Bisimulations for Concurrency

Ilaria Castellani

Doctor of Philosophy
University of Edinburgh
1987

Abstract

We study three notions of bisimulation equivalence for concurrent processes. Bisimulation equivalences are based on an operational interpretation of processes as labelled transition systems, and constitute the strongest notion of equivalence one may adopt for such systems: two systems are equivalent if and only if they have the same step–by–step behaviour.

We focus first on Milner's notion of *weak bisimulation* (also known as observational equivalence) and propose an alternative formulation for it. More specifically, we show that Milner's notion may be redefined as one of reducibility to a same system – via a reduction function called *abstraction homomorphism*. We use our characterisation to derive a complete set of reduction rules for observational equivalence on finite processes. We also show how abstraction homomorphisms may be extended to labelled event structures: however we do not consider the possibility of unobservable events here.

We look then for notions of bisimulation which account for the concurrent aspects of processes. Traditional transition systems – evolving via successive elementary actions – only provide an interleaving semantics for concurrency.

We suggest two generalisations of the notion of transition system: *distributed transition systems*, obtained by generalising the residual of a transition, and *pomset transition systems*, obtained by extending the notion of action labelling a transition (an action being now a partially ordered multiset). For the latter we find a corresponding notion of bisimulation on labelled event structures.

Based on these new kinds of transitions, we obtain two bisimulation equivalences – one stronger than the other – which are both more discriminating than Milner's equivalence. For both of them we present an algebraic characterisation by means of a complete set of axioms.

Acknowledgements

I would like first of all to express my deep gratitude to my supervisor M. Hennessy, for his guidance, encouragement, multiple suggestions, and moral support in difficult moments. Whichever "method" of research I may have acquired in these years, I feel I owe it in large part to him.

I am very grateful to R. Milner, for being such an inspiring figure, for his willingness to listen and comment on my work, and for giving me helpful advices.

I would also like to thank many people involved in the theory postgraduate course in Edinburgh, for taking interest in my work and giving me their comments: Marek Bednarczyk, Gerardo Costa, Rocco De Nicola, Kim Larsen, Eugenio Moggi, K.V.S. Prasad, Edmund Robinson and Allen Stoughton, among others.

The definition of abstraction homomorphism and the idea of using it to characterise Milner's notion of observational equivalence were first elaborated with U. Montanari at Pisa University. I would like to thank him for encouraging my interest in "true concurrency", and for many stimulating discussions.

Thank you so much to Aarhus University (DAIMI) for their kind hospitality and particularly to Mogens Nielsen and P.S. Thiagarajan for very instructive discussions during my six month stay there.

Finally I would like to thank INRIA and CMA at Sophia–Antipolis, and particularly all people in the research group directed by G. Berry, for providing me with such a rich and sympathetic environment during the writing of this thesis. Thank you particularly to G. Boudol for his continuous interest and many useful suggestions, as well as for reading a draft of this work. Some of this material (the pomset transition semantics of chapter 6) was elaborated with him.

During my stay in Edinburgh I was supported by a research studentship from CNR, the Italian National Council of Research. Since January 86, I have been an "invited researcher" at INRIA.

Declaration

This thesis was composed by myself. Chapters 4 and 5 are essentially a revision of the paper [CH 85] , produced in collaboration with M. Hennessy, and Chapter 6 is derived from a paper written with G. Boudol [BC 86] . For the rest the work reported is my own, developed under the guidance of my supervisor M. Hennessy.

Contents

1 Introduction 6

2 Abstraction on nondeterministic processes 15

2.1 Labelled transition systems 16

2.1.1 Bisimulation relations 17

2.2 Nondeterministic Systems 18

2.3 Nondeterministic Processes 19

2.3.1 Abstraction homomorphisms 22

2.3.2 Abstraction equivalence 33

2.4 Relating bisimulations to abstraction homomorphisms 34

2.4.1 Weak bisimulation relations 34

2.4.2 Quasi-bisimulations 35

2.4.3 Characterisation of abstraction homomorphisms 38

2.4.4 Abstraction equivalence equals bisimulation equivalence . 41

2.5 Minimal NDP's 42

2.6 A language for finite processes 43

2.7 Discussion 49

3 Abstraction on concurrent processes 52

3.1 Some existing LTS models for concurrent programs 55

3.1.1 The reference language 55

3.1.2 Algebraic calculi: CCS, MEIJE, SCCS 56

3.1.3 Partial ordering models. Petri nets firing rules 62

3.2 Labelled Event Structures 71
3.3 Interpretation of terms as LES's. 72
3.4 Abstraction homomorphisms on LES's 74
3.5 Characterisation of the abstraction relation 80

4 Distributed Bisimulations **90**

4.1 Distributed transition systems 91
4.2 Distributed bisimulation equivalence 94
4.3 Adding communication 99
4.4 Alternative formulations of the semantics 100
4.5 Algebraic characterisation 108

5 Weak distributed bisimulations **118**

5.1 Weak distributed transition systems 119
5.2 Weak distributed bisimulation equivalence 122
5.3 Alternative formulation of the semantics 124
5.4 Behavioural congruence 130
5.5 Algebraic Characterisation 132

6 Pomset transition systems **143**

6.1 Pomset bisimulation equivalence 146
6.2 A corresponding bisimulation equivalence for Labelled Event Structures 150
6.3 Algebraic characterisation 154
6.4 Comparison with other equivalences 158

7 Conclusions **160**

Chapter 1

Introduction

In recent years much effort in theoretical computer science has been devoted to the search of semantic models for concurrent programs.

A concurrent program is inherently part of a larger environment, with which it interacts in the course of its computation. It is therefore to be regarded as a *reactive system* – in Pnueli's terminology [Pnu 84] – or perhaps more appropriately, to suggest a looser correlation with the environment, as an *interactive* system.

It has now become common sense that functional semantics, describing one program's behaviour as a function from an input to an output state, is not adequate for communicating programs. On the one hand intermediate states of computation cannot be overlooked, since they are relevant to the program's behaviour in a surrounding environment. Properties like deadlock, liveness etc. cannot be described in an input–output semantics. On the other hand the notion of final state, which is crucial to functional semantics, is not as important here. A program may be nonterminating, and thus meaningless as a simple input–output function, and yet have a perfectly defined semantics as an interactive system (think for instance of operating systems or process control systems).

More generally, attempts at using denotational semantics for concurrent processes (yielding interleaving models) have not been completely successful, mainly because they failed to match operational intuitions [Plo 76, MM 79] .

What prompted the abandon of denotational models in favour of operational ones, and gave in general a new impulse to operational semantics, was Plotkin's SOS (Structural Operational Semantics [Plo 81]), which allows for clean op-

erational specifications in terms of *structural rules*, thus recovering some of the advantages of denotational semantics. More specifically Plotkin shows that *labelled transition systems* determined by such structural rules may be used to model a wide range of programming languages and concepts.

Labelled transition systems (LTS's) are essentially objects evolving from one state to the other via successive transitions. Hence LTS's appear as a convenient model for concurrent programs characterised by their on–going behaviour. In fact most recent research in concurrency, starting with Milner's Calculus of Communicating Systems (CCS) [Mil 80] , has been based on an interpretation of concurrent programs as LTS's. Each transition is labelled by an *action*, representing an interaction with the environment or an internal computational task.

Because of their simple and general definition, labelled transition systems are widely used in semantics. Adopted to define the semantics of abstract machines and for modelling sequential and concurrent languages, they also provide the semantic universe for dynamic and temporal logics ([Par 81, Pnu 85, Sti 85]), and support notions of effective or computable process [Bou 85, DSi 85] .

A notable advantage of LTS's, as regards the modelling of program behaviours, is that they allow for different levels of description of the same program. By varying the definition of the transition relation one obtains a whole range of different descriptions of the same program, going from a full account of its structure to some more interesting "abstract" specifications. Examples of abstract transitions are Milner's *weak* transitions for CCS, as well as the *observation criteria* of [Bou 85] , which are at the basis of a verification system for parallel processes [LMV 87] .

In spite of their internal flexibility, transition systems still need to be factored by *equivalence relations* to yield an adequately abstract model. Several equivalences (and preorders) have been put forward to this purpose, corresponding to different notions of extensional behaviour. We mention here some of the proposed equivalences, by order of increasing strength: *string* equivalence – the traditional language–theoretic equivalence, *testing* and *failure* equivalences [DH 84,BHR 84] , *observational* equivalence [Mil 80] , and *bisimulation* equivalence [Park 81, Mil 82] .

We shall be mostly interested here in bisimulation equivalence, a refinement of Milner's observational equivalence [Mil 80, HM 85] that rapidly superseded it in Milner's own works (mainly by virtue of better mathematical properties). In fact the two equivalences coincide on a large class of processes and we shall often use the two names interchangeably. A first concern of this thesis will be to characterise the weak bisimulation equivalence as a relation of reducibility to a same process, as explained in the second part of this introduction.

While they account for the interactive and nondeterministic aspects of concurrent computation, ordinary transition systems do not seem suited to represent the *nonsequential behaviour* of processes. Evolving by sequences of transitions, LTS's essentially simulate parallelism between independent processes by a choice between the possible interleavings of their activities. They provide a so–called *interleaving semantics* of concurrency.

In fact, interpreting concurrency as nondeterministic interleaving may be a useful simplifying assumption. It has allowed for the construction of elegant algebraic models in which the specification and verification of large concurrent systems are possible. All the models mentioned above [BHR 84, DH 84, HM 85, Mil 80] are based on interleaving assumptions. It may be argued, on the other hand, that such assumptions are not realistic: that they may be justified in the case of a single processor performing multitasking, but do not appropriately account for distributed systems.

Perhaps the most representative theory of "true concurrency" is that of Petri nets, developed from the early work of Carl Adam Petri [Petri 62] . Petri nets provide a nonsequential model for concurrent systems, built on the primitive notions of *causality* and *concurrency*, which has been at the basis of many subsequent proposals. Though operational in nature, this model did not have until recently [Gra 81, Rei 85, GV 87] a transition system semantics reflecting its original intention. The usual operational rules (*firing rules*) for Petri nets – allowing a single transition or a set of concurrent transitions to fire in one step – do not fully preserve the causal and concurrent structure of a system. Thus these rules, which proved very useful in the analysis of properties of nets, did not lead to a satisfactory notion of behaviour of a net.

In [Petri 77] Petri indicated in the concept of *process* – roughly, a deterministic unfolding – the appropriate notion of nonsequential computation for

a net. The *behaviour* of a net is then defined to be the set of its processes. Processes have been subsequently studied and generalised in a series of works [Best 84, GSW 80, GR 83, Rei 85] . In fact, what seems essential in a process is the *partial order* of its transitions, what W. Reisig calls *abstract processes* in [Rei 85] . The idea of concentrating on the partial order of events of a Petri net – of a particular kind – was used also for building the theory of *event structures*, started by Nielsen, Plotkin and Winskel [NPW 79] and then developed mainly by Winskel [Win 87] .

An event structure (ES) describes a concurrent nondeterministic computation as a partially ordered set of events. The ordering represents causality; two non–related events may be in one of two relations, concurrency or conflict. The notion of *configuration* of an ES is introduced to represent the possible stages of a computation: in fact a configuration is something very similar to an abstract process. Event structures constitute a simple system model, where the relations of concurrency and causality among events (transitions) are clearly expressed; moreover they are close to abstract processes and thus to a behavioural representation of a net. This explains why they have received much attention as a model for concurrency in recent years [CFM 82, Win 82, BC 86, GV 87] .

As well as labelled transition systems, Event structures play a unifying role in the semantics of languages. We just saw that the configuration space of an ES is a way of representing the behaviour of a Petri net. On the other hand one may use labelled event structures to model algebraic languages like CCS and CSP (see [Win 82]). Event structures also provide a very clean model for data type theory: here the space of configurations is the domain of possible values of a type. Evolving from Kahn & Plotkin's *concrete domains*, ES's give a concrete representation of a data type, well–suited for the definition of standard domain constructions [Win 87] .

To come back to the modelling of concurrency, many other authors have been concerned with partial ordering models. An earlier formalization of a partially ordered set of actions was given by Mazurkiewicz, with the notion of *trace* [Maz 84] . This concept was generalised to that of *pomset* – partially ordered multiset – by Pratt in [Pra 85] . In his thesis [Gi 84] , Gischer presents very elegant results about pomsets, and gives a characterisation of the class of pomsets required to model a simple algebra of concurrent processes. The notion

of pomset is also related to that of *partial word* studied by Grabowski [Gra 81] . Nonsequential behaviours have also been studied extensively by M. Shields [Shi 82] . Another contribution to this line of research is Montanari & Degano model of *concurrent histories* [DM 86] , sort of concrete computations that may be concatenated, and from which one can eventually extract a partial ordering of atomic actions. This approach has been used in [DDM 85, DDM 87] to give a partial ordering semantics for CCS.

To conclude this discussion, we should like to mention another point of difference between true concurrency models and the algebraic models mentioned earlier. The first are mostly concerned with *representations* of processes, and generally do not provide constructs (a language) for defining processes. As a consequence these models lack a notion of hierarchical decomposition, and do not lend themselves to the definition of abstraction mechanisms. On the contrary algebraic models are based on the choice of a syntax (a set of programming constructs) for processes, and the semantics itself – describing processes as labelled transition sytems – is syntax directed (structural, in the sense of Plotkin). Based on the operational semantics, a notion of *behavioural equivalence* – be it bisimulation, testing equivalence or others – is introduced on processes. Because processes are essentially terms of an algebra, it is often possible to develop equational theories for the behavioural equivalence.

* * *

In this thesis, we present an attempt at bringing together the advantages of partial ordering models, representing causality and concurrency as primitive notions, and algebraic models, expressing a chosen set of primitive constructs and offering a now consolidated methodology for reasoning about processes. We shall now briefly discuss our guiding ideas, before passing to a more detailed overview of our work.

Our starting point are some existing algebraic calculi of processes, essentially CCS and MEIJE – a calculus presented in [AB 84] , which already accounts for some aspects of nonsequential behaviour. We concentrate on a simple algebraic language CL whose operators are prefixing by an atomic action (a:), sum ($+$) and parallel composition ($|$). This is a subset of both CCS and MEIJE (as well as of other algebraic languages for concurrency).

The terms of this language are interpreted in CCS as transition systems performing successive *atomic* actions a, b, etc. Thus CCS provides an interleaving model of concurrency, leading typically to the identification:

$$a \mid b \;=\; a{:}b \;+\; b{:}a$$

The calculus MEIJE gains some more descriptive power as regards concurrency by allowing *compound* actions made out of concurrent atomic actions. Hence the above identification is not valid in MEIJE, since the first process has a compound action $a \cdot b$, which the second cannot match. On the other hand MEIJE retains a notion of *global time* and does not capture all the relations of causality among actions. For example the two processes:

$$a \mid b \qquad \text{and} \qquad a \mid b \;+\; a{:}b \;+\; b{:}a$$

have the same operational behaviour in MEIJE, whereas the second one shows some additional *causal dependencies* which, to our view, should not be ignored.

A main object of this thesis is to devise a more expressive semantics for the language *CL*, capable of reflecting both the *causal* and *concurrent* relations specified in a process. Basically, this is achieved by *enriching* the transitions:

$$p \xrightarrow{a} p'$$

of a system. We have studied essentially two generalisations of the notion of transition system : *distributed transition systems*, obtained by generalising the residual p' of a transition to a compound residual $<p', p''>$, and *pomset transition systems*, obtained by extending the notion of action a labelling a transition (an action being now a partially ordered multiset).

The idea underlying *distributed transition systems* is that for any atomic action, one may identify the local component which has performed the action. More precisely, a distributed transition gives rise to a compound residual, made out of a *local* component and a remote *concurrent* component. For example the process $a{:}b \mid c$ will have a transition:

$$a{:}b \mid c \;\xrightarrow{a}\; <b, c>$$

where b and c are respectively the local and concurrent residuals. This description reflects a view of concurrent processes as distributed in space. Separating

the components allows us, intuitively, to distinguish causality – relating an action to its local residual – from concurrency – relating it to the nonlocal residual. Based on the new transitions, we define a *distributed bisimulation* equivalence on DLTS's, which acts recursively on both components of the residual.

