim$ coincide when restricted to strict normal forms.

Proposition 4.5.5 If P and Q are in strict normal form then $P \sqsubseteq Q$ iff $P \lesssim Q$.

Proof: It is easy to show, by induction on the structure of processes, that if P is in strict normal form then $\mathcal{B}[\![P]\!]e = \mathcal{F}[\![P]\!]e$. This suffices to prove the claim. □

Theorem 4.5.6 For every $P, Q \in CFREC_\Sigma$ $P \lesssim Q$ implies $\mathbf{A'} \vdash P \sqsubseteq Q$.

Proof: The claim follows directly from the existence of strict normal forms provably equal to P and Q (proposition 4.5.4), from Proposition 4.5.5, from the completeness theorem for **A** and $\sqsubseteq$ (Theorem 4.2.16), and from the fact that all axioms in **A** hold in **A'**. □

Theorem 4.5.7 $\lesssim$ is the least precongruence on $CFREC_\Sigma$ generated by the set of axioms **A'**

Proof: See the proof of Theorem 4.2.17. □

The latter theorem gives a complete characterization of $\lesssim$ when restricted to finite closed terms. As we have seen in Section 2.6 there are relatively standard methods for extending this

characterization to arbitrary terms. They rely on the possibility of determining the behaviour of infinite terms as the limit of the behaviour of all the syntactically finite terms which approximate them. In our case the generalization is possible because the semantic domain and the precongruence considered enjoy some natural inductive properties, in particular both of them are algebraic.

The rest of the chapter is dedicated at generalizing the axiom system **A'** to infinite terms; the approach followed is the one used in Chapter 2 and discussed in details in Chapter 3.

First we define the "syntactically finite" approximants for terms of REC_Σ. Their definition is based on the same syntactical ordering defined in Sections 2.4 and 3.1 with CHAOS playing the role of Ω in Ω1.

Let $<$ be the least Σ-precongruence over REC_Σ generated by the equations:

C1 $\quad CHAOS < X$

REC1 $\quad T[rec\ x.\ T/x] < rec\ x.\ T$

and let

$FIN(T) = \{ t \mid t < T,\ t \in FREC_\Sigma \}$.

We can prove that there is a close relationship between the finite elements of the semantic domain **B** and the set of finite terms $FREC_\Sigma$. In fact as we did in Chapter 3 (Definition 3.3.9 and following), we can show that any isolated element of the semantic domain (bounded refusal sets) is denotable by a, syntactically finite, strict normal form.

To prove this we will use the alternative definition of refusal sets given in Theorem 4.3.6, which characterizes refusal sets in terms of sets of traces and sets of maximals associated to every trace. To do this, first, we need to study the relationships between sets of maximals and saturated sets, since the latter are used in the definition of TCSP normal forms.

In the following, if A is a set and **L** is a set of sets we let $A + \mathbf{L}$ denote $\{A - L \mid L \in \mathbf{L}\}$.

Theorem 4.5.8 If $\mathbf{L}$ is a (finite) saturated set and $\mathbf{M}$ is a (cofinite) set of maximals then we have:

i. $Sat(A\div Maximals(A\div \mathbf{L}) \cup \{A(\mathbf{L})\}) = \mathbf{L}$

ii. $Maximals(A\div Sat(A\div \mathbf{M})) = \mathbf{M}$.

Proof: Let SAT and MAX the left hand sides of i. and ii. respectively.

i. We prove **a.** $SAT \supseteq \mathbf{L}$ and **b.** $\mathbf{L} \supseteq SAT$.

a. If $X \in \mathbf{L}$ then we have that either $X \in A\div Maximal(A\div\mathbf{L})$ or $X \notin A\div Maximal(A\div\mathbf{L})$. In case the former holds then the claim follows trivially. In case $X \notin A\div Maximal(A\div\mathbf{L})$ then we have that there exists $M_1 \in Maximal(A\div\mathbf{L})$ such that $M_1 \supseteq A-X$, i.e. $A-M_1 \subseteq X$. Now since $X \in \mathbf{L}$ we have also $X \subseteq A(\mathbf{L})$ and therefore that $X \in SAT$ by definition of Sat.

b. Suppose $X \in SAT$ then we have $L \subseteq X \subseteq A(\mathbf{L})$, for some $L \in A\div Maximals(A\div \mathbf{L})$, i.e. for some $L \in \mathbf{L}$. Since $\mathbf{L}$ is saturated, we have also $X \in \mathbf{L}$.

ii. Also in this case we prove **a.** $Max \supseteq \mathbf{M}$ and **b.** $\mathbf{M} \supseteq MAX$.

a. The claim follows from $A\div Sat(A\div \mathbf{M}) \supseteq A\div(A\div \mathbf{M}) = \mathbf{M}$.

b. Suppose there exists $M \in MAX$, this implies there exists $L \in Sat(A\div \mathbf{M})$ such that $M = A-L$ and that there exists $N \in \mathbf{M}$ such that $A-N \subseteq A-M \subseteq A(A\div\mathbf{M})$. Now $A-N \subseteq A-M$ implies $N \supseteq M$ and $N \in \mathbf{M}$ implies $N \in MAX$ by case **a.**, moreover MAX is a set of maximals and so we have $N = M$ and the claim follows. □

Now as we did in Chapter 3 for AT_i's, we can establish the connections between strict normal forms and bounded refusal sets. The proofs follow the same pattern of the ones of Section 3.3 and most of them will be omitted. We start by defining two functions which map strict normal form to bounded refusals set and viceversa. For strict normal forms we will have the same ordering ($\lessdot$) we have defined for TCSP normal forms in Definition 4.2.10. However, it should be noted that in this case the ordering is simpler since clause 1. of Definition 4.2.9 ($<$) holds whenever clause 2. does, and is thus meaningless. The operations on TCSP strict normal forms we will use will be defined similarly to those of Definition

3.3.10, for CCS normal forms. Below we give two functions which allow to obtain normal forms from bounded refusal sets and bounded refusal sets from normal forms. In the second case we will get refusal sets expressed as sets of traces and sets of maximal with abuse of notation we will still denote their ordering by $\supseteq$.

Definition 4.5.9

1. If f is an isolated element of $\mathbf{B}$ and we let $\mathbf{After}(a, f) = \{ \langle s, V\rangle \mid \langle as, V\rangle \in f, a \in A\}$ and $\{\Box a\to P \mid a \in \{\}\}$ denote Stop, we can define a function Ψ: $FIN(\mathbf{B}) \to SNF$ by structural induction as follows:

If $f \neq \{\langle t, V\rangle \mid t \in A^*, V \in FPOW(A)\}$

$\Psi(f) = \{\prod\{\Box a\to\Psi(\mathbf{After}(a, f)) \mid a \in L\} \mid L \in Sat(A - \mathbf{R}(\varepsilon, f))\}$

If $f = \{\langle t, V\rangle \mid t \in A^*, V \in FPOW(A)\}$

$\Psi(f) = CHAOS$.

2. If n is a strict normal form we can define a function Φ: $SNF \to FIN(\mathbf{B})$ by structural induction as follows:

If $n = \{\prod\{\Box a\to n(a) \mid a \in L\} \mid L \in \mathbf{L}\}$ then

$traces(\Phi(n)) = \{\varepsilon\} \cup \{as \mid s \in traces(\Phi(n(a)))\}$;

$\mathbf{R}(\varepsilon, \Phi(n)) = Maximals(A \div \mathbf{L})$;

$\mathbf{R}(as, \Phi(n)) = \mathbf{R}(s, \Phi(n(a)))$.

If $n = CHAOS$ then

$traces(\Phi(n)) = \{s \mid s \in A^*\}$;

$\mathbf{R}(s, \Phi(n)) = A$ □

Lemma 4.5.10

a. For all $f \in \mathbf{B}$ we have $\Psi(f) \in SNF$;

b. For all $f_1, f_2 \in \mathbf{B}$, $f_1 \supseteq f_2$ implies $\Psi(f_1) \lessdot \Psi(f_2)$ □

Lemma 4.5.11

a. For all $n \in SNF$ we have $\Phi(n) \in FIN(\mathbf{B})$.

b. For all $n_1, n_2 \in SNF$, $n_1 \not\ll n_2$ implies $\Phi(n_1) \supseteq \Phi(n_2)$ □

Lemma 4.5.12 If n is a strict normal form then $\mathbf{B}[\![n]\!]$ is isomorphich to $\Phi(n)$.

Proof: The proof follows by structural induction on n and uses the Characterization Theorem for Refusal Sets (Theorem 4.3.6.). □

Theorem 4.5.13 If f is a finite element of $\mathbf{B}$ then there exists a closed term $t \in FREC_\Sigma$ such that $f = \mathbf{B}[\![t]\!]$.

Proof: We have that the wanted t is $\Psi(f)$. It can be proved by induction on the structure of the normal forms and the lenght of the sequences of actions of the elements of $FIN(\mathbf{B})$ that Φ and Ψ are inverses. This implies that $f = \Phi(\Psi(f))$ and Theorem 4.5.12 guarantees that $\Phi(\Psi(f))$ is equal, up to isomorphisms, to $\mathbf{B}[\![\Psi(f)]\!]$. □

The previous results are sufficient to show that $\overline{\sqsubseteq}$ is completely determined by its restriction to finite terms.

Proposition 4.5.15 $\overline{\sqsubseteq}$ is an algebraic relation.