On the other hand *pomset transitions* are obtained from CCS transitions by relaxing the requirement of *atomicity* for actions (both in space and in time). We saw that the calculus MEIJE already renounces atomicity in *space* for actions, by allowing parallel composition within actions. We proceed one step further and drop the condition of atomicity in *time* too, by introducing sequencing (prefixing) in the actions. In this way actions become whole nonsequential *computations*. We have for example a transition:

$$a:(b+c) \mid d \xrightarrow{a:b \mid d} \text{NIL}$$

where the action d occurs in parallel with the *sequence* $a:b$. In this sense we obtain a completely asynchronous calculus, free of global time assumptions.

We shall see that pomset transitions allow for an exact representation of the relations of concurrency and causality between atomic actions of a process.

Distributed transitions give a more intensional description: they capture the relations between atomic actions as well as the relation between actions and *choices*. As a consequence distributed bisimulation equivalence is strictly included in pomset bisimulation equivalence (which is in turn incuded in Milner strong bisimulation).

* * *

The remainder of this introduction is devoted to a summary of the work presented in the thesis.

This work is roughly divided in two parts. The first part, which includes chapters 2 and 3, is dedicated to the study of *abstraction homomorphisms*, functions that simplify the structure of a process while preserving their behaviour. We show how such morphisms may be used to define a notion of abstract behaviour, both on transition systems (chapter 2) and on labelled event structures (chapter 3).

In chapter 2, we use the notion abstraction homomorphism to establish an alternative formulation for Milner's *weak bisimulation* equivalence. We concentrate on a class of acyclic labelled transition systems determined by weak transitions, which we call *nondeterministic processes.* We define abstraction homomorphisms on this class and a corresponding reduction relation on processes. We show that this relation enjoys the Church-Rosser property, and yields unique minimal forms for processes. Based on this relation, we then introduce an *abstraction equivalence* $\sim_{abs}$ on processes: two processes are equivalent iff they are reducible to a same process. We then study the relationship between our notion of reduction and that of weak bisimulation. We show that abstraction homomorphisms are essentially *single-valued bisimulations.* As a corollary we may prove that our abstraction equivalence coincides with weak bisimulation equivalence, and can therefore be used as a simple alternative formulation for it. Our characterisation proves helpful in deriving a a complete set of *reduction rules* for the weak equivalence on finite nondeterministic processes.

In chapter 3, after a short review of the algebraic calculi CCS, MEIJE and SCCS, and a discussion on Petri nets, we introduce the model of Labelled Event Structures (LES's). We may then interpret processes of CL (the language mentioned earlier) as finite LES's. We extend to LES's our definition of *abstraction homomorphism*, and present an axiomatisation for the corresponding reduction relation. In conclusion we suggest the possibility of deriving a notion of bisimulation (on LES's) from that of abstraction homomorphism.

Chapter 4 is devoted to the study of *distributed transition systems* (DLTS's), which partly originated as an attempt to provide an operational counterpart for abstraction homomorphisms on LES's. As mentioned earlier, distributed transitions yield a compound residual, and the corresponding *distributed bisimulation equivalence* $\sim_d$ tests recursively both components of the residual. DLTS's are used first to interpret terms of CL. We may then show that $\sim_d$ is weaker than the abstraction equivalence $\sim_{abs}$ defined on LES's in chapter 3. In a further section we extend our DLTS semantics to deal with communication. We give an alternative formulation of the semantics, which uses a local and a *global* residual, and present a complete axiomatisation for $\sim_d$ on the language with communication.

Chapter 5 extends the DLTS semantics to a language with unobservable actions (and communication). Following Milner's style, we describe the behaviour of processes by *weak* distributed transitions. The treatment of the corresponding *weak distributed bisimulation* $\approx_d$ is considerably more complicated than that of $\sim_d$. However the results of chapter 4 mostly carry over to weak DLTS. In particular, we find a complete axiomatisation for (the congruence induced by) $\approx_d$.

In chapter 6 we present an interpretation of processes of CL as *pomset transition systems*, whose actions are partially ordered sets. We show that our semantics for terms agrees with an (intuitive) transition system semantics for labelled event structures, where transitions are labelled by *configurations* of events. We give an algebraic characterisation for the new bisimulation $\asymp$, called *pomset bisimulation equivalence.* Finally we establish the relation between $\asymp$ and the bisimulations examined in previous chapters: it turns out that $\asymp$ is strictly included between our distributed bisimulation $\sim_d$ and Milner's strong bisimulation $\sim$.

Chapter 7 presents a short conclusion and points to some interesting open problems.

Chapter 2

Abstraction on nondeterministic processes

Labelled transition systems (LTS's) [Kel 76, Plo 81] are generally recognised as an appropriate model for nondeterministic processes. LTS's describe processes as evolving through states via successive transitions. Each transition is labelled by an *action*, which may represent an interaction with the environment or an internal computational task.

LTS's are often considered modulo *bisimulation equivalence*, a notion proposed by Park and Milner [Park 81, Mil 83] : informally speaking, two systems are said to be equivalent if a full correspondence can be established between their sets of states in such a way that corresponding states are accessed by the same transition sequence and give rise to equivalent subsystems.

We shall here concentrate on a class of acyclic labelled transition systems – with internal *unobservable* actions – that we call *nondeterministic processes.* We show that for this class of systems, the notion of bisimulation equivalence may be restated as one of "reducibility to a same system" via a simple reduction relation. This relation is proven to enjoy the Church-Rosser property, and to yield unique minimal forms for processes. We also show that, when restricted to finite nondeterministic processes, the relation can be characterised algebraically by a set of reduction rules.

The chapter is organised as follows. In section 2.1 we recall the definitions of labelled transition system and bisimulation. In sections 2.2 and 2.3 we present our computational model, the class of *nondeterministic processes.* We argue that this basic model is not abstract enough, particularly when systems are

allowed unobservable transitions as well as observable ones. We therefore introduce *abstraction homomorphisms* [CFM 82] as a means of simplifying the structure of a process by merging together some of its states: the result is a process with a simpler description, but "abstractly equivalent" to the original one. We can then infer a reduction relation between processes from the existence of abstraction homomorphisms between them. We prove some significant properties of this relation, such as substitutivity under standard operators and the Church-Rosser property. Based on the reduction relation, we define an *abstraction equivalence* relation on processes: two processes are equivalent iff they are reducible to a same process.

In section 2.4 we study the relationship between our notions of reduction and abstraction and the notion of *bisimulation* between transition systems. The criterion we use for identifying states of a process via abstraction homomorphisms is similar to the one underlying the definition of bisimulation: we show in fact that our abstraction equivalence coincides with (the substitutive version of) *bisimulation equivalence*, and can therefore be used as a simple alternative formulation for it. In section 2.5 we establish a notion of *minimality* for processes: a process is minimal if it cannot be further reduced by means of abstraction homomorphisms.

In section 2.6 we consider a small language for defining finite nondeterministic processes: essentially a subset of R. Milner's CCS. On this language our equivalence is just Milner's *observational congruence*, for which a complete finite axiomatisation has been given in [HM 83] . So, on the one hand, we get a ready-made algebraic characterisation for abstraction equivalence; on the other hand, our characterisation proves helpful in working out a complete system of *reduction rules* for that language.

2.1 Labelled transition systems

We recall here the definition of *labelled transition system*, a general model of computation first proposed by Keller [Kel 76] and subsequently extensively used for describing concurrent programs (see [Plo 81] , [Mil 84]).

Labelled transition systems describe programs as progressing through states by sequences of actions. Formally, if A is a set of *actions*:

Definition 2.1 *A labelled transition system (LTS) over A is a triple $S = (Q \cup \{r\}, A, \longrightarrow)$, where $(Q \cup \{r\})$ is the set of states of S, $r \notin Q$ is the initial state (or root) of S, and $\longrightarrow \subseteq [(Q \cup \{r\}) \times A \times (Q \cup \{r\})]$ is the transition relation on S.*

We shall use $q, q' \ldots$ to range over Q, and $a, b \ldots$ to range over A. The interpretation for $(q, a, q') \in \longrightarrow$ is that S may evolve from state q to state q' via a action a. We will generally write $q \xrightarrow{a} q'$ in place of $(q, a, q') \in \longrightarrow$.

For $L \subseteq A$, let $\xrightarrow{L} =_{def} \{(q, q') \mid \exists\, a \in L \text{ s.t. } q \xrightarrow{a} q'\}$. With a slight abuse of notation, we shall write $\longrightarrow^*$ for the transitive and reflexive closure of $\xrightarrow{A}$, which we call the *derivation relation* on S. We may assume that all states of an LTS $S = (Q \cup \{r\}, A, \longrightarrow)$ are accessible from the root, that is: $\forall q \in Q, r \longrightarrow_S^* q$. For an LTS $S = (Q \cup \{r\}, A, \longrightarrow)$, we will use Q_S, $\longrightarrow_S$, instead of Q, $\longrightarrow$, whenever an explicit reference to S is required.

Note: one does not in general require LTS's to have an initial state. We adopt here rooted LTS's since we are primarily interested in modelling individual programs.

According to our definition, an LTS S is a machine starting in some definite state and evolving through successive states by means of elementary transitions. On the other hand, each state of S may be thought of as the initial state of some other LTS: we may then regard the system S as giving rise to new systems, rather than going through successive states.

In fact, the whole class $\mathcal{L}$ of LTS's may be described as a (nonrooted) labelled transition system whose states are LTS's. The transition relation $\longrightarrow_{\mathcal{L}}$ is defined as follows: if $S = (Q \cup \{r\}, A, \longrightarrow)$ and $a \in A$, then $S \xrightarrow{a}_{\mathcal{L}} S'$ iff $S' = (Q' \cup \{r'\}, A, \longrightarrow)$ with $r \xrightarrow{a}_S r'$, $Q' \cup \{r'\} = \{q \in (Q \cup \{r\}) \mid r' \longrightarrow_S^* q\}$. We say that S' is a *derivative* of S whenever $S \longrightarrow_{\mathcal{L}}^* S'$. Thus for any $S \in \mathcal{L}$, a one-to-one correspondence can be established between the states and the derivatives of S. In the following we will often avail ourselves of this correspondence between states and (sub)systems.

2.1.1 Bisimulation relations

A natural method for comparing different computing systems is to check to which extent they can *behave like* each other, according to some definition of

behaviour.

The behaviour of an LTS S is determined step by step by the actions it can perform. For any $a \in A$, let us say that S' is an *a-derivative* of S if $S \xrightarrow{a}_{\mathcal{L}} S'$. Then the behaviour of S can be defined recursively in terms of the behaviours of its a-derivatives, for any $a \in A$.

Based on such a recursive notion of behaviour, one gets an (equally recursive) notion of equivalence of behaviour, or *bisimulation equivalence*, between LTS's: two LTS's are said to bisimulate each other if for any a-derivative of either of the two, there exists an a-derivative of the other, such that the two derivatives still bisimulate each other. The definition of bisimulation equivalence, which is given below, is due to D. Park [Park 81] and R. Milner [Mil 83] .

Definition 2.2 *A bisimulation is a relation* $R \subseteq (\mathcal{L} \times \mathcal{L})$ *s.t.* $R \subseteq B(R)$, *where* $(S_1, S_2) \in B(R)$ *iff* $\forall a \in A$:

i) $S_1 \xrightarrow{a} S_1'$ *implies* $\exists\, S_2'$ *s.t.* $S_2 \xrightarrow{a} S_2'$, *with* $S_1' \, R \, S_2'$

ii) $S_2 \xrightarrow{a} S_2'$ *implies* $\exists\, S_1'$ *s.t.* $S_1 \xrightarrow{a} S_1'$, *with* $S_1' \, R \, S_2'$

In other words a bisimulation is a postfixed-point of the function B. Since B is monotonic for relations under inclusion, it has a *largest* postfixed-point (which is also its largest fixed-point) given by $\bigcup\{R \mid R \subseteq B(R)\}$. This largest bisimulation is easily shown to be an equivalence relation. We shall denote it by $<\sim>$, and refer to it as *the bisimulation equivalence.*

2.2 Nondeterministic Systems

In the previous section, we introduced LTS's without specifying anything about the nature of their actions. We shall now get more precise about this point.

To model the nondeterministic aspects of concurrent systems, it is sufficient to consider LTS's performing *atomic* (unstructured) actions – or sequences of such actions. From now onwards, we shall call *nondeterministic system* (NDS) any such LTS. To account for specifically concurrent aspects, on the other hand, we will need LTS's with *composite* actions, generally involving some concurrency between their constituents. Such LTS's will be treated in chapter 6.

In the rest of this chapter, we shall only be concerned with NDS's. In fact, we are interested in NDS's with two kinds of atomic actions: *observable* and *unobservable* ones. Unobservable actions were introduced by Milner in his calculus CCS (Calculus for Communicating Systems) to represent internal communications. Intuitively, since the communication mode of CCS is that of binary rendez-vous, a communication action is no longer available for external observation – or for a rendez-vous (observers and communicating agents are treated in the same way in CCS). We shall here entirely adopt Milner's motivations, and generally use CCS notation. For a proper background we refer to Milner's book [Mil 80] .

We shall thus assume the set A to contain a single *unobservable* action τ. Henceforward, NDS's will be labelled over this enlarged set of actions. We will now use $\mu, \nu \ldots$ to range over A, and $a, b \ldots$ to range over (A - τ).

Let $\mathcal{S}$ denote the class of all NDS's. We suppose $\mathcal{S}$ to be closed w.r.t. some simple *operators* (taken from CCS): a nullary operator NIL, a set of unary operators μ: (one for each $\mu \in A$), and a binary operator $+$. The intended meaning of these operators is the following: NIL represents a terminated system, $+$ is a nondeterministic choice operator, and the μ:'s provide a simple form of sequentialisation, called *prefixing* by the action μ. We shall generally write μS in place of $\mu : S$.

The transition relation of a compound NDS is taken to satisfy the properties:

i) $\mu S \xrightarrow{\mu} S$

ii) $S \xrightarrow{\mu} S'$ implies $(S + S'') \xrightarrow{\mu} S'$, $(S'' + S) \xrightarrow{\mu} S'$

The operators NIL, μ: and $+$ will be given a precise definition for a subclass of $\mathcal{S}$, the class of *nondeterministic processes* that we introduce in the next section.

2.3 Nondeterministic Processes

As they are, NDS's have an immediate representation as (rooted) *labelled directed graphs*, whose nodes and arcs represent respectively the states and the transitions of a system. On the other hand, any NDS may be unfolded into an *acyclic* graph. We shall here concentrate on a class of acyclic NDS's that we call *nondeterministic processes* (NDP's).

Basically, NDP's are acyclic NDS's – that is, NDS's whose derivation relation $\longrightarrow^*$ is a *partial ordering* – where each state is assigned a *label.* The label of a state represents the sequence of observable actions leading from the root to that state. To make such a labelling consistent, we only allow two paths to join in the graph if they correspond to the same observable derivation sequence. The labelling is subject to the following further restriction: for any label σ, there are at most finitely many states labelled by σ. As it will be made clear subsequently, this amounts to impose a general *image-finiteness* condition on the systems, and is a crucial hypothesis for some of our results.

In the formal definition, we will use the following notation: the *covering relation* $\multimap$ associated with a partial ordering $\leq$ is given by: $x \multimap y$ iff $x < y$ and $\nexists z$ such that $x < z < y$. A^* is the set of finite sequences over A, with the usual prefix-ordering and with empty sequence ε. In fact, since we are interested in observable sequences, we would like τ to be replaced by ε when occurring in strings. Let then

$$A^{\odot} = A^* \text{ modulo the law } \tau = \varepsilon$$

be the set of observable sequences over A.

Definition 2.3 *A nondeterministic process (NDP) over A is a triple $P = (Q \cup \{r\}, \leq, l)$ where:*

$(Q \cup \{r\}, \leq)$ *is a rooted poset of states:* $\forall\ q,\ r \leq q$

$l : Q \cup \{r\} \longrightarrow A^{\odot}$ *is a monotonic labelling function, satisfying:*

$l(r) = \varepsilon$

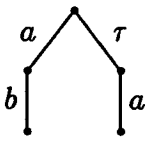
$q \multimap q'$ *implies* $\exists\, \mu \in A$ s.t. $l(q') = l(q)\mu$

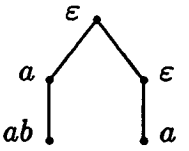
$\forall \sigma \in A^{\odot}$, *the set* $\{q \mid l(q) = \sigma\}$ *is* finite

It follows from our conditions and the requirement that l is a *monotonic* function (w.r.t. prefix–ordering) that the ordering $\leq$ on states is *discrete.* Note that an NDP is very nearly a *labelled tree*: it only differs from a labelled tree in that it may have some confluent paths. The reason we do not directly adopt labelled trees as a model is purely technical (this point will be discussed in section 2.7).

As pointed out already, we label nodes with sequences of actions, rather than labelling arcs with single actions: this minor variation w.r.t. the standard notation (see e.g. Milner's *synchronisation trees*) will make it easier to compare different states of a process.

It is easy to see that any NDP P generates an NDS, with $\xrightarrow{A}_P$ given by $\multimap\subset$. More precisely, for any $\mu \in A$, the relation $\xrightarrow{\mu}_P$ is given by $\{(q,q') \mid q \multimap\subset q' \text{ and } l(q') = l(q)\mu\}$. Note that, because of our law: $\tau = \varepsilon$, a transition $\xrightarrow{\tau}$ is represented in an NDP by the repetition of the same label on the two $\xrightarrow{\tau}$ related nodes. More generally, the label of a node will now represent the sequence of *observable* actions leading to it. For example:

the tree 

will be in our notation 

In what follows, nondeterministic processes will always be considered up to isomorphism (equality up to a renaming of states). We shall thus use $P_1 = P_2$ to mean that P_1 is isomorphic to P_2.

We proceed now to define the operators NIL, μ: and $+$ on NDP's. Let P_1, P_2 be NDP's, with $P_i = (Q_i \cup \{r_i\}, \leq_i, l_i)$. We have the following:

Definition 2.4 *(Operators on NDP's)*

NIL is the NDP with just a root r_{NIL}

$\mu : P_1$ is the NDP $P = (Q \cup \{r\}, \leq, l)$, where $r \notin (Q_1 \cup \{r_1\})$ and:

$$Q = Q_1 \cup \{r_1\}$$

$$\leq = \leq_1 \cup \{(r,q) \mid q \in Q\}$$

$$l(q) = \begin{cases} \varepsilon, & \text{if } q = r \\ \mu l_1(q), & \text{otherwise} \end{cases}$$

$P_1 + P_2$ is the NDP $P = (Q \cup \{r\}, \leq, l)$, where $r \notin (Q_1 \cup Q_2)$, and:

$$Q = Q_1 + Q_2 \quad \text{(disjoint union)}$$

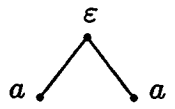
$$\leq = \leq_1 \restriction Q_1 \cup \leq_2 \restriction Q_2 \cup \{(r,q) \mid q \in Q\}$$

$$l = l_1 \restriction Q_1 \cup l_2 \restriction Q_2 \cup \{(r,\varepsilon)\}$$

Let $\mathcal{P} \subseteq \mathcal{S}$ denote the class of all NDP's: in what follows our treatment of nondeterministic systems will be confined to $\mathcal{P}$.

2.3.1 Abstraction homomorphisms

The NDP-model, though providing a helpful conceptual simplification, does not appear yet abstract enough. It still allows, for example, for structural redundancies such as:



Moreover we want to be able to ignore *unobservable* transitions. Such transitions, being internal to a system, should only be detectable indirectly, on account of their ability to affect the *observable behaviour* of the system.

Our intention is to define an abstraction mechanism directly on the model. To this purpose we introduce a *simplification* operation on processes, which we call *abstraction homomorphism.* Essentially an abstraction homomorphism will transform a process in a structurally simpler (but semantically equivalent) process by merging together some of its states.

The criterion for identifying states is that they be *equivalent* in some recursive sense: informally speaking, two states will be equivalent iff they have *equivalent histories* (derivation sequences) and *equivalent futures* or potentials (sets of subsequent states). Formally:

Definition 2.5 *If* $P_1 = (Q_1 \cup \{r_1\}, \leq_1, l_1)$, $P_2 = (Q_2 \cup \{r_2\}, \leq_2, l_2)$ *are NDP's, a function* $h : \begin{matrix} r_1 \longmapsto r_2 \\ Q_1 \longrightarrow Q_2 \end{matrix}$ *is an abstraction homomorphism (a.h.) from* P_1 *to* P_2 *iff for any* $q \in Q_1$ *:*

i) $l_2(h(q)) = l_1(q)$

ii) $succ_2(h(q)) = h(succ_1(q))$

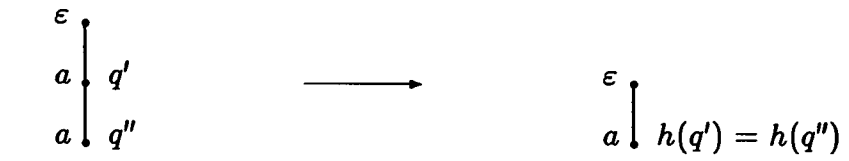
where $succ(q) = \{q' \mid q \leq q'\}$ is the set of successors of q, *including* q.

The notation we use to define h may look slightly odd: what it means is that h maps the root and only the root of P_1 into the root of P_2. Before giving

examples, let us also remark that any a.h. is *surjective* (instance of ii)) and preserves the ordering $\leq$ (again by ii)).

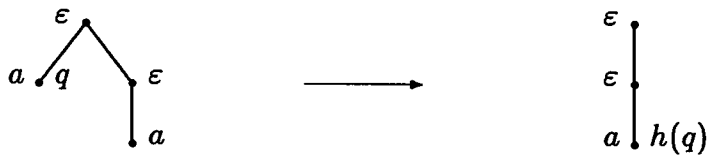
Examples

1)



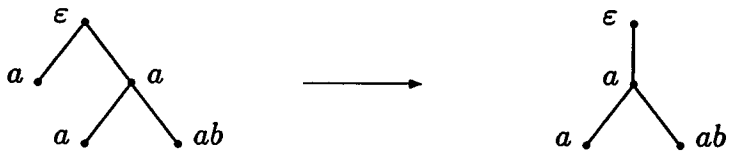
This example motivates our definition of succ(q): we want to allow q'', a *proper* successor of q', to be mapped to $h(q'') = h(q')$.

2)



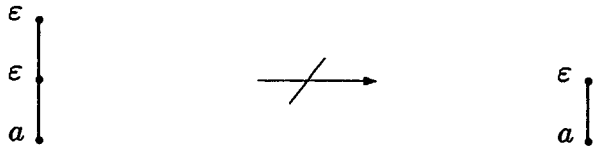
Note that the set of predecessors of q is *not* preserved by the homomorphism, and that the covering relation is not necessarily preserved.

3)



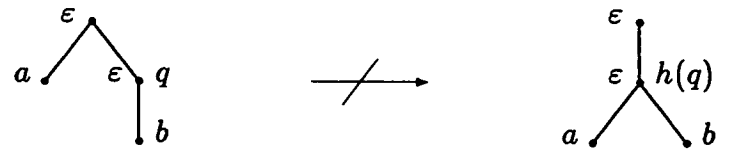
Counterexamples

4)



There is no a.h. here, since by definition an a.h. never maps a proper state into a root. As a consequence, a process of the form τP can only be transformed

into a process of the same form.

5) 

Again, there is no a.h. here, because the set of successors of q would increase.

Abstraction homomorphisms may be proven to satisfy some expected properties of homomorphisms: it can be easily checked, for example, that the composition of two a.h.'s is again an a.h.. We shall also see, later in this section, that any a.h. on an NDP P induces a *congruence* on the states of P. As for now, we want to make sure that our notion of abstraction homomorphism is compatible with the ordinary notion of isomorphism. We noted already that any a.h. is surjective. In fact, it is the case that:

Fact 2.6 *Any injective a.h. from P_1 to P_2 is an isomorphism between P_1 and P_2 and vice versa.*

Proof: Let h be an injective a.h. from P_1 to P_2. To prove that h is an isomorphism between P_1 and P_2, it is sufficient to show that:

$$h(q) \leq_2 h(q') \text{ only if } q \leq_1 q'$$

as the other properties are trivially implied by those of a.h.'s.

So suppose $h(q) \leq_2 h(q')$, i.e. $h(q') \in succ_2(h(q))$. Then we have also $h(q') \in h(succ_1(q))$, by prop. ii) of h. Therefore $\exists\, q'' \in succ_1(q)$ s.t. $h(q'') = h(q')$. Since h is injective, it can only be $q'' = q'$, whence $q' \in succ_1(q)$, *i.e.* $q \leq_1 q'$.

We have thus proved that an injective a.h. is an isomorphism. The proof of the converse is trivial, and is therefore omitted. □

For h an a.h. from P_1 to P_2, we shall often write $h : P_1 \longrightarrow P_2$.

Abstraction homomorphisms induce the following *reduction relation* $\xrightarrow{abs}$ on processes:

Definition 2.7 $P_1 \xrightarrow{abs} P_2$ *iff* $\exists$ *a.h.* $h : P_1 \longrightarrow P_2$.

We prove next a few properties of this relation.

Property 2.8 *The relation* $\stackrel{abs}{\longrightarrow}$ *is reflexive, transitive and antisymmetric (up to isomorphism).*

Proof: Reflexivity and transitivity are easy to check. We prove here that $\stackrel{abs}{\longrightarrow}$ is antisymmetric, namely that: if $h : P_1 \longrightarrow P_2$ and $h' : P_2 \longrightarrow P_1$ are a.h.'s, then $P_1 = P_2$.

For any NDP $P = (Q \cup \{r\}, \leq, l)$ and for any $\sigma \in A^*$, let:

$$Q_\sigma = \{q \mid q \in (Q \cup \{r\}), l(q) = \sigma\}.$$

Note that, because of our finiteness restriction on l, any such Q_σ is finite. Let now $P_1 = (Q_1 \cup \{r_1\}, \leq_1, l_1)$, $P_2 = (Q_2 \cup \{r_2\}, \leq_2, l_2)$, and $h : P_1 \longrightarrow P_2$, $h' : P_2 \longrightarrow P_1$.

For any $\sigma \in A^*$, define $h_\sigma = h \restriction Q_{1\sigma}$, $h'_\sigma = h' \restriction Q_{2\sigma}$. We then have:

$$h_\sigma : Q_{1\sigma} \longrightarrow Q_{2\sigma} \text{ surjectively, whence } |Q_{1\sigma}| \geq |Q_{2\sigma}|.$$

$$h'_\sigma : Q_{2\sigma} \longrightarrow Q_{1\sigma} \text{ surjectively, whence } |Q_{2\sigma}| \geq |Q_{1\sigma}|.$$

Summing up, we have $|Q_{1\sigma}| = |Q_{2\sigma}| < \infty$. Therefore the function h_σ is injective and thus also $h = \bigcup_{\sigma \in A^*} h_\sigma$ is injective. By fact 2.6 we can then conclude that h is an isomorphism between P_1 and P_2. □

Property 2.9 *The relation* $\stackrel{abs}{\longrightarrow}$ *is preserved by the operators* μ*: and* $+$.

Proof: Let $P_1 = (Q_1 \cup \{r_1\}, \leq_1, l_1), P_2 = (Q_2 \cup \{r_2\}, \leq_2, l_2)$ be NDP's, and $h : P_1 \longrightarrow P_2$ be an a.h.. We can then deduce that:

1) $\mu P_1 \stackrel{abs}{\longrightarrow} \mu P_2, \quad \forall \mu \in A$

2) $P_1 + P \stackrel{abs}{\longrightarrow} P_2 + P, \quad \forall \text{ NDP } P$

Proof of 1): Let $P'_1 = \mu P_1$, $P'_2 = \mu P_2$, with sets of states $(Q'_1 \cup \{r'_1\})$ and $(Q'_2 \cup \{r'_2\})$ respectively. Then the function $f : (Q'_1 \cup \{r'_1\}) \longrightarrow (Q'_2 \cup \{r'_2\})$ defined by:

$$f(q) = \begin{cases} r'_2, & \text{if } q = \{r'_1\} \\ h(q), & \text{otherwise} \end{cases}$$

is (trivially) an a.h. from P_1' to P_2'.

Proof of 2): Let $P_1' = P_1 + P$, $P_2' = P_2 + P$, with sets of states $(Q_1' \cup \{r_1'\})$ and $(Q_2' \cup \{r_2'\})$ respectively. Let $(Q \cup \{r\})$ denote the states of P.

Then the function $f : (Q_1' \cup \{r_1'\}) \longrightarrow (Q_2' \cup \{r_2'\})$ defined by:

$$f(q) = \begin{cases} r_2', & \text{if } q = r_1' \\ h(q), & \text{if } q \in Q_1 \\ q, & \text{if } q \in Q \end{cases}$$

is (trivially) an a.h. from P_1' to P_2'. □

In what follows, a relation which is preserved by our operators will often be called *substitutive* (w.r.t. these operators).

We turn now to what is perhaps the most interesting feature of our reduction relation, namely its *confluent* behaviour. Confluence of a.h.'s can be proved by standard algebraic techniques, once the notion of congruence associated to an a.h. is formalised.

Definition 2.10 *If $P = (Q \cup \{r\}, \leq, l)$ is an NDP, we say that an equivalence relation $\sim$ on Q is a congruence on P if and only if, whenever $q \sim q'$:*

i) $l(q) = l(q')$ (labels are preserved)

ii) $q < p \implies \exists\, p' \sim p$ s.t. $q' \leq p'$ (successors are preserved)

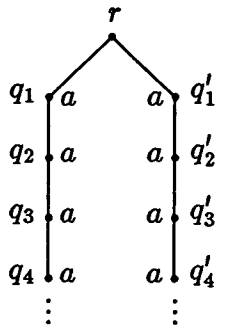
It can be proved that any congruence $\sim$ satisfies the following:

convexity property: $q < p < q'$ and $q \sim q'$ implies $q \sim p \sim q'$

The proof is by induction on the length n of the longest chain: $q' \mathrel{-\!\subset} q_1 \mathrel{-\!\subset} \cdots \cdots \mathrel{-\!\subset} q_n$ s.t. $l(q') = l(q_1) = \cdots = l(q_n)$. That this length is finite is ensured by our finiteness restriction on the labelling l. In fact, in absence of this restriction, the convexity property would not hold, as shown by the following example.

Example

The figure below shows a process P, whose labelling does *not* satisfy the finiteness property, and a congruence $\sim$ on P:

$P =$ 

with $\sim$ given by:

$$\begin{array}{ccccccc} q_1 & \sim & q_1' & \sim & q_3 & \sim & q_3' \sim \cdots \\ \wedge & & \wedge & & \wedge & & \wedge \\ q_2 & \sim & q_2' & \sim & q_4 & \sim & q_4' \sim \cdots \end{array}$$

The equivalence $\sim$ can be easily checked to be a congruence (the ordering symbols are inserted in the figure to illustrate prop. i) of congruences). However, $\sim$ does *not* satisfy the convexity property, since for example:

$$q_1 < q_2 < q_3 \quad \text{and} \quad q_1 \sim q_3, \quad \text{but} \quad q_1 \not\sim q_2,\ q_2 \not\sim q_3$$

We show now that, for any NDP P, there is a one-to-one correspondence between congruences and abstraction homomorphisms on P. First, some notation: If $P = (Q \cup \{r\}, \leq, l)$ is an NDP and h an a.h. on P, we define the equivalence $\sim_h$ on Q by:

$$\sim_h = \{(q, q') \mid q, q' \in Q,\ h(q) = h(q')\}$$

We can then prove the following two theorems:

Theorem 2.11 *If P is an NDP and $\sim$ is a congruence on P, then there exists an NDP $P/\sim$, the quotient of P by $\sim$, and an a.h. $h_\sim$ from P to $P/\sim$ s.t. $\sim_{h_\sim} = \sim$.*

Proof: If $P = (Q \cup \{r\}, \leq, l)$, define $P/\sim = (Q/\sim \cup \{r'\}, \leq', l')$ by:

$r' <' [q]\,, \quad \forall q \in Q$

$[q] \leq' [p]$ iff $\exists\, p'$ s.t. $q \leq p' \sim p$

$l'(r') = \varepsilon$

$l'([q]) = l(q)$

Also, define $h_\sim : (Q \cup \{r\}) \longrightarrow (Q/\sim \cup \{r'\})$ by:

$$h_\sim(r) = r'$$

$$h_\sim(q) = [q]\,, \quad \forall q \in Q$$

We shall then prove that:

1) $P/\sim$ is an NDP

2) $h_\sim$ is an a.h. from P to $P/\sim$.