Proof: Theorem 4.3.12 and theorem 4.4.4 imply that $\mathbf{B}$ is an algebraic Σ-cpo.The previous lemma and proposition 3.1.7 imply that $\overline{\sqsubseteq}$ is substitutive. Because of this, proposition 3.1.8 suffice to prove the claim. □

We are now ready to compare the generalizations of $\sqsubseteq$ and $\overline{\sqsubseteq}$ to infinite processes. Let $I_{\mathbf{A}^*}$ be the initial Σ -cpo in the category of Σ-cpos which satisfy the set of axioms $\mathbf{A}^*$. As we have seen in Section 3.1, $I_{\mathbf{A}^*}$ can be obtained by ordering the set of ideals of $(FREC_\Sigma, \sqsubseteq)$ by set inclusion and taking its chain completion.

Theorem 4.5.16 $I_{\mathbf{A}^*}$ is fully abstract with respect to $\overline{\sqsubseteq}$.

Proof let $<_{I(\mathbf{A}^*)}$ denote the precongruence induced by $\mathcal{M}_{I(\mathbf{A}^*)}$. By construction we have that $<_{I(\mathbf{A}^*)}$ satisfies the set of axioms $\mathbf{A}^*$ and is algebraic. Theorem 4.5.7 and Proposition 4.5.3 together with Proposition 4.5.15 are sufficient to prove the claim. □

The following corollary is a direct consequences of Theorem 4.5.16 and Proposition 3.1.9.

Corollary 4.5.17

For every $t_1, t_2 \in REC_\Sigma$ we have $t_1 \sqsubseteq t_2$ if and only if $t_1 \overline{\sqsubseteq} t_2$. □

The fact that the relation $\overline{\sqsubseteq}$ is an algebraic relation which is completely characterized by a set of axioms has other important consequences. It gives us a complete axiomatic proof system for TCSP. This proof system, as suggested in Section 3.1, will be $\mathbf{DED}(\mathbf{A}^*)$ and will consist of: The axioms from Table 4.4, the reflexivity, transitivity, substitutivity and instantiation rules, and the general induction rule:

$$\frac{\text{for all } d \in Fin(X),\ d \sqsubseteq Y}{X \sqsubseteq Y}$$

Theorem 4.5.16 gives an equational characterization of the congruence induced by the semantic function $\mathbf{B}$. We can also give an equational characterization of $\mathbf{B}$, the domain used to define $\mathbf{B}$. It is in fact possible to characterize the domain of bounded refusal sets, together with the operations $op_{\mathbf{B}}$'s of the previous section, as the initial cpo generated by a particular set of equations. The proof are similar to that given in Chapter 3, when studying the axiomatic characterizations of the various RT_i's. In the following we mention only the main differences.

Lemma 4.5.18

If t_1 and $t_2 \in FREC_\Sigma$ then we have $t_1 \sqsubseteq t_2$ if and only if $snf(t_1) \ll snf(t_2)$.

Proof: The proof follows from Lemma 4.2.15, Proposition 4.5.2 and Lemma 4.2.14. □

Proposition 4.5.19 (SNF, $\ll$) is a partial order and when equipped with the operators op_{snf} defined as follows $op_{snf}(n_1,\ldots n_k) = snf(op(n_1,\ldots n_k))$ for every $op \in \Sigma$, is isomorphic as a Σ-po to $(FIN(\mathbf{B}), \supseteq)$ equipped with the operators $op_{\mathbf{B}}$.

Proof: The isomorphism pair is given by Ψ and Φ defined above. They are proved monotonic and well defined in Lemmas 4.5.10 and 4.5.11. The fact that they are inverses follows from Theorem 4.5.8. The only thing we are left to prove is that both of them are indeed homomorphisms. This proof is long and tedious but basically follows the same pattern of the proof of Proposition 3.3.12. However, since in the case of bounded refusals sets the operations are defined directly, induction on terms and knowledge of the procedure for reducing TCSP terms to normal forms is needed. □

Lemma 4.5.20 (SNF, $\ll$) is a partial order and when equipped with the operators op_{snf} defined as above is isomorphic as a Σ-po to $(FREC_{\Sigma}/\sqsubseteq, \sqsubseteq)$ equipped with the operators $op \in \Sigma$. □

Theorem 4.5.21 $\mathbf{B}$ is isomorphic as a Σ-cpo to $I_{\mathbf{A}^*}$, the initial cpo which satisfies all the equations in $\mathbf{A}^*$. □

5. MODELS AND OPERATORS FOR NONDETERMINISTIC PROCESSES

5.0 Introduction

In the last two chapters, we presented and discussed two groups of semantic domains for communicating processes: Representation Trees (RT) and Refusal Sets. These two groups arose from two apparently opposite approaches. Representation Trees stemmed out of an attempt to give a denotational semantics to CCS which would be fully abstract with respect to the two level operational semantics based on a simple transition relation and on testing equivalences, /DH84/. Refusal Sets were postulated in /BHR84/ to be the right domain to "play the same role, in defining the semantics of communicating processes, as the domain of partial functions does for sequential programming languages".

In spite of their different origins, these two groups of semantic domains are very similar. Both associate every process with sets of pairs, $\langle s, X\rangle$, which carry information about the sequences of actions which the process can perform and about the possible future of the process after every sequence. In the case of Refusal Sets, the future is expressed in terms of the sets of experiments the process would refuse, whereas in the case of Representation Trees, it is expressed in terms of the experiments the process would certainly accept. However, the two sets used to represent the future can be proved complementary with respect to the set of all the elementary actions which processes may perform. The main difference is that representation trees in RT can have unlabelled roots while every refusal set F is such that every trace of F has an associated refusal. The reason of this discrepancy lays in the different emphasis put on modelling internal behaviour of systems. It is not very difficult to prove that

an injective function exists which maps every representation tree into a refusal set, see /DeN84/. Unfortunately, both Representation Trees and Refusal Sets have their drawbacks: There is no immediate extension of Refusal Sets which allows them to treat internal moves properly and the dishomogeneity allowed for the labels of the roots of Representation Trees introduces some inelegance in the definitions of their operations.

In this chapter, we present yet another domain for communicating processes. This is very similar to the RT model but avoids the problems caused by unlabelled roots. The trees in this new model, instead of having labelled and unlabelled roots, have two sets of sets associated with their root: an **acceptance set** and a **preemptive set**. The preemptive set is used to describe the possibility a process has of performing initial silent moves, while the acceptance set is used to describe the subsets of initial visible actions which a process may decide to offer to the external environment, by performing internal moves. We will call the new model **Preemptive Representation Trees (PRT)**. The minor changes to Representation Trees will allow us to define their operators straightforwardly and to overcome the difficulties encountered in /DeN84/ and discussed in Chapter 3.

The operators we will define will be based on some of the operators defined in Chapters 2 and 4 for CCS and TCSP, respectively. We will not consider all the operators but only the "primitive" operators of the two languages; the operators which can be expressed in terms of the primitive ones will be ignored. For example we will not define any parallel combinator, because the languages we are interested in reduce parallel compositions of processes to nondeterministic interleaving of their actions. In fact, the sets of axioms discussed in the previous chapters show that the various parallel combinator ($|$, $\|$ and $\|_A$), the restriction ($/$), the hiding ($\backslash$) and the relabelling ($[S]$) operators can all be expressed in terms of the various nondeterministic operators ($+$, $\Box$ and Π), of the prefixing operators (a, τ and $a\rightarrow$) and of the basic processes (NIL, Ω, Stop and CHAOS).

An extension of Representation Trees very similar to our PRT is described in /Hen84/. There the model is called Asynchronous Strong Nondeterministic Trees (**ASNT**) and our preemptive set is named withdraw set. However, since the author is interested in modelling a different language (the variant of CCS described in /Hen83a/), the set of operations defined on ASNT is different from ours.

In the following, first we define formally the domain of Preemptive Representation Trees and their operations, then relate PRT to the other domains for communicating processes discussed in the previous chapters. In particular, we show that Strong Representation Trees (denoted by SRT and RT_2 in Chapter 2) and Bounded Refusal Sets (denoted by **B in Chapter 4**) are isomorphic to two different reducts (see /BG82/ or /Coh81/), of PRT. The significance of this result is twofold, it allows us, firstly, to understand the relationships between Representation Trees and Refusal Sets and, secondly, to compare two significative subsets of TCSP and CCS and to discuss the relationships between the nondeterministic operators of these two important theories of communicating processes.

Previous attempts at a direct comparison of CCS and TCSP have been hindered both by the different sets of operators of the two languages and by the different methods used to define their semantics. In /Bro83a/, synchronization trees are used as the semantic domain for both languages: The comparison is hampered by the different equivalences used to quotient the trees. In /MZ83/ the normal forms of /DH84/ and /DeN83/ (also presented in Chapters 2 and 4) are used to define translations from one language to the other and to compare their expressive power: No additional insight into the interrelationships between the various operators is gained.

The rest of this chapter is organized as follows. In Section 5.1, we present the new model together with some examples. In Section 5.2, we present and discuss the various operators. In the final section we relate the new model to Representation Trees and Refusal Sets and use the established relationships to discuss and compare CCS and TCSP operators.