Proof of 1): To prove that $P/\sim$ is an NDP:

First, we have to check that $\leq'$ is a partial ordering relation. Reflexivity and transitivity follow easily from the definition. To prove antisymmetry, use the convexity property of $\sim$.

Second, we show that the labelling l' meets the requirements. The property of finiteness can be easily deduced from the same property of the labelling l. We prove here that $[q] -\!\subset [p]$ implies $l'([p]) = l'([q])\mu$ for some $\mu \in A$. In fact, suppose $[q] -\!\subset [p]$: this is because $q < p' \sim p$, for some p'. That is, $\exists\, p_0, \ldots, p_n$, $n \geq 1$, s.t. $q = p_0 -\!\subset \ldots -\!\subset p_n = p'$. Now it can be easily shown, by induction on $n \geq 1$, that:

$p_0 -\!\subset \cdots -\!\subset p_n$ and $[p_0] -\!\subset [p_n]$ implies $\exists\, p_0', p_n'$ s.t. $p_0 \sim p_0' -\!\subset p_n' \sim p_n$

So, from $[q] -\!\subset [p']$ we deduce: $\exists\, q', p''$ s.t. $q \sim q' -\!\subset p'' \sim p'$. Then: $l'([p]) = l(p'') = l(q')\mu = l'([q])\mu$. This ends the proof of 1).

Proof of 2): We want to show that $h_\sim$ is an a.h. from P to $P/\sim$, and that $\sim_{h_\sim} = \sim$. By definition $h_\sim$ is a function mapping r to r' and Q to $Q/\sim$.

We check now the properties i) and ii) of a.h.'s.

Property i):

$$l'(h_\sim(r)) = l'(r') = \varepsilon = l(r)$$

$$l'(h_\sim(q)) = l'([q]) = l(q), \text{ for } q \in Q$$

Property ii):

$$\begin{aligned} succ(h_\sim(r)) &= succ(r') = (Q/\sim) \cup \{r'\} = h_\sim(Q \cup \{r\}) = h_\sim(succ(r)) \\ succ(h_\sim(q)) &= succ([q]) = \{[p] \mid [q] \leq' [p]\} = \{[p'] \mid q \leq p'\} \\ &= h_\sim(\{p' \mid q \leq p'\}) = h_\sim(succ(q)) \end{aligned}$$

So $h_\sim$ is indeed an a.h. from P to $P/\sim$. □

Theorem 2.12 *If P, P' are NDP's and h is an a.h. from P to P', then $\sim_h$ is a congruence on P and P' is isomorphic to $P/\sim_h$.*

Proof: Again, we show the result in two steps:

1) $\sim_h$ is a congruence on P

2) P' is isomorphic to $P/\sim_h$

Proof of 1): We know that $\sim_h$ is an equivalence relation on Q. We check that it satisfies the properties i) and ii) of congruences. Suppose $q \sim_h q'$: this is because $h(q) = h(q')$. Therefore we have:

Property i): $l(q) = l'(h(q)) = l'(h(q')) = l(q')$

Property ii): $q \leq p$ means $p \in succ(q)$. Then $h(p) \in h(succ(q)) =$
$= succ(h(q)) = succ(h(q')) = h(succ(q'))$. So $\exists\, p' \in succ(q')$
s.t. $h(p) = h(p')$. That is, $\exists\, p'$ s.t. $q' \leq p'$ and $p \sim_h p'$.

Proof of 2): If $P' = (Q' \cup \{r'\}, \leq', l')$ and $P/\sim_h = (Q/\sim_h \cup \{r''\}, \leq'', l'')$ is defined as for theorem 2.11 , let $\Phi : (Q/\sim_h \cup \{r''\}) \longrightarrow (Q' \cup \{r'\})$ be the function given by:

$$\Phi(r'') = r'$$

$$\Phi([q]) = h(q)$$

Then it is clear that Φ is well-defined. We show now that Φ is an injective a.h. from $P/\sim_h$ to P'. It will then follow, by fact 2.6, that Φ is an isomorphism between $P/\sim_h$ and P'.

It is easy to check that Φ is injective, as:

$$h(q) = h(p) \text{ implies } [q] = [p]$$

Moreover, Φ satisfies the properties i) and ii) of a.h.'s:

$$\begin{aligned}
\text{Property i):}\quad & l'(\Phi(r'')) = l'(r') = \varepsilon = l''(r'') \\
& l'(\Phi([q])) = l'(h(q)) = l(q) = l''([q]) \\
\text{Property ii):}\quad & succ(\Phi([q]) = succ(h(q)) = h(succ(q)) = \\
& = \{h(p) \mid q \leq p\} = \{\Phi([p]) \mid q \leq p\} = \\
& = \{\Phi([p']) \mid q \leq p \sim_h p'\} = \Phi(succ([q])
\end{aligned}$$

□

To prove the confluence property of a.h.'s, we will finally make use of the following:

Lemma 2.13 *If $\sim_1, \sim_2$ are congruences on an NDP P, then the relation $\sim_{1,2} =_{def} [\ \sim_1 \cup \sim_2\ ]^*$, the symmetric and transitive closure of the union of $\sim_1$ and $\sim_2$, is the least congruence $\sim$ on P s.t. $\sim_1 \subseteq \sim$ and $\sim_2 \subseteq \sim$.*

Proof: It is a standard result that $\sim_{1,2}$ is the least equivalence on Q which includes both $\sim_1$ and $\sim_2$. Then, if $\sim_{1,2}$ is a congruence, it will also be the *least* congruence which includes $\sim_1$ and $\sim_2$.

Thus all we have to show is that $\sim_{1,2}$ is a congruence, namely that it satisfies the required properties i), ii). Let $\sim_{1|2} = \sim_1 \cup \sim_2$. Now $q \sim_{1,2} q'$ iff $\exists n, \exists q_0, \ldots, q_n$ s.t.

$$q = q_0 \sim_{1|2} \cdots \sim_{1|2} q_n = q'$$

Then property i) of congruences is easy to check. As for property ii), suppose $q \leq p$. Since both $\sim_1$ and $\sim_2$ satisfy ii), there exist $p_0, \ldots, p_n$ s.t. $q_i \leq p_i$ and:

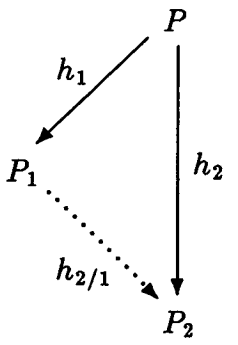
$$p = p_0 \sim_{1|2} \cdots \sim_{1|2} p_n$$

Thus, if we let $p' = p_n$, we have $p \sim_{1,2} p'$, and $q' \leq p'$. □

For the coming theorems, we will need some more notation. If h, h' are two a.h.'s on the same process, we say that h is *weaker* than h', and write $h \preceq h'$, iff $\sim_h \subseteq \sim_{h'}$. The following fact is then (almost) standard:

Lemma 2.14 *(Factorisation of an a.h. by a weaker one)*

If P, P_1, P_2 are NDP's, and $h_1 : P \longrightarrow P_1$, $h_2 : P \longrightarrow P_2$ are a.h.'s s.t. $h_1 \preceq h_2$, then there exists a unique a.h. $h_{2/1} : P_1 \longrightarrow P_2$ s.t. the following diagram commutes:



Proof: Let $\sim_1, \sim_2$ stand for $\sim_{h_1}, \sim_{h_2}$. In view of theorem 2.12, we can assume:

$$P_1 = P/\sim_1, P_2 = P/\sim_2$$

Then the unique mapping $h_{2/1}$ that can make the diagram commute is the one defined by:

$$h_{2/1}(r_1) = r_2$$

$$h_{2/1}([q]_1) = h_2(q) = [q]_2, \quad \forall q \in Q$$

This mapping is a function, because $[q]_1 = [q']_1$ implies $[q]_2 = [q']_2$ for the hypothesis that $\sim_1 \subseteq \sim_2$. We now show that $h_{2/1}$ is an a.h.. Let as usual l_i and $succ_i$ refer to P_i . Then $h_{2/1}$ satisfies:

i) $l_1([q]_1) = l(q) = l_2([q]_2) = l_2(h_{2/1}([q]_1))$

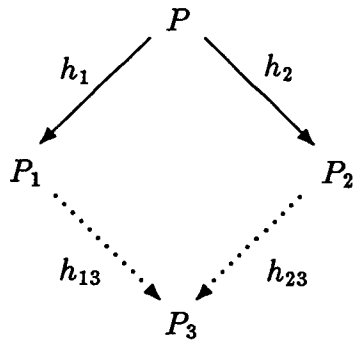
ii) $$\begin{aligned} h_{2/1}(succ_1([q]_1)) &= h_{2/1}(succ_1(h_1(q))) = h_{2/1}(h_1(succ_1(q))) = \\ &= h_{2/1}([q']_1 \mid q \leq q') = h_2(q' \mid q \leq q') = \\ &= h_2(succ_1(q)) = succ_2(h_2(q)) = succ_2(h_{2/1}([q]_1)) \end{aligned}$$

□

We can finally prove the result:

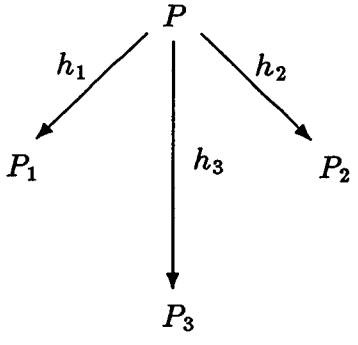
Theorem 2.15 *(Confluence of abstraction homomorphisms)*

If P, P_1, P_2 *are NDP's, and* $h_1 : P \longrightarrow P_1$, $h_2 : P \longrightarrow P_2$ *are a.h.'s, then there exists an NDP* P_3 *and two unique a.h.'s* $h_{13} : P_1 \longrightarrow P_3$, $h_{23} : P_2 \longrightarrow P_3$ *s.t. the following diagram commutes:*



Proof: Let again $\sim_1$ and $\sim_2$ stand for $\sim_{h_1}$ and $\sim_{h_2}$. Define $\sim_3 = [\ \sim_1 \cup \sim_2\ ]^*$. Since $\sim_3$ is a congruence (by lemma 2.13), there exist correspondingly an NDP $P/\sim_3$ and an a.h. $h_{\sim_3} : P \longrightarrow P/\sim_3$ (by theorem 2.12).

Let now P_3 be $P/\sim_3$ and $h_3 = h_{\sim_3}$. We have the following situation:



where both the pairs (h_1, h_3) and (h_2, h_3) meet the hypothesis of lemma 2.14. Whence the result with $h_{13} = h_{3/1}$, $h_{23} = h_{3/2}$. □

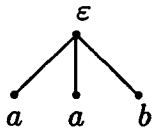
Convention: in the following we will use $\xleftarrow{abs}$ instead of $\xrightarrow{abs}{}^{-1}$ whenever convenient. We conclude this section by stating the following:

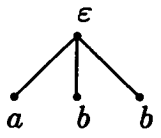
Corollary 2.16 ($\xrightarrow{abs}$ *is Church-Rosser*)

If P, P_1, P_2 *are NDP's s.t.* $P \xrightarrow{abs} P_1$ *and* $P \xrightarrow{abs} P_2$ *, then there exists an NDP* P_3 *s.t.* $P_1 \xrightarrow{abs} P_3$ *and* $P_2 \xrightarrow{abs} P_3$ □

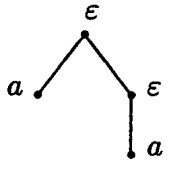
2.3.2 Abstraction equivalence

The relation $\xrightarrow{abs}$ gives us a criterion to regard two processes as "abstractly the same". However, being essentially a simplification, $\xrightarrow{abs}$ is not symmetric and does not, for example, relate the two processes:





or the processes:



Based on $\xrightarrow{abs}$, we will then define on NDP's a more general relation $\sim_{abs}$, of *reducibility* to a same process:

Definition 2.17 $\sim_{abs} =_{def} \xrightarrow{abs} \cdot \xleftarrow{abs}$

We can immediately prove a few properties for $\sim_{abs}$.

Property 2.18 $\sim_{abs}$ *is an equivalence relation.*

Proof: Transitivity follows from the fact that $\xrightarrow{abs}$ is Church-Rosser, which can be restated as: $[\ \xrightarrow{abs} \cup \xleftarrow{abs}\ ]^* = \xrightarrow{abs} \cdot \xleftarrow{abs}$ □

Property 2.19 $\sim_{abs}$ *is preserved by the operators* μ*:* *and* $+$.

Proof: Direct consequence of the substitutivity of $\xrightarrow{abs}$ under μ: and $+$. □

To sum up, we have now a *substitutive* equivalence $\sim_{abs}$ for NDP's that can be split, when required, in two reduction halves. The equivalence $\sim_{abs}$ will be called *abstraction equivalence.* In the coming section we shall study how abstraction equivalence relates to bisimulation equivalence.

2.4 Relating bisimulations to abstraction homomorphisms

2.4.1 Weak bisimulation relations

In section 2.1 we gave the definition of bisimulation for general LTS's. There the criterion for testing a system and selecting its subsystems was simply given by the transition relation. In the case of NDS's, however, where one wants to abstract from internal transitions, a weaker criterion is needed. To this purpose, Milner introduced the following *weak* transition relations $\stackrel{\mu}{\Longrightarrow}$:

$$\stackrel{a}{\Longrightarrow} = \stackrel{\tau^n}{\longrightarrow}\stackrel{a}{\longrightarrow}\stackrel{\tau^m}{\longrightarrow} \qquad n,m \geq 0,\ a \in A - \{\tau\}$$

$$\stackrel{\tau}{\Longrightarrow} = \stackrel{\tau^n}{\longrightarrow} \qquad n \geq 0$$

In terms of these new transitions, one can then define *weak bisimulation* relations on NDS's as follows:

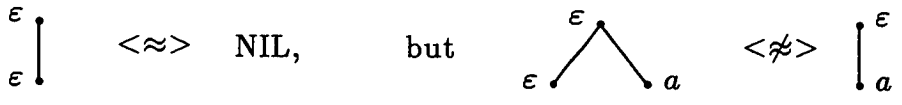
Definition 2.20 *A weak bisimulation is a relation $R \subseteq (S \times S)$ s.t. $R \subseteq F(R)$, where $(S_1, S_2) \in F(R)$ iff $\forall \mu \in A$:*

i) $S_1 \stackrel{\mu}{\Longrightarrow} S_1'$ *implies* $\exists\, S_2'$ *s.t.* $S_2 \stackrel{\mu}{\Longrightarrow} S_2'$, *with* $S_1'\, R\, S_2'$

ii) $S_2 \stackrel{\mu}{\Longrightarrow} S_2'$ *implies* $\exists\, S_1'$ *s.t.* $S_1 \stackrel{\mu}{\Longrightarrow} S_1'$, *with* $S_1'\, R\, S_2'$

Again, the equation $R \subseteq F(R_a)$ has a *largest solution* which is an equivalence relation, and will be denoted by $<\approx>$. This equivalence is generally referred to as the *weak bisimulation equivalence*. We shall mostly omit the "weak" in what follows, since we will never use the simple bisimulation $<\sim>$ on NDS's.

Now, it is well-known that $<\approx>$ is not preserved by the operator $+$, as shown by the classical example:



On the other hand the relation $<\approx>^+$, obtained by closing $<\approx>$ w.r.t. the operator $+$:

$$S_1 <\approx>^+ S_2 \text{ iff } \forall S:\ (S + S_1) <\approx> (S + S_2)$$

can be shown to be a *substitutive* equivalence, and in fact to be the largest such equivalence contained in $<\approx>$. (For more details on $<\approx>$ and $<\approx>^+$ we refer to [Mil 83] and [Mil 84]).

To conclude, $<\approx>^+$ seems a convenient restriction on $<\approx>$ to adopt when modelling NDS's. We will see in the next section that $<\approx>^+$ coincides, on NDP's, with our abstraction equivalence $\sim_{abs}$.

2.4.2 Quasi-bisimulations

Looking back at our relations $\stackrel{abs}{\longrightarrow}$ and $\sim_{abs}$, we notice that they rely on a notion of *equivalence of states* which, like bisimulations, is *recursive.* Moreover, the recursion builds up on the basis of a similarity requirement (equality of *labels*) that reminds of the criterion (equality of *observable derivation sequences*) used in (weak) bisimulations to derive corresponding subsystems. All this indicates there might be a close analogy between abstraction equivalence and the bisimulation equivalence $<\approx>$.

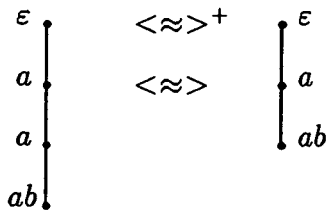
In fact, since we know that $\sim_{abs}$ is substitutive, we shall try to relate it with the substitutive closure $<\approx>^+$ of $<\approx>$. To this purpose, it will be convenient to have $<\approx>^+$ itself be defined recursively.

Note that $<\approx>^+$ only differs from $<\approx>$ in that it takes into account the *preemptive capacities* a system can develop when placed in a sum-context. Such preemptive capacities depend on the system having some silently reachable state where, informally speaking, some of the "alternatives" offered by the sum-context are no longer available. This suggests that we should adopt, when looking for a direct definition of $<\approx>^+$, the more *restrictive* transition *relations* $\stackrel{\mu}{\Longmapsto}$:

$$\stackrel{\mu}{\Longmapsto} =_{def} \stackrel{\tau^n}{\longrightarrow} \stackrel{\mu}{\longrightarrow} \stackrel{\tau^m}{\longrightarrow} \quad n, m \geq 0$$

In particular, we will have $\stackrel{\tau}{\Longmapsto} = \stackrel{\tau^n}{\longrightarrow}, n > 0$. Note on the other hand that, for $a \in A - \{\tau\}$, it will be: $\stackrel{a}{\Longmapsto} = \stackrel{a}{\Longrightarrow}$.

However, the equivalence $<\approx>^+$ is *restrictive* with respect to $<\approx>$ only as far as the first $\stackrel{\tau}{\Longrightarrow}$ derivation steps are concerned: at further steps $<\approx>^+$ behaves like $<\approx>$, as it can be seen from the example:



So, if we are to recursively define $<\approx>^+$ in terms of the transitions $\stackrel{\mu}{\Longmapsto}$, we will have to somehow counteract the strengthening effect of the $\stackrel{\mu}{\Longmapsto}$'s at steps other than the first.

To this end, for any relation $R \subseteq (S \times S)$, a relation R_a ("almost" R) is introduced:

$$(S_1, S_2) \in R_a \text{ iff } (S_1, S_2) \in R, \text{or } (\tau : S_1, S_2) \in R, \text{or } (S_1, \tau : S_2) \in R.$$

We can then define quasi-bisimulation relations on NDS's as follows (the definition is due to M. Hennessy):

Definition 2.21 *A (weak) quasi-bisimulation is a relation $R \subseteq (S \times S)$ such that $R \subseteq E(R_a)$, where $(S_1, S_2) \in E(R_a)$ iff $\forall \mu \in A$:*

i) $S_1 \stackrel{\mu}{\Longmapsto} S_1'$ *implies* $\exists\, S_2'$ *s.t.* $S_2 \stackrel{\mu}{\Longmapsto} S_2'$, *with* $S_1' \, R_a \, S_2'$

ii) $S_2 \stackrel{\mu}{\Longmapsto} S_2'$ *implies* $\exists\, S_1'$ *s.t.* $S_1 \stackrel{\mu}{\Longmapsto} S_1'$, *with* $S_1' \, R_a \, S_2'$

Once more, the equation $R \subseteq E(R_a)$ has a *largest solution* which is an equivalence relation. Now this equivalence has been proven by M. Hennessy to *coincide* with $<\approx>^+$. From now on, we shall use this as our definition for $<\approx>^+$.

Before proceeding, let us make a few remarks about the relationship of quasi-bisimulations with ordinary bisimulations.

To this purpose, we will need some more notation. For any NDP P, let $Der(P) = \{P' \mid P \longrightarrow^* P'\}$. If R is a binary relation , $R \subseteq (X \times Y)$, we say that R is *total* if $\{x \mid (x, y) \in R\} = X$, and write $Dom(R)$ for $\{x \mid (x, y) \in R\}$.

Let us now introduce the following definition:

Definition 2.22 *(First step condition) If R is a bisimulation s.t. $(P_1, P_2) \in R$, we say that R satisfies the first step condition on (P_1, P_2) iff $\forall \mu \in A$:*

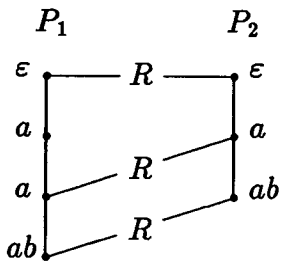
i) $P_1 \stackrel{\mu}{\Longmapsto} P_1'$ *implies* $\exists P_2'$ *s.t.* $P_2 \stackrel{\mu}{\Longmapsto} P_2'$, *with* $P_1' \, R \, P_2'$

ii) $P_2 \stackrel{\mu}{\Longmapsto} P_2'$ *implies* $\exists P_1'$ *s.t.* $P_1 \stackrel{\mu}{\Longmapsto} P_1'$, *with* $P_1' \, R \, P_2'$

We then have the following characterisation:

Proposition 2.23 *A bisimulation R is a quasi-bisimulation if and only if, whenever $(P_1, P_2) \in R$, then R satisfies the first step condition on (P_1, P_2).*

Conversely a quasi-bisimulation is not, in general, a bisimulation, as illustrated by the following example. Let $P_1 = a\tau b\text{NIL}$, $P_2 = ab\text{NIL}$. Then the relation R as shown in the figure:



is a quasi-bisimulation, but not a bisimulation. Note on the other hand that in this example the quasi-bisimulation R could be extended to a bisimulation by adding the pair $(\tau a\text{NIL}, a\text{NIL})$ to it, i.e. by turning it into a total relation. We have in fact the following:

Proposition 2.24 *A quasi-bisimulation R is a bisimulation if and only if, whenever $(P_1, P_2) \in R$, then R is total on $Der(P_1)$ and R^{-1} is total on $Der(P_2)$.*

To end this digression, let us just add that, if R is a quasi-bisimulation, then R_a is an ordinary bisimulation. In particular, for the maximal quasi-bisimulation $<\approx>^+$, we have $<\approx>_a^+ = <\approx>$ (this fact will be used in the proof of theorem 2.31 below).

2.4.3 Characterisation of abstraction homomorphisms

We proceed in this section to compare $\sim_{abs}$ with $<\approx>^+$. We shall see first that our reduction relation $\stackrel{abs}{\longrightarrow}$ is a quasi-bisimulation. We come then to a characterisation of abstraction homomorphisms as particular bisimulation relations. Only at that point will we have all the elements to prove our main result, namely that $\sim_{abs}$ is equal to the largest quasi-bisimulation, and therefore to $<\approx>^+$.

We thus start by showing that $\stackrel{abs}{\longrightarrow}$ is a quasi-bisimulation. Let us first redefine μ-derivatives in terms of the new relations $\stackrel{\mu}{\Longmapsto}$: we say that S' is a (proper) *μ-derivative of S* iff $S \stackrel{\mu}{\Longmapsto} S'$. We can then prove the following two lemmas:

Lemma 2.25 ($\stackrel{abs}{\longrightarrow}$ *almost preserves μ-derivatives*)

If $P_1 \stackrel{abs}{\longrightarrow} P_2$ *and* $P_1 \stackrel{\mu}{\Longmapsto} P_1'$ *then* $\exists P_2'$ *s.t.* $P_2 \stackrel{\mu}{\Longmapsto} P_2'$ *where either* $P_1' \stackrel{abs}{\longrightarrow} P_2'$ *or* $P_1' \stackrel{abs}{\longrightarrow} \tau P_2'$.

Proof: We recall that any state of an NDP P is the initial state $r_{P'}$ of some derivative NDP P'. Note that $P \stackrel{\mu}{\Longmapsto} P'$ implies $l(r_{P'}) = \mu$ in P.

Let now $h : P_1 \longrightarrow P_2$ be an a.h., and let as usual Q_i, r_i refer to $P_i, i = 1,2$.

Suppose $P_1 \stackrel{\mu}{\Longmapsto} P_1'$. Let r_1' be the initial state of P_1' in P_1: then $r_1' \in Q_1$ (i.e. $r_1' \neq r_1$) and $l_1(r_1') = \mu$. Let now $r_2' = h(r_1')$. Then r_2' is the root of some derivative P_2' of P_2. Since h is an a.h., we have $r_2' \neq r_2$ and $l(r_2') = l(r_1') = \mu$. Therefore $P_2 \stackrel{\mu}{\Longmapsto} P_2'$.

Let now $Q_i' = \{q \mid q \in P_i, r_i' < q\}$, $i = 1,2$. From prop. ii) of a.h.'s, we know that: $h(Q_1' \cup \{r_1'\}) = Q_2' \cup \{r_2'\}$.

Then we are in one of the following two cases:

1) $h(Q_1') = Q_2'$

2) $h(Q_1') = Q_2' \cup \{r_2'\}$

In case 1), $h \restriction (Q_1' \cup \{r_1'\})$ is trivially an a.h. from P_1' to P_2'. Therefore $P_1' \stackrel{abs}{\longrightarrow} P_2'$.

In case 2), h maps some states of Q_1' to r_2'. Note that these states will be labelled by μ in P_1 and by ε in P_1'. Let Q_ε denote the set of such states in P_1', namely $Q_\varepsilon = \{q \mid q \in Q_1', l(q) = \varepsilon\}$.

Also, let $P_2'' = \tau P_2'$. Then $Q_2'' = Q_2' \cup \{r_2'\}$ and the function: $h' : (Q_1' \cup \{r_1'\}) \longrightarrow (Q_2'' \cup \{r_2''\})$ defined by:

$$h'(r_1') = r_2''$$

$$h'(Q_e) = r_2'$$

$$h'(q) = h(q), \quad \text{otherwise}$$

is (trivially) an a.h., so that in this case we have: $P_1' \xrightarrow{abs} \tau P_2'$. □

Lemma 2.26 ($\xrightarrow{abs}{}^{-1}$ *almost preserves* μ*-derivatives*)

If $P_1 \xrightarrow{abs} P_2$ *and* $P_2 \stackrel{\mu}{\Longmapsto} P_2'$ *then* $\exists P_1'$ *such that* $P_1 \stackrel{\mu}{\Longmapsto} P_1'$, *where either* $P_1' \xrightarrow{abs} P_2'$ *or* $P_1' \xrightarrow{abs} \tau P_2'$.

Proof: Suppose $P_2 \stackrel{\mu}{\Longmapsto} P_2'$. Let r_2' be the initial state of P_2' in P_2. Since h is surjective, $\exists r_1' \neq r_1$ s.t. $h(r_1') = r_2'$, $l(r_1') = l(r_2') = \mu$. Then, if P_1' is the derivative of P_1 with root r_1', we have $P_1 \stackrel{\mu}{\Longmapsto} P_1'$, and the rest of the proof goes as for lemma 2.25. □

Theorem 2.27 $\xrightarrow{abs}$ *is a quasi-bisimulation relation on NDP's.*

Proof: Follows immediately from lemma's 2.25 and 2.26 □

Corollary 2.28 $\xrightarrow{abs} \subseteq <\approx>^+$

Proof: $<\approx>^+$ is the largest quasi-bisimulation. □

Note that in lemma's 2.25 and 2.26 we do not need consider the case $\tau P_1' \xrightarrow{abs} P_2'$. The reason this case does not arise is that a.h.'s are *single-valued* relations. In fact, our next task will be to characterise a.h.'s as relations on processes.

Let us first introduce some terminology. We defined earlier the set $Der(P)$ of derivatives of an NDP P. Let now $PDer(P) = \{P' \mid P \longrightarrow^* P', P \neq P'\}$ be the set of *proper* derivatives of P. Also, we say that a bisimulation relation R is *between* P_1 and P_2 if it relates P_1 and P_2 and is limited to their derivatives, namely if $P_1 \, R \, P_2$ and $R \subseteq [Der(P_1) \times Der(P_2)]$.

Note that a bisimulation R satisfies the first step condition on (P_1, P_2) if and only if $(P_1, P_2) \in R$ and $P_1 \notin R(PDer(P_2))$, $P_2 \notin R(PDer(P_1))$.

Let now $\stackrel{\sigma}{\Longrightarrow}$ stand for $\stackrel{\mu_1}{\Longrightarrow} \cdots \stackrel{\mu_n}{\Longrightarrow}$ if $\sigma = \mu_1 \cdots \mu_n$. It is easy to see that, for any two systems S_1, S_2, the following holds:

Lemma 2.29 *If $S_1 \, R \, S_2$ for some bisimulation R, then for any $\sigma \in A^{\odot}$:*

i) $S_1 \stackrel{\sigma}{\Longrightarrow} S_1'$ *implies* $\exists \, S_2'$ *s.t.* $S_2 \stackrel{\sigma}{\Longrightarrow} S_2'$, *with* $S_1' \, R \, S_2'$

ii) $S_2 \stackrel{\sigma}{\Longrightarrow} S_2'$ *implies* $\exists \, S_1'$ *s.t.* $S_1 \stackrel{\sigma}{\Longrightarrow} S_1'$, *with* $S_1' \, R \, S_2'$ □

Now, if we regard a.h.'s as relations on processes, we have the following characterisation:

Theorem 2.30 *A relation R on processes is an abstraction homomorphism from P_1 to P_2 iff R is a single-valued bisimulation between P_1 and P_2 s.t. $P_2 \notin R(PDer(P_1))$.*

Proof: **Only if:** Let R be an a.h. from P_1 to P_2. Again, we assume that any state of P_i is the root $r_{P_i'}$ of some derivative P_i'. Also, we write $P_1' \, R \, P_2'$ in place of $R(r_{P_1'}) = r_{P_2'}$ whenever we want to treat R as a relation on processes.

Now, R is by definition single-valued and such that $P_1 \, R \, P_2$ and $R \subseteq (Der(P_1) \times Der(P_2))$. Also, $R(PDer(P_1)) = PDer(P_2)$ implies that $P_2 \notin R(PDer(P_1))$. What is left to show is that R is a bisimulation, namely that $R \subseteq F(R)$. Suppose $P_1' \, R \, P_2'$, i.e. $R(r_{P_1'}) = r_{P_2'}$.

Clause i): If $P_1' \stackrel{\mu}{\Longrightarrow} P_1''$, we have: $r_{P_1''} \in succ(r_{P_1'})$ and $l(r_{P_1''}) = l(r_{P_1'})\mu$. Now, since $R(succ(r_{P_1'}) = succ(r_{P_2'})$ [prop.ii) of a.h.'s] , there exists $r_{P_2''} \in succ(r_{P_2'})$ s.t. $R(r_{P_1''}) = r_{P_2''}$, that is $P_1'' \, R \, P_2''$. Then $l(r_{P_2''}) = l(r_{P_1''}) = l(r_{P_1'})\mu = l(r_{P_2'})\mu$ [using prop.i) of a.h.'s] . So we have $P_2' \stackrel{\mu}{\Longrightarrow} P_2''$, with $P_1'' \, R \, P_2''$.

Clause ii): the proof is symmetric to that of clause i).