5.1 Preemptive Representation Trees

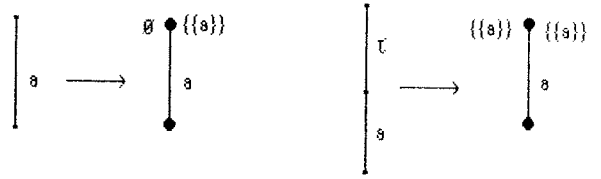
In this section, we present yet another domain for communicating processes, which we will call Preemptive Representation Trees. Once again, the elements of this domain will be particular kinds of trees. In fact, they are very similar to the representation trees discussed in Section 3.3; the only difference is that the roots of each tree, similarly to all the other nodes, are labelled by an **acceptance set**, and that associated to each root there is also a saturated subset of its acceptance set, which we will call **preemptive set**. This new set will allow us to denote the set of states to which a process can silently move, thus preventing some of its initial actions to take place. While in Chapter 3 we considered three different classes of representation trees, each one corresponding to a different way of handling divergence, in this chapter, for the sake of simplicity, we will only discuss the class of Preemptive Representation Trees which corresponds to the choice of considering divergence as catastrophic. The domain we obtain is closely related to Strong Representation Trees (RT_2). The same notation as in Chapter 3 will be used. The preemptive set of a tree t will be denoted by $\mathcal{P}(t)$ and sometimes we will render $\mathcal{A}(t, \varepsilon)$ as $\mathcal{A}(t)$. Moreover, given a set of sets $\mathcal{S}$ we will use $\mathbf{Sat}(\mathcal{S})$ to denote the least saturated set which contains $\mathcal{S}$ and $A \dotplus \mathcal{S}$ to denote $\{A - S \mid S \in \mathcal{S}\}$.

Definition 5.1.1 Let L be a set of labels, then PRT_L is the set of rooted, finitely branching trees, such that $t \in PRT$ implies:

1. Every branch is labelled by an element, a of L, and for every node there is at most one outgoing branch labelled by a;
2. Every node is either open or closed;
3. If a node is open, then it is a leaf;
4. Every closed node, s, is labelled by $\mathcal{A}(t, s)$, a saturated set of subsets of Succ(t, s) which contains Succ(t, s).
5. The root is also labelled by $\mathcal{P}(t)$, a saturated subset of $\mathcal{A}(t)$. □

A few examples will help to evidence the differences between PRT and SRT. (Once again, the index L will be omitted whenever it is clear from the context). As in Chapter 3, we can show how trees from PRT's can be used to give a meaning to synchronization trees. We have:

Examples



These examples show that we associate the representation tree whose preemptive set is empty to the deterministic synchronization tree which can only perform an a-move and the representation tree whose preemptive set is equal to {{a}} to the synchronization tree which can perform a silent (uncontrollable) move before performing the a-move. Apart from this, the preemptive representation tree associated to any synchronization trees will be equal to the representation tree associated to it in Section 3.1. Indeed, we have that the trees from PRT, associated to all the other synchronization trees of Table 3.1 will be equal to the corresponding representation trees in the table, apart from the fact that each tree in PRT has its root labelled by the acceptance set {{a}}. Other examples of elements of PRT will be given in the next section when we define operations on them.

We can define an ordering on PRT, in much the same way as that on RT.

Definition 5.1.2 If $t, t' \in PRT$ then $t < t'$ if for every $s \in Nodes(t')$ we have

$s \in CNodes(t)$ implies $s \in CNodes(t')$;

and $\mathcal{A}(t', s) \subseteq \mathcal{A}(t, s)$;

and $\mathcal{P}(t') \subseteq \mathcal{P}(t)$. □

Intuitively, this ordering suggests that we can "improve" (render more deterministic) a tree either by grafting a new tree into an open leaf, or by removing elements from its acceptance sets or from its preemptive set.

Theorem 5.1.3 (PRT,<) is an algebraic cpo whose finite elements are the trees from PRT with a finite number of nodes.

Proof: This proof is similar to that of Theorem 3.3.6. When defining the limit of directed subsets of PRT, we will also have $\mathcal{P}(t) = \cap \{ \mathcal{P}(d) \mid d \in D, \varepsilon \in CNodes(d)\}$. □

5.2 Operators for Nondeterminism

In this section, we define a class of continuous operators over PRT. The class introduced will not be a minimal one. The various operations have been chosen mainly to allow an easy expression of fundamental concepts of the languages to which we want to give a meaning. In the following we will first give an informal description of the operational significance of the operators and then define them formally. We will define two zeradic operators (Θ and Ω), two unary operators ($a\rightarrow$ and $\tau\rightarrow$) and three binary operators ($\sqcap$, $\Box$ and +).

The constant Θ will denote the trivial tree •, consisting only of a single closed node, while the constant Ω will denote the trivial tree, o, consisting of a single open node. The operator "$a\rightarrow$" prefixes a process with the elementary action a, while "$\tau\rightarrow$" gives a tree the possibility of changing its internal state silently without the external environment having any control over it. The three binary operators are such that if P and Q are processes then: a. $P \sqcap Q$, b. $P \Box Q$ and c. P + Q are processes which behave like P or like Q. The difference between them consists in the amount of control over the choice between the two processes which is left to the external environment. In case a. the choice is wholly nondeterministic, the environment does

not have any control over it. In case b. the choice will be completely determined by the first communication which the environment offers to the process. In case c. the choice will again be determined by the first communication offered by the environment, unless one of the summands has the possibiliy of changing state silently. In this case P + Q will nondeterministically decide whether to leave the choice to the environment or to perform the silent move.

The binary operators and the operator "$\tau\rightarrow$" allow us to describe the nondeterministic behaviour of processes; the analysis and comparison of operators for nondeterminism are among the principal aims of this chapter. As mentioned above, we will not be considering parallel operators. We will let Σ denote the set of operators over PRT, i.e.

$\Sigma = \{ \Theta, \Omega, a\rightarrow (\text{for all } a \in L), \tau\rightarrow, \sqcap, \Box, + \}$.

The precedence of the various operators in Σ is the following: $a\rightarrow > \tau\rightarrow > \sqcap > \Box > +$.

We now give a formal definition of the various operators in Σ. In order to define any operation on PRT it is sufficient to describe four endomorphisms on Nodes, CNodes, $\mathcal{A}$ and $\mathcal{P}$.

Action

The first operation is the prefixing operation $a\rightarrow$. Given any tree t, $a\rightarrow t$ produces a new tree whose root has a single branch which is labelled by a and after a is the same as t. Formally, we have:

For all $t \in PRT$, $a\rightarrow t$ is described by

1. $Nodes(a\rightarrow t) = \{\varepsilon\} \cup \{as \mid s \in Nodes(t)\}$
2. $CNodes(a\rightarrow t) = \{\varepsilon\} \cup \{as \mid s \in CNodes(t)\}$
3. $\mathcal{A}(a\rightarrow t) = \{\{a\}\}$
   $\mathcal{A}(a\rightarrow t, as) = \mathcal{A}(t, s)$
4. $\mathcal{P}(a\rightarrow t) = \emptyset$

Internal Action

We have another unary operator $\tau\rightarrow$ which given any tree as argument produces a new tree which is equal to the argument apart from the fact that it may have additional preemptive power. In fact, the preemptive set of the argument is set to be equal to the acceptance set associated with the its root. Otherwise everything is left unchanged.

For all $t \in PRT$, $\tau\rightarrow t$ is described by

1. $Nodes(\tau\rightarrow t) = Nodes(t)$
2. $CNodes(\tau\rightarrow t) = CNodes(t)$
3. $\mathcal{A}(\tau\rightarrow t, s) = \mathcal{A}(t, s)$
4. $\mathcal{P}(\tau\rightarrow t) = \mathcal{A}(t, \varepsilon)$

Choice Operators

Given two trees from PRT, the various choice operators we consider will glue them together at their roots. The acceptance and preemptive sets associated with the roots of the new trees are obtained from the sets associated to the roots of the original trees. The way in which the new sets are obtained depends on the particular stress the different operators for nondeterminism put on the preemptive power of their operands. Particular care is needed to guarantee that the resulting tree is an element of PRT. For example when glueing two trees, which have an initial action a in common, we need to glue all the subtrees reachable via the branch labelled by a recursively to have only one initial branch of the resulting tree labelled by a. Moreover, since the open nodes can only be the leaves of the preemptive representation trees, whenever an open node is glued to any tree, the overall result will just be the trivial tree consisting of an open node.

In the following we formally define $\sqcap$, $\Box$ and $+$. For the sake of brevity, we will have the convention that if $s \notin CNodes(t)$ then $\mathcal{A}(t, s) = \emptyset$.

Choice

If t_1 and $t_2 \in PRT$ then $t_1 + t_2$ is defined as follows

1. $Nodes(t_1 + t_2) = Nodes(t_1) \cup Nodes(t_2) - \{ss' \mid s \in ONodes(t_1) \cup ONodes(t_2), s' \neq \varepsilon\}$
2. $CNodes(t_1+t_2) = \{s \mid s \in Nodes(t_1+t_2)$ and $s \in Nodes(t_i)$ implies $s \in CNodes(t_i)$, $i = 1,2\}$
3. $\mathcal{A}(t_1 + t_2) = Sat(\{A \cup B \mid A \in \mathcal{A}(t_1), B \in \mathcal{A}(t_2)\} \cup \mathcal{P}(t_1) \cup \mathcal{P}(t_2))$

   $\mathcal{A}(t_1 + t_2, s) = Sat(\mathcal{A}(t_1, s) \cup \mathcal{A}(t_2, s))$ for all $s \in CNodes(t_1+t_2) \cap A^+$.
4. $\mathcal{P}(t_1 + t_2) = \mathcal{A}(t_1 + t_2)$ if $\mathcal{P}(t_1) \cup \mathcal{P}(t_2) \neq \emptyset$