If: Let now R be a single-valued bisimulation between P_1 and P_2 s.t. $P_2 \notin R(PDer(P_1))$. Then R can be regarded as a function $R : (Q_1 \cup \{r_1\}) \longrightarrow (Q_2 \cup \{r_2\})$. By hypothesis we have: $R(r_1) = r_2, R(Q_1) = Q_2$. We check now that R satisfies properties i) and ii) of a.h.'s. Suppose $R(r_{P_1'}) = r_{P_2'}$, i.e. $P_1' \, R \, P_2'$.

Property i): $l(r_{P_1'}) = \sigma \in A^{\odot}$ means $P_1 \stackrel{\sigma}{\Longrightarrow} P_1'$. Since $P_1 \, R \, P_2$, we know, by lemma 2.25, that $\exists \, P_2''$ s.t. $P_2 \stackrel{\sigma}{\Longrightarrow} P_2''$ and $P_1' \, R \, P_2''$. Since R is single-valued, it must be $P_2'' = P_2'$. Whence $l(r_{P_2'}) = \sigma$.

Property ii): $r_{P_1''} \in succ(r_{P_1'})$ means that $\exists \, \sigma \in A^{\odot}$ s.t. $P_1' \stackrel{\sigma}{\Longrightarrow} P_1''$. Since $P_1' \, R \, P_2'$, by lemma 2.25 there exists P_2'' s.t. $P_2' \stackrel{\sigma}{\Longrightarrow} P_2''$ and $P_1'' \, R \, P_2''$. So $r_{P_2''} = R(r_{P_1''}) \in succ(r_{P_2'})$. We have thus shown that $R(succ(r_{P_1'}) \subseteq succ(r_{P_2'})$. By a symmetrical argument we can show also: $succ(r_{P_2'}) \subseteq R(succ(r_{P_1'}))$, and this ends the proof of the theorem. □

2.4.4 Abstraction equivalence equals bisimulation equivalence

So far we have been concentrating on how bisimulations relate to the reduction relation $\stackrel{abs}{\longrightarrow}$. We now come to our main result, concerning the relationship between the abstraction equivalence $\sim_{abs}$ and the substitutive bisimulation equivalence $<\approx>^+$. It turns out that these two equivalences coincide:

Theorem 2.31 $\sim_{abs} = <\approx>^+$

Proof of $\subseteq$: From corollary 2.28 we can infer: $\sim_{abs} = [\stackrel{abs}{\longrightarrow} \cdot \stackrel{abs}{\longleftarrow}] \subseteq <\approx>^+$, since $<\approx>^+$ is symmetrically and transitively closed.

Proof of $\supseteq$: Suppose $P_1 <\approx>^+ P_2$. We want to show that $\exists \, P_3$ s.t. $P_1 \stackrel{abs}{\longrightarrow} P_3 \stackrel{abs}{\longleftarrow} P_2$. For any NDP P, and $\sigma \in A^{\odot}$, let $Der_\sigma(P) = \{P' | P \stackrel{\sigma}{\Longrightarrow} P'\}$ and $PDer_\sigma(P) = \{P' \mid P \stackrel{\sigma}{\Longrightarrow} P', P \neq P'\}$.

Note that $P_1 <\approx>^+ P_2$ implies $P_1 E(<\approx>_a^+) P_2$, i.e. $P_1 E(<\approx>) P_2$, where E is the relation defined at page 36. Then it is easy to check that the relation:

$$R = (P_1, P_2) \cup <\approx> \upharpoonright (PDer_\sigma(P_1) \times PDer_\sigma(P_2))$$

is a bisimulation between P_1 and P_2 s.t. $P_2 \notin R(PDer(P_1))$. However R will not, in general, be single-valued. Let then $\sim$ be the equivalence induced by R on the states Q_2 of P_2:

$$r_{P_2'} \sim r_{P_2''} \text{ iff } \exists \, P_1' \in PDer(P_1) \text{ s.t. both } (P_1', P_2') \text{ and } (P_1', P_2'') \in R$$

It can be easily shown that $\sim$ is a congruence on P_2. Therefore, by theorem 2.30, there exist an NDP P_3 and an a.h. h s.t. $h : P_2 \longrightarrow P_3$. So $P_2 \stackrel{abs}{\longrightarrow} P_3$.

Now, by theorem 2.30, h can be regarded as a bisimulation between P_2 and P_3. Consider then the composition: $R' = h \circ R$. By construction R' is single-valued and s.t. $R' \subseteq (Der(P_1) \times Der(P_3))$. Also, R' is a bisimulation because both R and h are. Finally, since $P_2 \notin R(PDer(P_1))$ and $P_3 \notin h(PDer(P_2))$, we have: $P_3 \notin R'(PDer(P_1))$. Thus, by theorem 2.30 again, R' is an a.h. from P_1 to P_3. So $P_1 \xrightarrow{abs} P_3$. Summing up, we have shown that $P_1 \xrightarrow{abs} P_3 \xleftarrow{abs} P_2$, that is to say $P_1 \sim_{abs} P_2$. □

In view of the last theorem, the equivalence $\sim_{abs}$ can be regarded as an alternative definition for $<\approx>^+$. In the following sections, we will see how this new characterisation can be used to establish a notion of *minimality* on processes, and to derive a set of reduction rules for $<\approx>^+$ on finite processes.

2.5 Minimal NDP's

Our reduction relation $\xrightarrow{abs}$ can be used to define a notion of minimality on NDP's. Intuitively, a process P is *minimal* if it cannot be further simplified by means of an abstraction homomorphism, that is, if any a.h. on P is an isomorphism. Formally:

Definition 2.32 *An NDP P is minimal (or irreducible) iff, for any NDP P':*

$$P \xrightarrow{abs} P' \iff P = P' \ (up\ to\ isomorphism)$$

The natural question is whether any NDP P can be reduced to some minimal NDP $\hat{P}$. We already know, by virtue of $\xrightarrow{abs}$'s Church-Rosser property, that if such a $\hat{P}$ exists, it will be unique. We want now to show that $\hat{P}$ always exists. We will then have a notion of *canonical representative* for an NDP.

In the previous section we established a correspondence between abstraction homomorphisms on an NDP P and congruences on the states of P. The obvious idea is then to take $\hat{P}$ to be the quotient of P w.r.t. the *maximal* congruence on P. But first, we need to make sure that such a congruence exists.

We gave earlier, at page 39, a definition of bisimulation *between* two NDP's. It is easy to show that for any two NDP's, there exists a maximal bisimulation between them. We can then prove the following:

Proposition 2.33 *Let $P = (Q \cup \{r\}, \leq, l)$ be an NDP, and R_{max} be the maximal bisimulation between P and itself. Then the equivalence $\sim_{max}$ defined on Q as:*

$$\forall q_1, q_2 \in Q: \quad q_1 \sim_{max} q_2 \quad \textit{iff} \quad P_{q_1} R_{max} P_{q_2}$$

is the maximal congruence on P (it is important to note that $r \notin Dom(\sim_{max})$, since $\sim_{max}$ is restricted to Q). □

This proposition allows us to define representatives for NDP's:

Definition 2.34 *If P is an NDP and $\sim_{max}$ is the maximal congruence on P, then the canonical representative $\hat{P}$ for P is the NDP $\hat{P} = P/\sim_{max}$.*

We then have the following:

Proposition 2.35 $P \sim_{abs} P'$ *iff* $\hat{P} = \hat{P'}$ *(up to isomorphism).* □

To sum up, we have now a class of representative NDP's, which can be legitimately considered to be the abstract model we were seeking. In this light, NDP's are to be viewed as a pre-model, which needs to be further interpreted to yield the required model.

2.6 A language for finite processes

In this section, we study the subclass of *finite* NDP's, and show how it can be used to model terms of a simple language L.

The language is essentially a subset of R. Milner's CCS (Calculus of Communicating Systems [Mil 80]). In [HM 85] a set of axioms is presented for L that exactly characterises the equivalence $<\approx>^+$ (and therefore $\sim_{abs}$) on the corresponding transition systems. We show here that the reduction $\xrightarrow{abs}$ itself can be characterised algebraically, by a set of *reduction rules.* These rules yield normal forms which coincide with the ones suggested in [HM 85] .

We shall now introduce the language L . Following the approach of [HM 85] , we define L as the term algebra T_Σ over the signature:

$$\Sigma = A \cup \{NIL, +\}$$

If we assume the operators in Σ to denote the corresponding operators on NDP's (A is here identified with the set of unary operators $\mu:$), we can use

finite NDP's to model terms in T_Σ. If t is a term of T_Σ, we will write P_t for the corresponding NDP.

We shall point out, however, that the denotations for terms of T_Σ in $\mathcal{P}$ will always be *trees* (cf the definition of the operators on NDP's at page 21), that is NDP's $P = (Q \cup \{r\}, \leq, l)$ obeying the further constraint:

$$confluence\text{-}freeness: \quad \text{if } q \leq q'' \text{ and } q' \leq q'' \text{ then } q \leq q' \text{ or } q' \leq q$$

Consider now the set of axioms E_c:

	$E1.$	$x + x' = x' + x$
$sum - laws$	$E2.$	$x + (x' + x'') = (x + x') + x''$
	$E3.$	$x + \mathrm{NIL} = x$
	$E4.$	$\mu\tau x = \mu x$
$tau - laws$	$E5.$	$\tau x + x = \tau x$
	$E6.$	$\mu(x + \tau y) + \mu y = \mu(x + \tau y)$
$absorption\ law$	$E7.$	$x + x = x$

Let $=_c$ be the equality generated by E_c on L. It has been proved in [HM 83] that E_c is a sound and complete axiomatisation for Milner's *observational congruence* $\approx^c$ on L. We will not give here the definition of $\approx^c$, which can be found in [Mil 80, HM 83]. Let us just say that it presupposes the definition of the relations $\xrightarrow{\mu}$ (and $\stackrel{\mu}{\Longrightarrow}$) directly on the terms of T_Σ, as follows. For any $\mu \in A$, let $\xrightarrow{\mu}$ be the *least* relation on terms that satisfies:

i) $\mu t \xrightarrow{\mu} t$

ii) $t \xrightarrow{\mu} t'$ implies $(t + t'') \xrightarrow{\mu} t'$, $(t'' + t) \xrightarrow{\mu} t'$

The weak relations $\stackrel{\mu}{\Longrightarrow}$ are then defined in terms of the $\xrightarrow{\mu}$'s just as in section 2.4.2. Now, it is well-known that, for any $t, t' \in L$:

$$t \approx^c t' \quad \text{iff} \quad P_t <\approx>^+ P_{t'}$$

Combining these facts with the result of the preceding section, we can conclude that:

$$t =_c t' \quad \text{iff} \quad P_t \sim_{abs} P_{t'}$$

In other words, $=_c$ is an algebraic analogue for $\sim_{abs}$. Note on the other hand that, although each axiom of E_c could be viewed as a reduction rule (when applied from left to right), the corresponding reduction relation $\longrightarrow$ would not characterise $\stackrel{abs}{\longrightarrow}$. Consider for example the terms:

$$t = a\text{NIL} + \tau(a\text{NIL} + b\text{NIL}), \quad t' = \tau(a\text{NIL} + b\text{NIL})$$

We have $P_t \stackrel{abs}{\longrightarrow} P_{t'}$, but we cannot infer $t \longrightarrow t'$.

However, using the axiomatisation E_c as a reference, it is possible to derive a new system of reduction rules, which completely characterises $\stackrel{abs}{\longrightarrow}$.

Consider the reduction relation $\longrightarrow^c$ generated by the following set of reduction rules R_c, where $\longleftrightarrow$ stands for $(\longrightarrow \cap \longrightarrow^{-1})$, and the restrictive relations $\stackrel{\mu}{\Longmapsto}$ are derived from the $\stackrel{\mu}{\longrightarrow}$'s just as in section 2.4.2:

	$R1.$	$x + x' \longleftrightarrow x' + x$
$sum - laws$	$R2.$	$x + (x' + x'') \longleftrightarrow (x + x') + x''$
	$R3.$	$x + \text{NIL} \longrightarrow x$
$\tau - law$	$R4.$	$\mu\tau x \longrightarrow \mu x$
generalised absorption law	$R5.$	$x + \mu x' \longrightarrow x$ whenever $x \stackrel{\mu}{\Longmapsto} x'$

In what follows, we will often consider terms modulo the congruence induced by $R1 - R3$. When taken modulo $R1 - R3$, a term t can be expressed in the form $\sum_{i\in I} \mu_i t_i$, where $i \in I$ for some finite set of indices I. We set by convention $\sum_{i\in I} \mu_i t_i = \text{NIL}$ if $I = \emptyset$.

It is easy to check that the rules R_c are *sound* for NDP's under $\stackrel{abs}{\longrightarrow}$, in the sense that $t \longrightarrow^c t'$ implies $P_t \stackrel{abs}{\longrightarrow} P_{t'}$. We proceed now to show that the rules R_c are also *complete* for $\stackrel{abs}{\longrightarrow}$, namely that whenever $P_t \stackrel{abs}{\longrightarrow} P_{t'}$ we can infer $t \longrightarrow^c t'$.

To this purpose, we first prove a stronger version of lemma 2.26, the following:

Lemma 2.36 *($\stackrel{abs}{\longrightarrow}^{-1}$ almost preserves μ-summands)*

If $P_1 \stackrel{abs}{\longrightarrow} P_2$ *and* $P_2 \stackrel{\mu}{\longrightarrow} P_2'$ *then* $\exists P_1'$ *s.t.* $P_1 \stackrel{\mu}{\longrightarrow} P_1'$, *where either* $P_1' \stackrel{abs}{\longrightarrow} P_2'$ *or* $P_1' \stackrel{abs}{\longrightarrow} \tau P_2'$.

Proof: Let $h : P_1 \longrightarrow P_2$ be an a.h. and suppose $P_2 \xrightarrow{\mu} P_2'$. Then, if r_2' is the root of P_2' in P_2, we have $l(r_2') = \mu$. Also, since h is surjective, $r_2' = h(r_1')$ for some state $r_1' \in Q_1$. Now in general it will be:

$$r_1 \mathrel{-\!\!\subset} q_1 \mathrel{-\!\!\subset} \cdots \mathrel{-\!\!\subset} q_n = r_1'$$

Since h is order-preserving, this implies:

$$r_2 = h(r_1) \leq h(q_1) \leq \cdots \leq h(q_n) = h(r_1') = r_2'$$

Now, we know that $r_2 \mathrel{-\!\!\subset} r_2'$ and $r_2 \notin h(Q_1)$. Therefore it must be:

$$r_2 = h(r_1) \mathrel{-\!\!\subset} h(q_1) = \cdots = h(q_n) = h(r_1') = r_2'$$

This implies:

$$l(q_1) = \cdots = l(q_n) = l(r_2') = \mu$$

Let P_1' be the derivative of P_1 whose root is q_1. Then we have $P_1 \xrightarrow{\mu} P_1'$, and the rest of the proof goes as for lemma 2.25. □

We now have the following (soundness and) completeness result:

Theorem 2.37 $t \longrightarrow^c t'$ *iff* $P_t \xrightarrow{abs} P_{t'}$

Proof of completeness: We show, by induction on the sum of the sizes of P_t and $P_{t'}$, that $P_t \xrightarrow{abs} P_{t'}$ implies $t \longrightarrow^c t'$.

Assume $t = \sum_{i \in I} \mu_i t_i$ and $t' = \sum_{j \in J} \nu_j t_j'$, where $i \in I$ and $j \in J$. In the rest of the proof, we shall use P, P' for $P_t, P_{t'}$ and P_i, P_j' for $P_{t_i}, P_{t_j'}$. Let $\nu_j t_j'$ be a summand of t' . By lemma 2.36 $\exists\, i \in I$ s.t. $\mu_i = \nu_j$ and either $P_i \xrightarrow{abs} P_j'$ or $P_i \xrightarrow{abs} \tau P_j'$. Correspondingly we have, by the induction hypothesis, that either $t_i \longrightarrow^c t_j'$ or $t_i \longrightarrow^c \tau t_j'$. In both cases we can deduce $\mu_i t_i \longrightarrow^c \nu_j t_j'$ (using R4 for the latter case).