   $= \emptyset$ otherwise

External choice

If t_1 and $t_2 \in PRT$ then $t_1 \Box t_2$ is defined as follows

1. $Nodes(t_1 \Box t_2) = Nodes(t_1 + t_2)$
2. $CNodes(t_1 \Box t_2) = CNodes(t_1 + t_2)$
3. $\mathcal{A}(t_1 \Box t_2) = \{A \cup B \mid A \in \mathcal{A}(t_1), B \in \mathcal{A}(t_2)\}$

   $\mathcal{A}(t_1 \Box t_2, s) = \mathcal{A}(t_1 + t_2, s)$ for all $s \in A^+$
4. $\mathcal{P}(t_1 \Box t_2) = \mathcal{A}(t_1 \Box t_2)$ if $\mathcal{P}(t_1) \cup \mathcal{P}(t_2) \neq \emptyset$

   $= \emptyset$ otherwise

Internal choice

If t_1 and $t_2 \in PRT$ then $t_1 \sqcap t_2$ is defined as follows

1. $Nodes(t_1 \sqcap t_2) = Nodes(t_1 + t_2)$
2. $CNodes(t_1 \sqcap t_2) = CNodes(t_1 + t_2)$
3. $\mathcal{A}(t_1 \sqcap t_2, s) = Sat(\mathcal{A}(t_1, s) \cup \mathcal{A}(t_2, s))$ for all $s \in A^*$
4. $\mathcal{P}(t_1 \sqcap t_2) = \mathcal{A}(t_1 + t_2)$

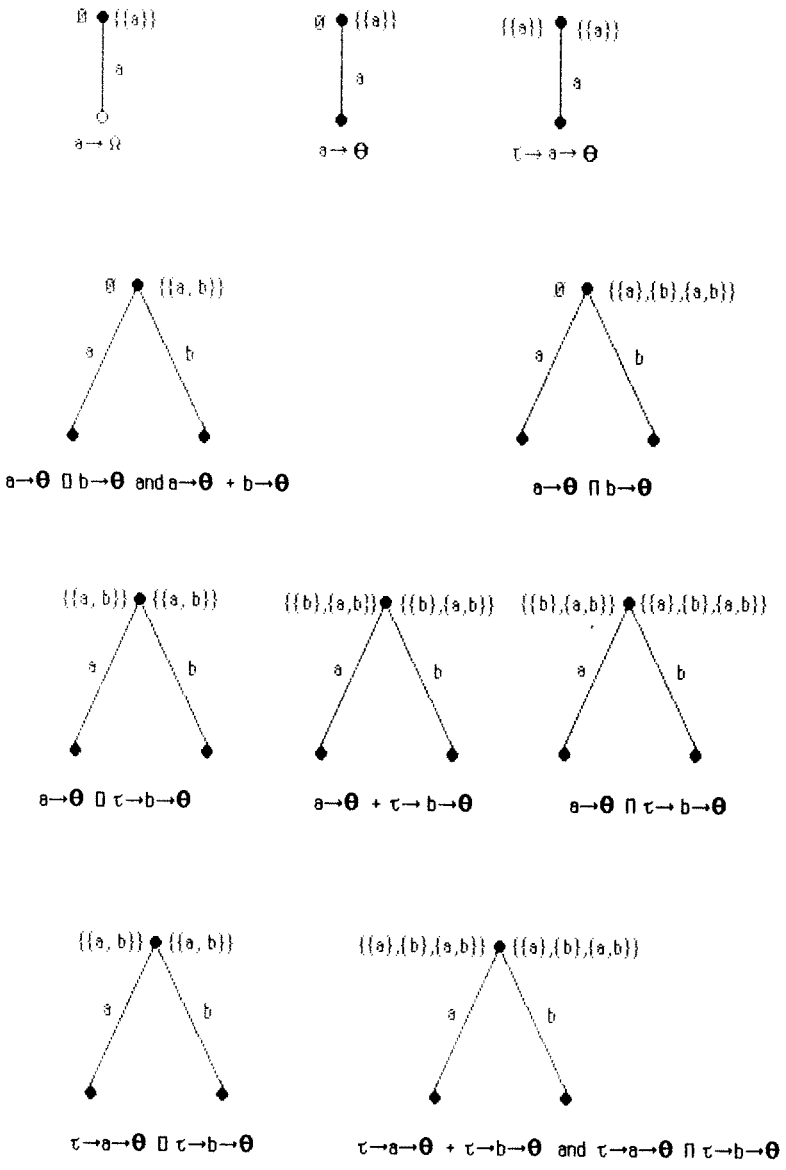


Table 5.1: Operations on PRT

It is worthwhile noting that the three binary operators described above are very similar. In fact, the closed and open nodes of the resulting trees are the same and so are all the acceptance sets associated to non-root nodes. They are distict only because of the different use they make of the preemptive sets and of the acceptance sets associated with the roots of their operands. Table 5.1 shows examples of trees from PRT which are built by using operators in Σ. We also have that every tree is less deterministic than ($<$) the corresponding ones on its right. The following theorem holds and allows us to use PRT as a model for nondeterministic languages by using the general framework of Section 3.1.

Theorem 5.2.1 (PRT, $<$, Σ) is an algebraic Σ-cpo.

Proof: From Theorem 5.1.3, we know that (PRT, $<$) is an algebraic cpo. Therefore to prove the claim we only need to prove that the various operators in Σ are continuous. This follows from known results about sets union and intersection. □

5.3. A comparison between CCS and TCSP

In this section we compare our new model with two of the models we have used in Chapters 3 and 4, to give a denotational semantics to CCS and TCSP. We will show that Strong Representation Trees and Bounded Refusal Sets, together with some of the operations defined on them, are isomorphic as Σ-cpo's to two reducts of Preemptive Representation Trees. In the following, we first give an account of the set of operators on SRT and **B** which we are going to consider, then we define the two reducts of PRT, and finally we outline the proofs of the claimed isomorphisms.

Definition 5.3.1

i. $\Sigma_{CCS} = \{ NIL, \Omega, a, \tau, + \}$;

ii. $\Sigma_{TCSP} = \{Stop, CHAOS, a\rightarrow, \Pi, \Box\}$;

iii. $\Sigma' = \{ \boldsymbol{\theta}, \Omega, a\rightarrow, \tau\rightarrow, + \}$;

iv. $\Sigma'' = \{ \boldsymbol{\theta}, \Omega, a\rightarrow, \Pi, \Box\}$;

v. PRT' is the set of members t of PRT such that

a. $\mathcal{P}(t) = \mathcal{A}(t)$ or

b. $\mathcal{P}(t) = \emptyset$ and $\mathcal{A}(t)$ is a singleton;

vi. PRT'' is the set of members t of PRT such that $\mathcal{P}(t) = \mathcal{A}(t)$. □

Theorem 5.3.2

i. (PRT', <, Σ') is an algebraic Σ'- cpo;

ii. (PRT'', <, Σ'') is an algebraic Σ''- cpo.

Proof:

i. Since PRT' ⊆ PRT and $\Sigma' \subseteq \Sigma$ and (PRT, <, Σ) is an algebraic Σ-cpo (Theorem 5.2.1), to prove the claim we only need to show that the limits of directed subsets of PRT' and all the operators in Σ' are closed with respect to PRT'. Since the acceptance and the preemptive sets of the limit of a directed subset of SRT' are defined in terms of sets intersections (Theorems 3.3.6 and 5.1.3) it is not difficult to verify that either Condition a. or Condition b. of v. in Definition 5.3.1 are satisfied. Therefore, since Theorem 5.1.3 guarantees that the limit of a directed subset of PRT is an element of PRT, we have that the limits of directed subsets of PRT' are elements of PRT'. On the other side it is not difficult to see that + preserves both Condition a. and Conditions b., and that the result of $a\rightarrow t$ and $\tau\rightarrow t$ is always a PRT which satisfies Condition a. and Condition b. respectively.

ii. The proof of this case is similar to the one above. □

We are now ready to study the relationships between our two reducts of SRT and RT_2 and **B**. Firstly, we will study their relationships as cpo's by defining explicitely isomorphism

pairs between SRT' and RT_2 and SRT'' and **B**, afterwards we will study their relationships as Σ-cpo's by exploiting the alternative characterizations of RT_2 and **B** in terms of initial algebras generated by set of axioms. The latter study will have the side effect of giving axiomatic characterizations also for SRT' and SRT''.

Theorem 5.3.3

i. (PRT', <) is isomorphic as a cpo to (RT_2, $<_2$);

ii. (PRT'', <) is isomorphic as a cpo to (**B**, ⊇).

Proof: Since the cpo's we are considering are all algebraic we need only to study the relationships between the set of their isolated elements.

i. We define two monotonic functions between the finite elements of PRT' and RT_2 which are easily seen to be inverses. They leave most parts of their arguments unchanged. In fact they take care only of the acceptance and preemptive sets associated to the roots of their arguments. We start by defining Φ: FIN(RT_2)→FIN(PRT').

If t is an element of FIN(RT_2) then $\Phi(t)$ is an element of FIN(PRT') defined as follows:

1. Nodes($\Phi(t)$) = Nodes(t)
2. CNodes($\Phi(t)$) = CNodes(t)
3. $\mathcal{A}(\Phi(t)) = Succ(t, \varepsilon)$ if $\mathcal{A}(t) = \emptyset$
   $= \mathcal{A}(t)$ otherwise
   $\mathcal{A}(\Phi(t), s) = \mathcal{A}(t, s)$ for all $s \neq \varepsilon$
4. $\mathcal{P}(\Phi(t)) = \mathcal{A}(t)$.

Similarly to Φ, we can define a function Ψ from FIN(PRT') to FIN(RT_2). In this case, if pt is a finite tree from PRT', we have that $\Psi(pt)$ gives us a new tree which is the same as its argument, apart from the fact that it does not have any preemptive set and that if the preemptive set of pt is empty then the acceptance set of the root of $\Psi(pt)$ is empty. Formally Ψ: FIN(PRT') → FIN(RT_2) is defined as follows:

1. Nodes(Ψ(pt)) = Nodes(pt))

2. CNodes(Ψ(t)) = CNodes(pt)

3. $\mathcal{A}$(Ψ(t)) = $\mathcal{P}$(pt)

   $\mathcal{A}$(Ψ(t), s) = $\mathcal{A}$(pt, s) for all s ≠ ξ.

ii. To exhibit the isomorphism pair between **B** and SRT" we must revert to the alternative characterization of **B** given in Theorem 4.3.6. This allows us to describe a refusal sets in terms of sets of traces and sets of maximals of refusals associated to every trace. We will define

θ : FIN(**B**) → FIN(PRT") and

η : FIN(PRT")→FIN(**B**).

If f is an element of FIN(**B**) then θ(f) is an element of FIN(PRT") defined as follows:

1. Nodes(θ(f)) = {s |s ∈ traces(f) and f↓s} ∪ {sa |s ∈ traces(f) and f↓s and a ∈ Succ(f, s)}

2. CNodes(θ(f)) = {s |s ∈ traces(f) and f↓s}

3. $\mathcal{A}$(θ(f), s) = A ÷ **R**(f, s) if s ∈ CNodes(θ(f))

   = ∅ otherwise

4. $\mathcal{P}$(Φ(t)) = $\mathcal{A}$(θ(f)).

If pt is an element of FIN(PRT") then η(pt) is an element of FIN(**B**) defined as follows:

1. traces(η(pt)) = Nodes(pt) ∪ {st | s ∈ Nodes(pt) - CNodes(pt), t ∈ A*}

2. **R**(η(pt), s) = A ÷ $\mathcal{A}$(pt, s) if s ∈ CNodes(pt)

   A otherwise.

Now, Theorem 4.5.8 should be sufficient to convince the reader that η and θ are well defined, monotonic and inverses. □

We can now show that our two reducts, PRT' and PRT", are completely characterized by two simple sets of inequations: the two sets of axioms given in Table 5.2 and 5.3. The proof is based on the following two lemmas.

Lemma 5.3.4

i. (PRT', <, Σ') satisfies all the axioms of Table 5.2;

ii. (PRT", <, Σ") satisfies all the axioms of Table 5.3.

Proof:

i. Most of the axioms in Table 5.2 follow from simple calculations and from set theoretical properties. The only non-trivial ones are N1-N3. As an example, we will outline the proofs of N1 and N3. The nontrivial parts of their proofs consist in proving that $\mathcal{A}$(lhs) = $\mathcal{A}$(rhs) and $\mathcal{P}$(lhs) = $\mathcal{P}$(rhs).

N1. If μ = τ then it is not difficult to see that τ→X + τ→Y = τ→(τ→X + τ→Y) since we have $\mathcal{P}$(τ→X + τ→Y) = $\mathcal{A}$(τ→(τ→X + τ→Y)).

If μ = a ≠ τ then $\mathcal{A}$(lhs) = {{a}} = $\mathcal{A}$(rhs) and the preemptive set of both trees is empty. We only need to prove that $\mathcal{A}$(lhs, a) = $\mathcal{A}$(rhs, a), but it is easy to see that both are equal to $\mathcal{A}$(X) ∪ $\mathcal{A}$(Y).

N3. We will only consider the case μ = a ≠ τ. In this case we have:

$\mathcal{A}$(lhs) = **Sat**({a ∪ B | B ∈ $\mathcal{A}$(Z)} ∪ {a} ∪ $\mathcal{A}$(Z)) = **Sat**({a} ∪ $\mathcal{A}$(Z)) and

$\mathcal{A}$(rhs) = **Sat**({a} ∪ $\mathcal{A}$(Z)) by definition.

ii. In this case too, most of the axioms are trivial to establish. C1, E1-E5, I1-I4 follow straighforwardly from set theoretical reasonings. The proofs of D1 and D2 are very similar to that of N1 given above. We are left to prove D3 and D4. As an example, we prove D3.

D3. $\mathcal{A}$(lhs) = **Sat**($\mathcal{A}$(X) ∪ {A ∪ B | A ∈ $\mathcal{A}$(Y) and B ∈ $\mathcal{A}$(Z)}) and

$\mathcal{A}$(rhs) = { A ∪ B | A ∈ **Sat**($\mathcal{A}$(X) ∪ $\mathcal{A}$(Y)) and B ∈ **Sat**($\mathcal{A}$(X) ∪ $\mathcal{A}$(Z))}.

The desired result will now follow from the properties of saturated sets and of the operation of unioning all the elements of a given pair of sets of set two by two which are reported below:

$\Omega \sqsubseteq X$	Ω1
$X + X = X$	A1
$X + Y = Y + X$	A2
$X + (Y + Z) = (X + Y) + Z$	A3
$X + NIL = X$	A4
$\mu\rightarrow X + \mu\rightarrow Y = \mu\rightarrow(\tau\rightarrow X + \tau\rightarrow Y)$	N1
$X + \tau\rightarrow Y \sqsubseteq \tau\rightarrow(X + Y)$	N2
$\mu\rightarrow X + \tau\rightarrow(\mu\rightarrow Y + Z) = \tau\rightarrow(\mu\rightarrow X + \mu\rightarrow Y + Z)$	N3
$\tau\rightarrow X + \tau\rightarrow Y \sqsubseteq X$	E1-N4

Table 5.2: Axioms for PRT'

$CHAOS \sqsubseteq X$	C1
$X \,\square\, X = X$	E1
$X \,\square\, Y = Y \,\square\, X$	E2
$X \,\square\, (Y \,\square\, Z) = (X \,\square\, Y) \,\square\, Z$	E3
$X \,\square\, Stop = X$	E4
$X \,\square\, CHAOS = CHAOS$	E5
$X \sqcap X = X$	I1
$X \sqcap Y = Y \sqcap X$	I2
$X \sqcap (Y \sqcap Z) = (X \sqcap Y) \sqcap Z$	I3
$X \sqcap Y \sqsubseteq X$	I4
$X \sqcap (Y \,\square\, Z) = (X \sqcap Y) \,\square\, (X \sqcap Z)$	D1
$X \,\square\, (Y \sqcap Z) = (X \,\square\, Y) \sqcap (X \,\square\, Z)$	D2
$(a\rightarrow X) \sqcap (a\rightarrow Y) = a\rightarrow(X \sqcap Y)$	D3
$(a\rightarrow X) \,\square\, (a\rightarrow Y) = a\rightarrow(X \sqcap Y)$	D4

Table 5.3: Axioms for PRT"

1. $\mathbf{Sat}(\{A \cup B \mid A \in \mathcal{A}, B \in \mathcal{B}\}) = \{A \cup B \mid A \in \mathbf{Sat}(\mathcal{A}), B \in \mathbf{Sat}(\mathcal{B})\}$ and
2. $\mathcal{A} \cup \{B \cup C \mid B \in \mathcal{B}, C \in \mathcal{C}\} = \{D \cup E \mid D \in \mathcal{A} \cup \mathcal{B}, E \in \mathcal{A} \cup \mathcal{C}\}$. □

Now that we have shown that the basic axioms for TCSP and CCS are satisfied by our domains, we can follow exactly the same pattern as in the last part of Sections 4.5 and 3.3, i.e. using the relationships between the term algebra factorized by the sets of equations and the algebra of the appropriate normal forms, to prove the wanted characterization.

Theorem 5.3.5 Let $I_{\mathbf{A}^*}$ and $I_{\mathbf{A}^{**}}$ be the initial Σ-cpo's in the category of the Σ-algebras which satisfy the axioms in Table 5.2 and 5.3, respectively. Then

i. $(PRT', <, \Sigma')$ is isomorphic to $I_{\mathbf{A}^*}$

ii. $(PRT'', <, \Sigma'')$ is isomorphic to $I_{\mathbf{A}^{**}}$ □

Finally, since in Chapter 3 and Chapter 4. we have proved that also RT and $\mathbf{B}$ are isomorphic as Σ-cpo's to initial Σ -algebras generated by sets of axioms respectively to $I_{\mathbf{A}_2}$ and $I_{\mathbf{A}^*}$, and since we have that $\mathbf{A}^*$ and $\mathbf{A}^{**}$ contain all the equations of the *primitive* operators in $\mathbf{A}_2$ and $\mathbf{A}^*$, we have that the above theorem implies the next one.

Theorem 5.3.6

i. $(PRT', <, \Sigma')$ is isomorphic to $(RT_2, <_2, \Sigma_{CCS})$

ii. $(PRT'', <, \Sigma'')$ is isomorphic to $(\mathbf{B}, \supseteq, \Sigma_{TCSP})$ □

From the definitions of Section 5.2, it is possible to derive informations about the interrelations of the various SRT operators which given one or two trees as argument transform them into a tree which is more nondeterministic then the starting ones. The derived interrelations together with Theorem 5.3.6 would allow us to get informations about the relationships among the corresponding TCSP and CCS operators.