So, corresponding to any $j \in J$, we can find $i \in I$ s.t. $\mu_i t_i \longrightarrow^c \nu_j t_j'$. Let $I_j \subseteq I$ be the set of all indices i thus chosen to match some $j \in J$. Then we have:

$$\sum_{i \in I_J} \mu_i t_i \longrightarrow^c \sum_{j \in J} \nu_j t_j'$$

Whence, substituting in t :

$$t = (\sum_{i \in I_J} \mu_i t_i + \sum_{k \in I - I_J} \mu_k t_k) \longrightarrow^c (\sum_{j \in J} \nu_j t_j' + \sum_{k \in I - I_J} \mu_k t_k) = t' + \sum_{k \in I - I_J} \mu_k t_k$$

We show now that $(t' + \sum_{k \in I-I_J} \mu_k t_k) \longrightarrow^c t'$, and this will end the proof of the theorem.

Each $\mu_k t_k$ is a summand of t. Thus for $P = P_t$ and $P_k = P_{t_k}$, we have $P \overset{\mu_k}{\Longmapsto} P_k$. Since $P \overset{abs}{\longrightarrow} P'$, we can deduce (by lemma 2.25) that $\exists\, P''$ s.t. $P' \overset{\mu_k}{\Longmapsto} P''$, where either $P_k \overset{abs}{\longrightarrow} P''$ or $P_k \overset{abs}{\longrightarrow} \tau P''$. Let now t'' be s.t. $P'' = P_{t''}$. Note that $t' \overset{\mu_k}{\Longmapsto} t''$.

By induction we have either $t_k \longrightarrow^c t''$ or $t_k \longrightarrow^c \tau t''$. In any case we can deduce $\mu_k t_k \longrightarrow^c \mu_k t''$. Thus we have:

$$(t' + \mu_k t_k) \longrightarrow^c (t' + \mu_k t'')$$

Since $t' \overset{\mu_k}{\Longmapsto} t''$, we can now use $R5$ to get:

$$(t' + \mu_k t'') \longrightarrow^c t'$$

As this can be repeated for all $k \in (I - I_j)$, we conclude that:

$$(t' + \sum_{k \in I-I_J} \mu_k t_k) \longrightarrow^c t'$$

To sum up, we have shown that:

$$t = (\sum_{i \in I_J} \mu_i t_i + \sum_{k \in I-I_J} \mu_k t_k) \longrightarrow^c (\sum_{j \in J} \nu_j t'_j + \sum_{k \in I-I_J} \mu_k t_k) = (t' + \sum_{k \in I-I_J} \mu_k t_k) \longrightarrow^c t'$$

□

Corollary 2.38 *R_c is a rewriting system for the equational theory E_c.* □

We can make use of our new axiomatisation for $=_c$ to characterise normal forms for terms in T_Σ. We say that a term is in *normal form* if no proper reduction (R3, R4 or R5) can be applied to it. It can be shown that:

Theorem 2.39 *A term $t = \sum_{i \in I} \mu_i t_i$ is a normal form iff it satisfies the properties* (Hennessy-Milner characterisation):

- i) *no t_i is of the form $\tau t'$*
- ii) *each t_i is a normal form*
- iii) *if $i \neq j$ then $\mu_j t_j \overset{\mu_i}{\Longrightarrow} t'_j$ implies $t_i \not\longleftrightarrow t'_j$*

A natural question at this point is whether there is a correspondence between normal forms and canonical NDP's, a notion we introduced in section 2.5. Let us just recall that, if P is an NDP, the corresponding canonical NDP $\hat{P}$ is a minimal NDP (no more reducible w.r.t. $\xrightarrow{abs}$), which can be viewed as the "abstract" representative for P. Now, as pointed out earlier, the denotation P_t of a term t is always a *tree*. However its "abstract" denotation P_t might not be a tree. So, if $\hat{t}$ is the normal form for the term t, we do *not* have, in general, the equality $\hat{P}_t = P_{\hat{t}}$. On the other hand, the following holds:

Proposition 2.40 $\hat{t} = \hat{t'} \iff \hat{P}_t = \hat{P}_{t'}$

We conclude this section by giving a characterisation of finite minimal NDP's.

Proposition 2.41 *An NDP P is minimal iff it meets the requirements:*

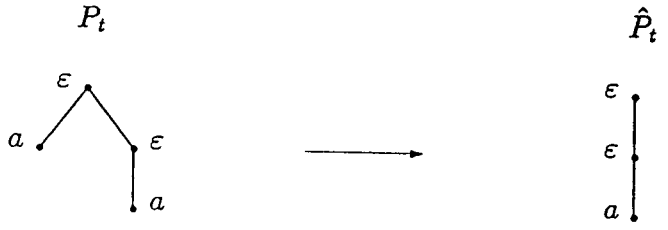
i) *A node which has no brothers does not have the same label as its father, unless the latter is the root.*

ii) *No node of P has two isomorphic σ-labelled (proper) successors.*

Note that identifying isomorphic σ-successors in an NDP (or in a tree) is made particularly easy by our labelling on the nodes.

Examples

1) Let $t = a\mathrm{NIL} + \tau a\mathrm{NIL}$. The corresponding NDP is not minimal. We have:



On the other hand, it is generally preferable to have a congruence rather than a simple equivalence relation, especially when an algebraic characterisation is desired. Moreover, we need very little change in the definition of a.h. to obtain $\sim_{abs} = <\approx>^+$ rather than $\sim_{abs} = <\approx>$ (whereas it is somewhat laborious, as we saw it in the previous sections, to pass from $<\approx>$ to its closure $<\approx>^+$ while keeping a recursive definition).

Let us spend now a few words about our finiteness restriction for NDP's: we said when defining it that this restriction amounts to the condition of *image-finiteness* for transition systems. We did not give at that point the definition of image-finiteness, since we had not yet introduced the weak transition relations. The definition is thus given here.

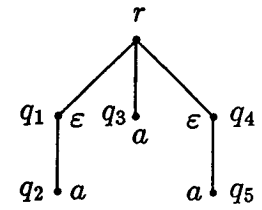
An NDP is *image-finite* iff $\forall$ state $q, \forall \mu$: the set $\{q' \mid q \stackrel{\mu}{\Longrightarrow} q'\}$ is *finite*

This property is often assumed for LTS's, and indeed required as a condition for many classical results (see [HM 83]). We saw that it was necessary here to prove some of our results.

As regards our choice of NDP's (in place of labelled trees), we mentioned that this was due to a technical reason. In fact, if we had taken trees as our model, we would have needed a more restrictive notion of congruence : there would have been some additional condition to preserve confluence-freeness. But then, our definition for the *sup* $\sim_{1,2}$ of two congruences $\sim_1, \sim_2$, which is somehow standard, would no longer be adequate. This can be illustrated by an example:

Consider the following NDP P (which is also a tree), and the two congruences $\sim_1$ and $\sim_2$ on P (of which we only show the nontrivial pairs):

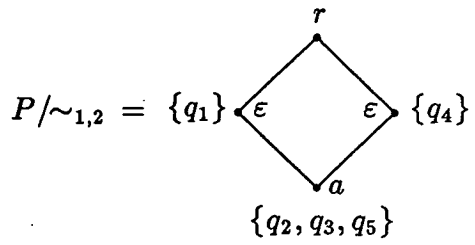
$P =$



with $\sim_1$ given by: $q_2 \sim_1 q_3$

and $\sim_2$ given by: $q_3 \sim_2 q_5$

Note that $\sim_1$ and $\sim_2$ would both be admissible congruences for trees, since they both preserve confluence-freeness. On the other hand the equivalence $\sim_{1,2} = [\ \sim_1 \cup \sim_2\ ]^*$ would *not* be an admissible congruence for trees, since it yields the following quotient NDP:



which is not a tree.

Chapter 3

Abstraction on concurrent processes

In the previous chapter we have been dealing with transition systems evolving through sequences of elementary actions, what we could call *sequential actions.* Such systems, factored by an appropriate equivalence relation, provide a good abstract model of nondeterministic behaviour.

We address now the question of finding a similarly abstract model for programs which are both concurrent and nondeterministic. In fact, as pointed out earlier, the interest in nondeterministic behaviours is largely due to the fact that they arise in models of concurrent programming. Indeed, if we adopt the – widely accepted – sequential interpretation of concurrency as nondeterministic interleaving, a model for nondeterminism is all we need to deal with concurrent programs. The model for the calculus CCS [Mil 80] , as proposed by Milner and his collaborators, is an example of such a sequential explanation of parallelism (if we forget for a moment about communication).

Here however we shall abandon this sequential view, and attempt to represent concurrency as a *primitive* structural feature of a system, which should be apparent in the system's behaviour. What is needed is then some notion of "concurrent" transition system. The obvious way to obtain this would be to generalise the notion of *action* of a system.

We recall that for an LTS S an action η , that is the label of a transition

$$S \xrightarrow{\eta} S'$$

represents a (partial) *computation* of the system S. In particular, if S is purely nondeterministic, η is just a sequence of elementary actions. On the other hand,

should be allowed to contain some nonsequentiality between its constituent actions. In other words we would need to generalise sequential actions into (possibly) *nonsequential actions.*

The transition system for the calculus MEIJE of Austry and Boudol [AB 84] , which we shall briefly examine in section 3.1.2, represents a step in this direction. Here actions of a concurrent program are elements of a commutative semigroup – an idea that originated with Milner's calculus SCCS [Mil 83] , and the occurrence of a compound action amounts to the simultaneous occurrence of all its components. Such composite actions allow us to distinguish the two systems (described in a CCS notation):

$$a \mid b \qquad \text{and} \qquad a\,b + b\,a \tag{3.1}$$

since the first one is capable of an action $\xrightarrow{a \cdot b}$, while the second one is not. On the other hand the following two systems:

$$a \mid b \qquad \text{and} \qquad a \mid b + a\,b + b\,a \tag{3.2}$$

are considered to have the same operational behaviour, whereas the second one shows some additional *causal dependencies* which, to our view, should not be ignored.

And here we come to what is usually taken as a major argument in favour of a partial ordering semantics for concurrency. The idea is that a concurrent system is firstly characterised by its *causal structure.* Now any semantics projecting concurrent activities onto a global time-scale – as are the interleaving semantics for CCS and the above-mentioned semantics for MEIJE – blurs the causal connectivity by adding new temporal sequences to the existing causal ones. Whence the "plea" for a partial ordering model, where two actions are ordered if and only if they are causally related. In such a model sequentiality becomes a synonym of causality.

Now, assuming for a moment that we agree with this philosophy, why don't we simply adopt one of the existing partial ordering models as it stands? I think here particularly of Petri-nets [Petri 77, Net 80] and related models – such as Winskel's Event Structures [Win 80] – some of which have by now been well investigated.

concurrency" . Perhaps the most serious is that they lack, to my knowledge, of an operational semantics which reflects their original intention. The operational rules (*firing rules*) usually adopted for labelled Petri nets, for example – allowing one action or a set of concurrent actions to fire in one step – do not preserve all the significant distinctions of the model. It is easy to see, in fact, that these two rules determine LTS's which correspond respectively with the one for CCS (yielding an interleaving semantics) and the one for MEIJE (see section 3.1.3). Another common objection to Petri nets is their lack of structure, which makes it difficult to change the level of abstraction of a representation.

To sum up there is not, as yet, a satisfactory notion of operational behaviour for true-concurrency models. This may lead us to question the capacity of "traditional" transition systems to capture the concurrent structure of programs. That is why, in Chapter 4, we shall propose a non-standard notion of LTS, what we call *distributed LTS*, which seems better suited to our purposes. The idea underlying distributed LTS's is that for any transition one can observe, besides the corresponding action and residual system, also the components of the system which are concurrent to that transition. In Chapter 5 we shall extend this "distributed" approach to concurrent programs with *unobservable* actions. Later still, in Chapter 6, we will recover a more standard notion of LTS, where the additional descriptive power as regards concurrency is gained by admitting whole concurrent programs as actions.

As for now, we shall momentarily put aside transition systems and turn, in section 3.2 of this chapter, to the more general question of establishing an *abstraction mechanism* – possibly not based on an operational intuition – on concurrent programs.

Essentially, we shall try to generalise our treatment for the sequential case. We recall that in that case a syntax was used to provide the desired algebraic setting, and our abstraction mechanism (based on abstraction homomorphisms) was introduced directly on the model. Only afterwards was this mechanism shown to be closely tied to an operational semantics – defined on the algebra by means of structural behavioural rules.

We shall here proceed along the same lines. We adopt a syntax which is the same as for the sequential case – essentially sum and prefixing – with the ad-

Labelled Event Structures, a model due to Winskel, Nielsen and Plotkin [NPW 81] , which appears simple enough, yet exhibiting many interesting features. We then define abstraction homomorphisms on LES's, and give an axiomatisation for the corresponding reduction relation.

At that point a question will naturally arise: does our "a priori" abstraction in any way help us in the search for an operational semantics? Can our abstraction criterion be turned into an operationally significant one? These questions will bring us to the following chapter, where we shall see that they can be answered – to a large extent – positively.

3.1 Some existing LTS models for concurrent programs

3.1.1 The reference language

Here and in the rest of the chapter, we shall limit our analysis to a very simple class of concurrent programs. We shall only consider *finite* programs, evolving via *observable* transitions. Programs will be described as terms of a language CL. Just like the language L examined in the previous chapter (section 2.6), the language CL is presented as the term-algebra over a given signature. In fact, our signature Σ for CL is the same as the one for L, with the addition of a *parallel operator* $|$. Namely we take :

$$CL = T_\Sigma, \qquad \text{with} \qquad \Sigma = A \cup \{\text{NIL}, +, | \}$$

Terms of CL will be denoted by t, t', etc. Also, since A is assumed to contain only observable actions, we shall use $a, b, \ldots$ to range over it.

Our intention is to find an interpretation of CL which ensures some "minimal" set of properties for $|$, like:

$P1.$ $\quad x \,|\, x' = x' \,|\, x$

$P2.$ $\quad x \,|\, (x' \,|\, x'') = (x \,|\, x') \,|\, x''$

while keeping a distinction between concurrency and nondeterministic interleaving.

guage CL: we shall point to what we consider to be the "weaknesses" of these interpretations as regards the modelling of concurrency, and thus try to justify our departure from them in subsequent sections.

3.1.2 Algebraic calculi: CCS, MEIJE, SCCS

According to the terminology in use [Mil 83, Bou 85] , we shall call *algebraic calculus of processes* any formalism for defining processes which meets the following requirements:

- Processes are defined as terms of a Σ-algebra , Σ being the *syntax* (signature) of interest.
- Operationally, processes are interpreted as Labelled Transition Systems. The meaning of the operators is specified by a set of rules. These are called *structural* in that they define the behaviour of a compound process in terms of the behaviour of its components.
- A *behavioural equivalence* is defined by means of the notion of *bisimulation* on transition systems.

The calculi CCS and SCCS of Milner [Mil 80, Mil 83] and the calculus MEIJE of Austry and Boudol [AB 84] , which we shall shortly review, are examples of such formalisms. These three calculi include in their syntax the operators of the language CL introduced in the previous section. Since we are here principally interested in the treatment of concurrency in these different calculi, we shall restrict ourselves to their common subset CL. Besides, for each of the calculi, there exists an axiomatisation of bisimulation equivalence on the subset CL, which will make our comparison easier.

To start with the simplest case, we shall interpret the operator $|$ as just putting processes side by side, without allowing communication between them (a more general interpretation for $|$ will be considered in the next chapter) . As will soon be apparent, such a *juxtaposition* operation is asynchronous in the case of CCS and MEIJE, and synchronous in the case of SCCS (where it is usually denoted by $\times$). Let us now formally present the calculi.

We say that a set of rules specifies the behaviour of a language of terms if, for any term t in the language, it allows us to infer exactly all the transitions of t.