Theorem 5.3.7: If $P, Q \in PRT$ then we have

$$\tau\rightarrow P + \tau\rightarrow Q \;<\; P \sqcap Q \;<\; P + Q \;<\; P \,\Box\, Q$$

Proof: It is easy to see that

$Nodes(\tau\rightarrow P + \tau\rightarrow Q) = Nodes(P \sqcap Q) = Nodes(P + Q) = Nodes(P \,\Box\, Q)$,

$CNodes(\tau\rightarrow P + \tau\rightarrow Q) = CNodes(P \sqcap Q) = CNodes(P + Q) = CNodes(P \,\Box\, Q)$ and

$\mathcal{A}(\tau\rightarrow P + \tau\rightarrow Q, s) = \mathcal{A}(P \sqcap Q, s) = \mathcal{A}(P + Q, s) = \mathcal{A}(P \,\Box\, Q, s)$ for every $s \in A^+$

we are left only to verify that

$\mathcal{A}(\tau\rightarrow P + \tau\rightarrow Q) \supseteq \mathcal{A}(P \sqcap Q) \supseteq \mathcal{A}(P + Q) \supseteq \mathcal{A}(P \,\Box\, Q)$ and

$\mathcal{P}(\tau\rightarrow P + \tau\rightarrow Q) \supseteq \mathcal{P}(P \sqcap Q) \supseteq \mathcal{P}(P + Q) \supseteq \mathcal{P}(P \,\Box\, Q)$.

Now, because of the way the various operators are defined, for any $P, Q \in PRT$, we have

$\mathcal{A}(\tau\rightarrow P) = \mathcal{A}(P)$ (*)

$\mathcal{P}(\tau\rightarrow P) = \mathcal{A}(P)$ (**)

$\mathcal{P}(P) \subseteq \mathcal{A}(P)$ (***)

and

1. $\mathcal{A}(\tau\rightarrow P + \tau\rightarrow Q) = Sat(\mathcal{A}(P) \cup \mathcal{A}(Q))$
2. $\mathcal{A}(P \sqcap Q) = \mathcal{A}(\tau\rightarrow P + \tau\rightarrow Q)$
3. $\mathcal{A}(P + Q) = Sat(\{A \cup B \mid A \in \mathcal{A}(P), B \in \mathcal{A}(Q)\} \cup \mathcal{P}(P) \cup \mathcal{P}(Q))$
4. $\mathcal{A}(P \,\Box\, Q) = Sat(\{A \cup B \mid A \in \mathcal{A}(P), B \in \mathcal{A}(Q)\})$

a. $\mathcal{P}(\tau\rightarrow P + \tau\rightarrow Q) = \mathcal{A}(\tau\rightarrow P + \tau\rightarrow Q)$

b. $\mathcal{P}(P \sqcap Q) = \mathcal{A}(P + Q)$

c. $\mathcal{P}(P + Q) = \mathcal{A}(P + Q)$

d. $\mathcal{P}(P \,\Box\, Q) = \mathcal{A}(P \,\Box\, Q)$.

It is not difficult to verify that we have

1. = 2. and a. $\supseteq$ b. because of (***)

b. = c. and 2. $\supseteq$ 3. because of (***) and since $\{A \cup B \mid A \in \mathcal{A}, b \in \mathcal{B}\} \subseteq \mathcal{A} \cup \mathcal{B}$

3. $\supseteq$ 4. and c. $\supseteq$ d. trivially.

This suffices to prove the claim. □

It should be noted that the examples of Table 5.1 show that all the above inclusions are proper and so we have that all the above are different operators.

We have that the ordering denoted by $<$ is based on the "amount" of nondetermism exhibited by a particular process; the observational approach taken implies that this is determined by the "amount" of control over the progression of a process which is left to the external environment. For these reasons Theorem 5.3.7 suggests criteria to keep in mind when choosing operators for describing systems.

Unfortunately expressivity is only one of the requirements for a set of operators of a model; it is important that they are orthogonal to each other and that their interactions can be characterized algebraically. In fact the approach followed with TCSP and CCS is to select a subset of the operators just in order to meet some of the above mentioned criteria. Indeed we can see that $\Box$ and $+$ and τ and $\sqcap$ are not orthogonal at all. For example if we have all processes without preeemptive power, i.e. if the terms do not contain any $\tau\rightarrow$ operator, then we have $P \,\Box\, Q = P + Q$, and if all the processes have non-empty preemptive set then $\tau\rightarrow P + \tau\rightarrow Q = P \sqcap Q$. These considerations seem to suggest that the operators $\tau\rightarrow$ and $+$ are more primitive than $\Box$ and $\sqcap$; on the other hand the equations of Tables 5.2 and 5.3 show that $\Box$ and $\sqcap$ and their interaction can be described via elegant algebraic laws. At the moment there is no ground to choose a set of operators against the other, probably only using the various operators to model "real life" systems will allow us to get a definite answer if there exists any.

6. CONCLUSIONS

In the Introduction we pointed out the need for formal models which make it possible to specify and prove properties of systems whose components can proceed in parallel and exchange messages. Operational, denotational and axiomatic approaches have been suggested to tackle this problem and, as a consequence of the large number of properties which can be relevant in the analysis of concurrent, communicating systems, very different models have been proposed. One of the main aims of our research has been to understand the relationships between some of the proposed theories of concurrency and to offer a uniform view of communicating processes.

We have studied the problem by taking the operational approach, which we call *two step operational semantics* as a starting point. This approach is based on Plotkin's **SOS** /Plo81/ and on Milner's idea of *extensional equivalence* /Mil80/. The semantics of communicating processes is described in terms of equivalence classes of Labelled Transition Systems. The transition rules describe the actions which processes can perform and the states in which they terminate after performing the actions. The equivalences, which rely on the reactions of systems to stimuli from the outside world, allow one to ignore internal (unimportant) states of processes and to consider only the behaviour of systems with respect to the outside world.

While we have faithfully followed Plotkin's style of describing system behaviour by means of structured rewrite rules, the observational criteria we have proposed are different from those implied by Milner's observational equivalence. From certain points of view, Milner's equivalence over-discriminates; it is more suitable for describing how systems operate rather than how they can be observed. Our equivalences, which we call *testing*

equivalences, are based only on the reactions of processes to sets of experiments and two processes will be in the same equivalence class only if they "pass" exactly the same tests from a given set. By varying the set of tests and the criteria used to consider an experiment successful, we obtain different equivalence notions. The two level operational semantics defined in this way is completely extensional. We have used this semantics to determine our operational criteria ("the touchstones" /Mil83/) for assessing and comparing different semantic interpretations and different approaches to system semantics.

We have shown how the operational approach, based on testing equivalences, naturally leads to a class of denotational models for nondeterministic processes. We have called this class *Representation Trees*. We have also shown how these models can be used as semantic domains for two fundamental calculi of communicating systems: CCS /Mil80/ and TCSP /HBR84/, and have studied the relationships of the semantics obtained for these two calculi with previously given ones. Moreover, we have shown that, once we adapt our equivalence notions to define testing preorders for CCS we can obtain equational characterizations of the calculus which lead to complete proof systems. The initial algebras generated by the sets of equations are proved isomorphic to some class of Representation Trees. Thus, for CCS we have been able to get equivalent denotational, operational and equational semantics. The operationally generated model of Representation Trees has also been found to be very interesting when dealing with the theory of CSP proposed in /BHR84/. In fact, after studying the original semantics of TCSP based on Refusal Sets and pointing out problems which were caused by the choice of denotations, we have shown that a subset of Representation Trees can be used to model TCSP more appropriately and to obtain equational characterizations and complete proof systems for the theory by using the same techniques developed for CCS. Using Representation Trees as semantic domains for both CCS and TCSP has also had the side benefit of helping us to understand the relationships between the operators of the two calculi.

The testing idea has proved very fruitful. In fact it has also been used by other authors, in other contexts. In /ST85/ "a notion of observational equivalence is defined in which two algebras are observationally equivalent if they both give the same answer to any observation

from a prespecified set". In /Abr83/ the testing approach is used to study the notions of indistinguishability which are embodied in the various powerdomain constructions. In /Mil85/ testing equivalences are used to prove correctness of translations and satisfiability of specifications. In /Mit85/ a version of testing equivalences, which aims at a slightly different treatment of divergent systems, is used to prove the correctness of the implementations of synchronization schemes for CCS. Finally, researches by M. Hennessy have shown that our approach can be extended to treat synchronous calculi and to capture such notions of concurrent systems as fairness, see /Hen83a/ and /Hen84/. Developments of the proof systems based on testing equivalences can be found in /Bro83a/ and /Gol85/.

It is generally true that in any experience a certain moment will arrive in which it is necessary to say that the end has been reached and that the search for improvements or refinements should stop. The research described in this thesis can certainly be improved and extended in many ways. Some of the improvements which it should be possible to carry out without too much additional efforts are listed below.

In Chapter 1, we have seen that Kennaway's equivalence /Ken81/ has major drawbacks because of the way it treats internal moves. We have have studied a version of the equivalence which is close to the original one but does not allow any experimentation on τ-moves. However, although ignoring the τ's in the traces seems inevitable, it would be interesting to study the consequences of keeping the possibility of having must-sets which contain invisible actions. It may well be that in this way we can directly capture congruences for CCS and also avoid explicitly defining the divergence predicate $\Uparrow$.

We feel that in the proof systems of Chapter 2 axiom REC is unnecessary, since it is derivable from the the other axioms by using the general induction rule. The axiom has been kept for technical reasons; we use it to prove the existence of head normal forms, which in turn are used to prove that the general induction rule holds. It should be possible to prove existence of head normal forms without using REC. Another problem is present in Chapter 2. This one is related to the way we have adapted testing equivalences to CCS. In fact, since Ω and

τ^{ω} are treated in much the same way, some identifications which are not immediately intuitive, are induced on CCS terms. For example $abNil + \tau\Omega$ after a MUST {b} and thus that $aNil + abNil + \tau\Omega$ is test equivalent to $abNil + \tau\Omega$.

In Chapter 3 it would be interesting to define operations directly on the three classes of Representation Trees or on the three classes of the corresponding Preemptive Representation Trees. This would allow a better understanding of the implications of the different choices of dealing with divergent experiments.

In Chapter 4, we had to appeal to operational intuitions, to discuss deficiencies of the denotational semantics of TCSP based on Refusals Sets. It would be nice to have an explicit operational semantics of TCSP against which to check the chosen denotations. This has now been developed in /DM85/.

In Chapter 5, together with the study of the relationships between Bounded Refusal Sets and the strict version of Preemptive Representation Trees, which both consider divergence as catastrophic, it would be interesting to study the relationships between the original Refusal Sets model and the version of PRT which considers divergence as underspecification. In fact, the latter two models deal with unbounded nondeterminism in different ways. Refusals sets have only one kind of node whereas representation trees use open and closed nodes to differentiate between bounded and unbounded choices. It would be nice to be able to have just one type of node for Representation Trees as well. This would further simplify the definition of their operations.

Even with all the improvements suggested above we are very far from a solution of the problems mentioned in the introduction. Much remains to be done. In the following we give an idea of some possible directions for future work.

The new equivalence should be investigated for the more general version of CCS which allows value-passing, and new kinds of tests which allow infinite experiments /Bro85, Abr83/ should be considered. The proof techniques and the various example proofs developed

in /Mil80/ and /San82/ should be examined to see if our axioms lead to simpler proofs and if similar proof techniques can be used. It would be interesting to see what the results of immersing the MEIJE language /AB84/ (very similar in spirit to the calculi discussed in the previous chapters but with a radically different and appealing set of primitive operators) into the general setting of testing equivalences would be. More generally, we should be able to produce models for other languages. For example we could see how our models are able to cope with LOTOS /ISO85/, a specification language, based on CCS, which is being defined by the International Organization for Standardization to describe the concurrent aspects of service and communication protocols. We should also study the impact of our approach on general purpose programming languages for concurrent programming such as ADA /Ich80/ or Distributed Processes /BrH78/.

As they stand, the calculi defined above offer a constructive approach to systems specifications. Although this has the advantage of immediate consistency, it has the disadvantage of requiring larger details than necessary. Researches based on the more traditional logical approach to program specifications are very important. This approach avoids problems with abstraction, while leaving open the problem of verifying consistency of logical specifications. Traditionally, formal reasoning about parallel programming has been developed using some kind of modal logic, e.g. temporal logic /OL82, Pnu81, MP83/ and the discussion on the expressiveness of these logics is still open. In recent papers, /Sti83, Sti85, HM84, BR83/, it has been shown that different behavioural equivalences may be characterized by modal languages. It would be interesting to find modal characterizations for testing equivalences and to use them to define modal proof systems and to obtain a taxonomy of behavioural equivalences based on such characterizations. It would be certainly very interesting to set a bridge between specification methods based on process algebras and methods based on logics.

In general, the equivalences and the model presented in the thesis do not distinguish "true" concurrency from its sequential nondeterministic simulation. We are interested in extending the approach discussed above to models of parallelism based on partial orderings, in which

events which are not ordered are considered as concurrent /CFM83, DM84, Lam78, Maz77, Pet80, Win80, Win82/. Admittedly partial ordering models are too concrete in that they offer no possibility to abstract from unwanted details. In order to establish whether two concurrent, non-interleaved, systems are equivalent, we need additional machinery which can allow us to reason about partially (instead of total) ordered sets of events which the systems may perform. In /DDM85/ we showed how partial ordering computations can be associated to CCS terms and in this way we perfomed a first step towards a partial order semantics for this calculus. It should be possible to use the computations of /DDM85/, to take the testing approach and define new congruence classes of CCS terms which coincide with the one presented in Chapter 2 but distinguish interleaving of sequential nondeterministic processes from their concurrent execution.

7. REFERENCES

LNCS n denotes Lecture Notes in Computer Science Volume n. Springer-Verlag

/Abr83/ Abramsky,S. Experiments, Powerdomains and Fully Abstract Models for Applicative Multiprogramming. Proc. FCT 83, LNCS 158, pp. 1-13, (1983).

/AZ82/ Astesiano,E. and Zucca,E. Semantics of CSP via translation into CCS. Proc. MFCS '82, LNCS 116, pp. 172-182, (1982).

/AB84/ Austry,D. and Boudol,G. Algebre de Processus et Synchronization. Theoret. Comput. Sci. Vol. 30, No. 1 North Holland, Amsterdam, (1984).

/BG82/ Burstall,R.B. and Goguen,J.A. Algebras, Theories and Freeness: an Introduction for Computer Scientists. In Theoretical Foundations of Programming Methodology, Broy,M. and Schmidt,G. eds., Reidel Dodrecht, (1982).

/BHR84/ Brookes,S.D., Hoare,C.A.R. and Roscoe,A.D. A Theory of Communicating Sequential Processes. Journal of ACM, Vol. 31, No. 3, pp. 560-599, (1984).

/BJ78/ Bjorner,D. and Jones,C.B. The Vienna Development Method. LNCS 61, (1978).

/BLW82/ Branquart,P. Louis,L. and Wodon,P. An Analytical Description of CHILL, the CCITT High Level Language. LNCS 128, (1982).

/BR83/ Brookes,S.D. Rounds, W.C. Behavioural Equivalence Relations induced by Programming Logics. Proc. ICALP 83, LNCS 154, pp 97-108, (1983)

/BrH75/ Brinch-Hansen,P. The Programming Language Concurrent Pascal. IEEE Transaction on Software Engineering 1, pp.99-205, (1975).

/BrH78/ Brinch-Hansen,P. Distributed Processes: A Concurrent Programming Concept. Communications of ACM, Vol. 21, No. 11, pp. 934-941, (1978).

/Bro83a/ Brookes,S.D. On the Relationship between CCS and CSP. Proc. ICALP '83, LNCS 154, pp 83-96, (1983)

/Bro83b/ Brookes,S.D. A Model for Communicating Sequential Processes. Ph.D. Thesis, University of Oxford. Also Carnegie Mellon University Internal Report, CMU-CS-149, (1983).

/Broy85/ Broy,M. Extensional Behaviour of Concurrent, Nondeterministic, Communicating Systems. In Control Flow and Data Flow: Concepts of Distributed Programming, M. Broy ed., pp. 229-276. Nato ASI Series F, Vol. 14, Springer Verlag (1985).

/CFM83/ Castellani,I., Franceschi, P. and Montanari,U. Labelled Event Structures: A Model of Observable Concurrency. In Formal Description of Programming Concepts II, D. Bjorner ed., pp. 383-399. North-Holland (1983).

/CN76/ Courcelle,B. and Nivat,M. Algebraic Families of Interpretations. Proc. 17th FOCS Annual Symposium, Houston, (1976).

/Coh81/ Cohn,P.M. Universal Algebra. Revised edition, Reidel Dodrecht, (1981).

/Dar82/ Darondeau,Ph. An Enlarged Definition and Complete Axiomatization of Observational Congruence of Finite Processes. LNCS 137, pp.47-62, (1982).

/DDM84/ Degano, P., De Nicola, R. and Montanari U. Partial Ordering Derivations for CCS. Internal Report IEI-Pisa, B84-31, (1984). To appear in Proc. FCT '85, LNCS.

/DH84/ De Nicola,R. and Hennessy,M. Testing Equivalences for Processes. Theoret. Comput. Sci., Vol. 34, pp. 83-133, North Holland, Amsterdam, (1984). Short version in Proc. ICALP '83, LNCS 154, pp. 548-560, (1983).

/DeN83/ De Nicola, R. Two Complete Set of Axioms for a Theory of Communicating Sequential Processes. Internal Report CSR-154-83, University of Edinburgh. To appear in Information and Control, Vol 64. Short version in Proc. FCT '83, LNCS 158, pp. 115-126, (1983).

/DeN84/ De Nicola, R. Models and Operators for Nondeterministic Processes. Proc. MFCS '84, LNCS 176, pp.433-442, (1984).

/DM85/ De Nicola, R. and Montanari, U. Observability of States. To appear in Proc. Workshop on Combining Specification Methods, LNCS, (1985).

/DM84/ Degano, P. and Montanari U. Liveness Properties as Convergence in Metric Spaces, Proc. 16th ACM Sigact Symposium on Theory of Computing '84, pp. 31-38, (1984).

/Dij68/ Dijkstra,E. Cooperating Sequential Processes. In Programming Languages, Genuys,F. ed., pp. 43-113, Academic Press, New York, (1968).

/Fel79/ Feldman,J.A. High Level Programming for Distributed Computing. Communications of ACM Vol. 22, No. 6, pp. 252-268, (1978).

/FHLR79/ Francez,N., Hoare,C.A.R., Lehmann,D.J., and De Roever,W.P. Semantics of Nondeterminism, Concurrency and Communication. Journal of Computer and Systems Science, No. 19, pp.290-308, (1979).

/Gin68/ Ginzburg,S. Algebraic Theory of Automata. Academic Press, New York, (1968).

/Gol85/ Golson,W.G. A Complete Proof System for an Acceptance-Refusal Model of CSP. Internal Report Rice University, COMP TR85-19, (1985).

/Gor79/ Gordon,M. The Denotational Description of Programming Languages. Springer-Verlag, (1979).

/GTWW77/ Goguen,J.A., Thatcher,J.W., Wagner,E.G. and Wright,J.B. Initial Algebra Semantics and Continuous Algebras. Journal of ACM, Vol. 24, No. 1, pp. 68-95, (1977).

/Gue81/ Guessarian,I. Algebraic Semantics. LNCS 99, (1981).

/Hal60/ Halmos,P. Naive Set Theory. Van Nostrand, Princeton (1960)

/Hen82/ Hennessy,M. Powerdomains and Nondeterministic Recursive Definitions. LNCS 137, pp. 178-193, (1982).

/Hen83a/ Hennessy,M. Synchronous and Asynchronous Experiments on Processes. Information and Control Vol. 59, Nos. 1-3, pp. 36-83, (1983).

/Hen83b/ Hennessy,M. A Model for Nondeterministic Machines. Internal Report University of Edinburgh, CSR-135-83, (1983). To appear in Journal of ACM.

/Hen84/ Hennessy,M. An Algebraic Theory of Fair Asynchronous Communicating Processes. Internal Report University of Edinburgh, CSR-171-84, (1984). Short version in Proc. ICALP '85, LNCS.

/Hen85/ Hennessy,M. An Algebraic Theory of Processes. Lecture Notes, Aarhus University, (1985).

/HLP81/ Hennessy,M., Li,W., Plotkin,G. A First Approach at Translating CSP into CCS. Proceedings of the 2nd Int.l Conf. on Distributed Systems, pp.105-115, Paris, (1981).

/HM85/ Hennessy,M., Milner,R. Algebraic Laws for Nondeterminism and Concurrency. Journal of ACM, Vol.32, No. 1, pp. 137-161, (1985).

/HP80/ Hennessy,M., Plotkin,G. A Term Model for CCS. LNCS 88, pp. 261-274, (1980).

/Hoa74/ Hoare,C.A.R. Monitors: An Operating Systems Structuring Concept. Communications of ACM Vol. 17, No. 10, pp. 549-557, (1974)

/Hoa78/ Hoare,C.A.R. Communicating Sequential Processes. Communications of ACM Vol. 21, No. 8, pp. 666- 677, (1978).

/Hoa81/ Hoare,C.A.R. A Model for Communicating Sequential Processes. Technical Monograph Prg-22, Computing Laboratory, University of Oxford, (1981).

/Ich80/ Ichbiach,J.D. et al. Reference Manual for the Ada Programming Language. United States Departement of Defence, (1980). Also LNCS 106.

/ISO85/ International Standard Organization, LOTOS - A Formal Description Technique. ISO/TC97/SC21/WG16-1 N299, (1985)

/Kel76/ Keller,R. Formal Verification of Parallel Programs. Communications of ACM, No. 19, Vol. 7, pp. 561-572, (1976).

/Ken81/ Kennaway,J.K. Formal Semantics of Nondetermism and Parallelism. Ph.D. Thesis, University of Oxford, (1981).

/KH80/ Kennaway,J.K., and Hoare,C.A.R. A Theory of Nondeterminism. LNCS 85, pp. 338-350, (1980).

/KM77/ Kahn,G. and McQeen,D. Coroutines and Networks of Parallel Processes. Proc. IFIP Congress '77, North-Holland, (1977).

/Lam78/ Lamport,L. Time, Clock and the Ordering of Events in a Distributed System. Communications of ACM, Vol. 21, No. 12, pp. 558-564, (1978).

/Lam83/ Lamport,L. What Good is Temporal Logic?. Proc. IFIP Congress '83, pp. 657-668, North-Holland, (1983).

/Lan64/ Landin,P. The Mechanical Evaluation of Expressions, Computer Journal, Vol. 6, No.4, (1964).

/Maz77/ Mazurkievicz,A. Concurrent Program schema and their Interpretation. Proc. Aarhus Workshop on Verification of Parallel Programs, DAIMI-PB-78, (1977).

/Mil73/ Milner,R. Processes: A Mathematical Model of Computing Agents. Proc. Logic Colloquium '72, Rose and Shepherdson, eds., pp. 157-173, North Holland, (1973).

/Mil80/ Milner,R. A Calculus of Communicating Systems, LNCS 92,1980.

/Mil83/ Milner,R. Calculi for Synchrony and Asynchrony. Theoret. Comput. Sci. Vol. 25, 267-310, (1983).

/Mil84/ Milner,R. A Complete Inference System for a Class of Regular Behaviours. Journal of Computers and Systems Sciences, Vol. 28, No. 3, pp.439-466, (1984).

/Mill85/ Millington,M. Fothcoming Ph.D. Thesis. University of Edinburgh, (1985).

/Miln85/ Milne,G. CIRCAL and the Representation of Communication, Concurrency and Time. ACM Toplas Vol. 7, No. 2, pp. 270-298, (1985).

/Mit85/ Mitchell,K. Fothcoming Ph.D. Thesis. University of Edinburgh, (1985).

/MM79/ Milne,G. and Milner,R. Concurrent Processes and their Syntax. Journal of ACM, Vol.26, No.2, pp. 302-321, (1979).

/Moo56/ Moore,E. Gedanken Experiments on Sequential Machines. in Automata Studies, edited by Shannon, C.E. and McCarthy, J., Princeton University Press, (1956).

/Mos83/ Mosses,P. Abstract Semantic Algebras!. In Formal Description of Programming Concepts II, D. Bjorner ed., pp. 45-70. North-Holland (1983).

/MP83/ Manna,Z. and Pnueli,A. How to Cook a Temporal Proof System for your Pet Language. In Proceedings POPL Conference, pp. 141-154, (1983).

/MR76/ Markowsky,G. and Rosen,B.K. Bases for Chain-complete Posets, IBM Journal of Research and Development, Vol. 20, No. 2, pp138-147, (1976).

/MZ83/ Margaria,I. and Zacchi,M. Modelli per processi concorrenti e comunicanti: confronto tra la teoria dei CSP e CCS, CNet Project, Thecnical Report, E.T.S. Pisa, (1983).

/Niv75/ Nivat,M. On the interpretation of Recursive Polyadic Program Schemes. Proc. Symposia Mathematica 15, pp. 255-281, Roma, (1975).

/OH83/ Olderog,E.R. and Hoare,C.A.R. Specification-Oriented Semantics for Communicating Processes, Proc. ICALP 83, LNCS 154, pp. 561-572, (1983).

/OL82/ Owicki,S. and Lamport,L. Proving Liveness Properties of Conurrent Programsms. ACM TOPLAS, Vol. 4, No. 3, pp.455-495, (1982)

/Par81/ Park,D. Concurrency and Automata on Infinite Sequences. Proc. 5th GI Conference, LNCS 104, pp. 167-183, (1981).

/Pet80/ Petri,C.A. Concurrency, in Net Theory and Applications, LNCS 84, pp. 1-19, (1980).

/Plo76/ Plotkin,G. A Powerdomain Constuction, SIAM Journal on Computing, No. 5, pp. 452-486, (1976).

/Plo81/ Plotkin,G. A Structural Approach to Operational Semantics, Lecture Notes, Aarhus University, (1981).

/Plo83/ Plotkin,G. An Operational Semantics for CSP. In Formal Description of Programming Concepts II, D. Bjorner ed., pp. 199-224. North-Holland (1983).

/RB81/ Rounds,W.C., and Brookes,S.D. Possible Futures, Acceptances, Refusals and Communicating Processes. Proc. of 22nd FOCS Annual Symposium, Nashville, (1981).

/Pnu81/ Pnueli,A. The Temporal Semantics of Concurrent Programs. Theoret. Comp. Scie., Vol. 13, pp. 45-60, (1981).

/Ros82/ Roscoe,A.W. A Mathematical Theory of Communicating Processes. Ph.D. Thesis, University of Oxford, (1982).

/San82/ Sanderson,M.T. Proof Techniques for CCS. Ph.D. Thesis, University of Edinburgh, CST-19-82, (1982).

/Sco76/ Scott, D. Data Types as Lattices. SIAM Journal on Computing, Vol. 5, No. 3, pp. 522-587, (1976).

/Smy78/ Smyth,M.B. Power Domains. Journal of Computers and Systems Science, Vol. 2, pp. 23-36, (1978).

/Sti83/ Stirling,C. A Proof Theoretic Characterization of Observational Equivalence. Internal Report University of Edinburgh, CSR-132-83, (1983). To appear in Special issue of Theoret. Comp. Sci. on FCT-TCS Bangalore 1983.

/Sti85/ Stirling,C. A Complete Modal Proof System for a subset of SCCS. Proc. TAPSOFT '85, LNCS 185, pp. 253-266, (1985).

/Sto77/ Stoy,J. Denotational Semantics: The Scott-Strachey Approach to Programming Language Theory. MIT Press, Boston (1977).

/ST85/ Sannella,D.T. and Tarlecki,A. On Observational Equivalence and Algebraic Specification. Internal Report University of Edinburgh, CSR-172-84. Extended abstract in Proc. TAPSOFT '85, LNCS 185, pp. 308-322, (1985)

/TWW79/ Thatcher,J.W., Wagner,E.G. and Wright,J.B. Notes on Algebraic Fundamentals for Theoretical Computer Science . In Fundations of Computer Science III, Mathematical Centre Tracts 109, pp. 83-163, (1979).

/Weg72/ Wegner,P. The Vienna Definition Language, ACM Computing Survey, Vol. 4, No. 1, (1972).

/Win80/ Winskel,G. Events in Computation, Ph.D. Thesis, University of Edinburgh, CST-10-80, (1980).

/Win82/ Winskel,G. Event Structures for CCS and Related Languages. Proc. ICALP '82, LNCS 166, pp. 561-576, (1982).