In CCS the behaviour of terms of T_Σ is specified by the rules:

i) $a:t \xrightarrow{a} t$

ii) $t \xrightarrow{a} t'$ implies $(t+s) \xrightarrow{a} t'$, $(s+t) \xrightarrow{a} t'$

iii) $t \xrightarrow{a} t'$ implies $(t \mid s) \xrightarrow{a} (t' \mid s)$, $(s \mid t) \xrightarrow{a} (s \mid t')$

The corresponding bisimulation equivalence (defined in the usual way, cf section 2.1.1) is characterised by the set of axioms (see [HM 83]) :

$\mathcal{A}$

	$A1.$	$x + x' = x' + x$
sum − laws	$A2.$	$x + (x' + x'') = (x + x') + x''$
	$A3.$	$x + \mathrm{NIL} = x$
absorption law	$A4.$	$x + x = x$
	$P1.$	$x \mid x' = x' \mid x$
par − laws	$P2.$	$x \mid (x' \mid x'') = (x \mid x') \mid x''$
	$P3.$	$x \mid \mathrm{NIL} = x$
interleaving	IN.	If $x = \sum_{i\in I} a_i x_i$, $y = \sum_{j\in J} b_j y_j$, *then* $x \mid y = \sum_{i\in I} a_i (x_i \mid y) + \sum_{j\in J} b_j (x \mid y_j)$

The law IN is often called *expansion theorem*, in that it allows a parallel term to be expanded into the sum of smaller parallel terms (with fewer occurrences of $\mid$). This and the preceding axioms A1-A3 justify the summation notation $\sum_{i\in I} a_i x_i$ used in IN (where again we assume $\sum_{i\in I} a_i x_i = \mathrm{NIL}$ if $I = \emptyset$).

By repeatedly applying the law IN , one can gradually push down the operator $\mid$, and indeed eliminate it from finite terms (in fact laws P1–P3 are not necessary here). In this sense we can say that the axiom IN expresses the simulation of concurrency as nondeterministic interleaving.

Example 1: $a \mid b = ab + ba$

Here actions constitute a commutative semigroup $(A, \cdot)$. The occurrence of a compound action represents the simultaneous occurrence of its constituent actions.

The behaviour of terms of T_Σ is specified by the rules:

i) $a : t \xrightarrow{a} t$

ii) $t \xrightarrow{a} t'$ implies $(t + s) \xrightarrow{a} t'$, $(s + t) \xrightarrow{a} t'$

iii') $t \xrightarrow{a} t'$ and $s \xrightarrow{b} s'$ imply $(t \mid s) \xrightarrow{a \cdot b} (t' \mid s')$

The corresponding bisimulation equivalence is characterised by the set of axioms [Mil 83] :

$\mathcal{B}$

	$A1.$	$x + x' = x' + x$
$sum - laws$	$A2.$	$x + (x' + x'') = (x + x') + x''$
	$A3.$	$x + \mathrm{NIL} = x$
$absorption\ law$	$A4.$	$x + x = x$
$par - laws$	$P1.$	$x \mid x' = x' \mid x$
	$P2.$	$x \mid (x' \mid x'') = (x \mid x') \mid x''$
$synchrony$	SP.	If $x = \sum_{i \in I} a_i x_i$, $y = \sum_{j \in J} b_j y_j$, *then* $x \mid y = \sum_{i \in I, j \in J} a_i \cdot b_j \, (x_i \mid y_j)$

The law SP states that in SCCS the operator $\mid$ (usually noted $\times$) is a *synchronous parallel composition*: at each step all parallel components are forced to move together. This amounts to impose a *maximal* degree of *parallelism* in the execution of a parallel process. In this sense the approach of SCCS is opposite to that of CCS, where the execution of a parallel process is strictly sequential.

Example 1. However, because of the synchrony constraint, it does not capture the "locality" of causal dependencies. For consider the exchange of causalities in the example:

Example 2: $a \mid bc = ac \mid b$

Note that the following law, which does not hold in CCS, can be derived in SCCS:

$$\textit{distributivity law} \qquad x \mid (y+z) = x \mid y + x \mid z$$

The synchrony law SP is another example of expansion theorem. Again, the operator $\mid$ can be eliminated from finite terms, and is therefore *not primitive.* Note however that, in contrast with CCS, some concurrency is retained here in the actions.

Finally, remark that the law:

$$P3. \quad x \mid \mathrm{NIL} = x$$

is not valid in SCCS, where we get instead $x \mid \mathrm{NIL} = \mathrm{NIL}$ as an instance of the synchrony law SP (for $y = \mathrm{NIL}$).

The calculus MEIJE

Here again actions are elements of a commutative semigroup $(A, \cdot)$. The rules specifying the behaviour of terms of T_Σ are given below. Note that the rules for the parallel operator are obtained by joining the rule iii) of CCS and the rule iii') of SCCS.

i) $a : t \xrightarrow{a} t$

ii) $t \xrightarrow{a} t'$ implies $(t+s) \xrightarrow{a} t'$, $(s+t) \xrightarrow{a} t'$

iii) $t \xrightarrow{a} t'$ implies $(t \mid s) \xrightarrow{a} (t' \mid s)$, $(s \mid t) \xrightarrow{a} (s \mid t')$

iii') $t \xrightarrow{a} t'$ and $s \xrightarrow{b} s'$ imply $(t \mid s) \xrightarrow{a \cdot b} (t' \mid s')$

The bisimulation equivalence is characterised by the set of axioms:

	$A1.$	$x + x' = x' + x$
$sum - laws$	$A2.$	$x + (x' + x'') = (x + x') + x''$
	$A3.$	$x + \mathrm{NIL} = x$
$absorption\ law$	$A4.$	$x + x = x$
	$P1.$	$x \mid x' = x' \mid x$
$par - laws$	$P2.$	$x \mid (x' \mid x'') = (x \mid x') \mid x''$
	$P3.$	$x \mid \mathrm{NIL} = x$
$asynchrony$	AP.	If $x = \sum_{i \in I} a_i x_i$, $y = \sum_{j \in J} b_j y_j$, *then* $x \mid y = \sum_{i \in I} a_i (x_i \mid y) + \sum_{j \in J} b_j (x \mid y_j) + \sum_{i \in I, j \in J} a_i \cdot b_j (x_i \mid y_j)$

The law AP shows that in MEIJE the operator $\mid$ is indeed an *asynchronous parallel composition* (this name was used already by Milner in [Mil 81]): at each step, either one of the components moves first, or both the components move together. Unlike CCS, MEIJE allows simultaneity between parallel components, while not enforcing it as does SCCS. This allows us to distinguish between the terms of the above examples 1 and 2.

Yet, this is not quite sufficient to account for the causal structure of processes. The following example, which we recall from the introduction:

Example 3: $\quad a \mid b \;=\; a \mid b \;+\; a\,b \;+\; b\,a$

shows that causalities can still be *absorbed* within parallel terms.

Note to conclude that the asynchrony law AP is a third instance of expansion theorem.

Let us close this review about calculi with a summary of their different approaches to concurrency.

- The calculus CCS models the asynchronous behaviour of parallel processes, while not allowing any real concurrency between their actions.

haviours but represents concurrency of actions to some extent. Precisely, SCCS allows – in fact, enforces – simultaneity among concurrent actions of equal "depth" (while it sequentialises concurrent actions of different depth).

- Finally, the calculus MEIJE represents both asynchrony and concurrency. Here concurrent actions of any depth are allowed to occur simultaneously. There remains a question whether causality is sufficiently expressed in MEIJE.

We should stress that the presentation of the various calculi we have given here is a simplified one. In particular what we say about the sequentialisation of concurrent actions in CCS is not true any more when communication is allowed (see Appendix for a description of CCS with communication).

Also, our review is by no means exhaustive: there are other examples of algebraic calculi for concurrent processes, like Hoare & Brookes/Roscoe's TCSP and the algebra ACP of Bergstra & Klop [BK 84].

In the next section we shall consider three notions of behaviour – determined by so-called *firing rules* – for Petri-nets, which closely correspond to the three approaches examined here.

3.1.3 Partial ordering models. Petri nets firing rules

In this section we eamine some notions of behaviour for Petri–nets. The theory of Petri–nets, developed from the early work of Carl Adam Petri [Petri 62] , provides a nonsequential model for concurrent systems which has been a reference for many subsequent proposals.

Petri-nets are essentially bipartite directed graphs – with two kinds of nodes, circles which represent *places* (local states) and boxes representing *transitions.* Circles and boxes are connected by arcs, which give the so-called *flow relation* of the net. Formally:

Definition 3.1 *A* (Petri) net *is a triple* $N = (S, T, F)$ *where:*

- S *is a set of* places
- T *is a set of* transitions, *such that* $S \cap T = \emptyset$
- $F \subseteq (S \times T) \cup (T \times S)$ *is the* flow relation, *such that* $T \subseteq range(F)$

The set $X = S \cup T$ is the set of *elements* of the net. For any $x \in X$, the sets ${}^\bullet x =_{def} \{y \mid y\, F\, x\}$ and $x^\bullet =_{def} \{y \mid x\, F\, y\}$ are called respectively the *preset* and *postset* of x. The condition $T \subseteq range(F)$ means that transitions cannot be isolated: they are always connected to some place. This is a natural restriction if one thinks of transitions as changes of state. On the other hand isolated places are allowed, and indeed turn out to be convenient for defining the notion of process of a net (see later).

The definition above only gives a static description of a system. To describe the dynamics of a system, one needs the notion of *marked net*, or "net in a given state". A *marking* of a net is a distribution of *tokens* on its places, which enables some of its transitions to occur (to *fire* in Petri nets terminology).

Intuitively, assigning a marking to a net is a way of designating a *global* state for it, made out of several local states. Formally, a *marking* of a net $N = (S, T, F)$ is a mapping $M : S \longrightarrow I\!N$.

Definition 3.2 *A* marked net *is a net* $N = (S, T, F)$ *together with a marking:*

- $M_0 : S \longrightarrow I\!N$, *called the* initial marking *of* N.

Graphically, a marking M is represented by drawing $M(s)$ dots on each place s of the net.

In fact, even before we start wondering about behaviours, the notion of marking is necessary to specify the relations among the elements of a net. While a notion of *independence* can be defined for transitions of an unmarked net (t_1 and t_2 are independent if they share no bordering places), the definitions of *concurrency* and *conflict* between transitions, which are fundamental to Net theory, can only be given relatively to a given marking. Let us now introduce these definitions.

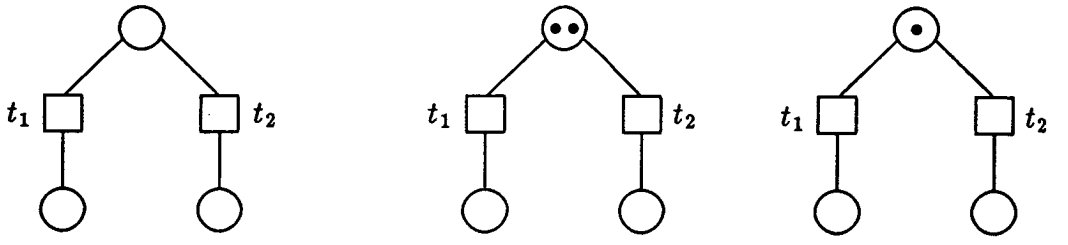
Definition 3.3 *Let $N = (S, T, F)$ be a net and M be a marking of N. Then:*

- *A transition $t \in T$ is* enabled at M *if and only if:*
  $\forall s \in S \quad M(s) \geq \mid s^\bullet \cap \{t\} \mid$ (i.e. $\forall s \in {}^\bullet t,\ M(s) \geq 1$).
- *Two transitions $t_1, t_2 \in T$ are* concurrently enabled (*or simply* concurrent) *at M iff* $\forall s \in S : \ M(s) \geq \mid s^\bullet \cap \{t_1\} \mid \ + \ \mid s^\bullet \cap \{t_2\} \mid$.
- *Two transitions $t_1, t_2 \in T$, $t_1 \neq t_2$, are in* conflict *at M if and only if they are both enabled at M but they are not concurrently enabled at M.*

Note that a transition may be concurrent with itself. We could have asked for $t_1 \neq t_2$ in the definition of concurrency. In that case the condition above would simplify to: $M(s) \geq \mid s^\bullet \cap \{t_1, t_2\} \mid$. However, the general definition turns out to be more satisfactory.

Let us see some examples. In the net below, the transitions t_1 and t_2 are statically *dependent* in that they are both connected to the place p. On the other hand they are *concurrent* in the marking shown on the right:

Consider now another net, with two transitions t_1 and t_2 which are again dependent, and become in turn *concurrent* and in *conflict* in the two given markings.



Based on the notion of marking, several transition rules can be defined on marked nets, specifying how a net evolves from one marking to the other. Each of these rules – called *firing rules* – gives rise to a different notion of computation for nets. Traditionally, a computation is assumed to be a sequence of firing steps, and the semantics of a net is defined to be the *set* of its computations (whichever is the chosen notion of computation).

Before passing to defining these rules, we want to add a further feature to our nets. Up to this point, the transitions of a net are all distinct. We shall now introduce a *labelling* on nets, so that different transitions may have the same label. This will enable us to maintain – at least informally – a correspondence between nets and the terms of our language CL (think e.g. of the terms $(a+a), (a \mid a)$, etc.). We give next the definition of *labelled marked net.*

Definition 3.4 *Let A be a given alphabet. Then:*
A labelled marked net *over A is a 5-tuple $N = (S, T, F, M_0, l)$, where:*

- S *is the set of* places
- T *is the set of* transitions, $S \cap T = \emptyset$
- $F \subseteq (S \times T) \cup (T \times S)$ *is the* flow relation, $T \subseteq range(F)$
- $M_0 : S \longrightarrow I\!N$, *is the* initial marking
- $l : T \longrightarrow A$, *is the* labelling function.

From now on, we shall always omit the names of transitions when drawing labelled nets. This kind of abstraction will be reflected in all the notions of computation that we shall consider.

Let us now proceed to examine these different notions of computation.

Simple firing rule

The basic firing rule for Petri nets (which we call *simple* here) allows *only one* enabled transition to fire at a time. The definition follows.

Definition 3.5 (*Simple firing rule*)
Let $N = (S, T, F, M_0, l)$ *be a labelled marked net, and* M, M' *be markings of* N. *Then:*

- *A transition* $t \in T$ fires from M to M', *in notation* $M\,[t>\,M'$, *iff:*

  i) t *is enabled in* M.

  ii) *For any* $s \in S$, $M'(s) = M(s) - |\,s^\bullet \cap \{t\}\,| + |\,{}^\bullet s \cap \{t\}\,|$.

- *Let* $a \in A$. *The marking* M moves under a *to the marking* M', *in notation* $M \xrightarrow{a} M'$, *iff there exists* $t \in T$ *s.t.* $M\,[t>\,M'$ *and* $l(t) = a$.

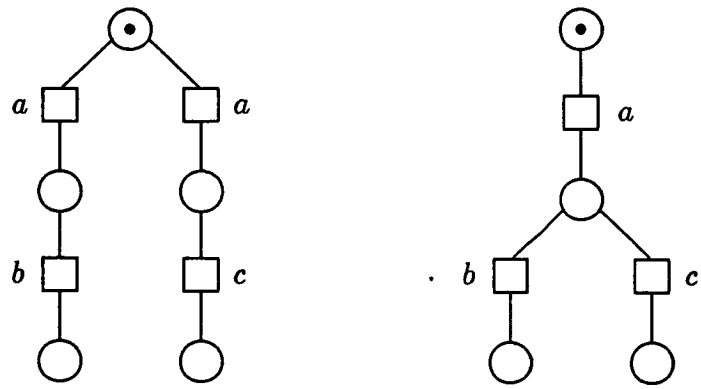
Of course, since we want to abstract from the names of transitions, we are interested in the "abstract" relations $\xrightarrow{a}$, rather than in the relations $[t>$. In particular, computations will be defined in terms of the abstract relations.

Definition 3.6 *Let* $N = (S, T, F, M_0, l)$ *be a labelled marked net over* A.

- *A finite or infinite sequence* $a_1, a_2, \ldots$ *of labels,* $a_i \in A$, *is a* firing sequence *of* N *if and only if there exist markings* $M_1, M_2, \ldots$ *such that* $M_0 \xrightarrow{a_1} M_1 \xrightarrow{a_2} M_2 \ldots$.

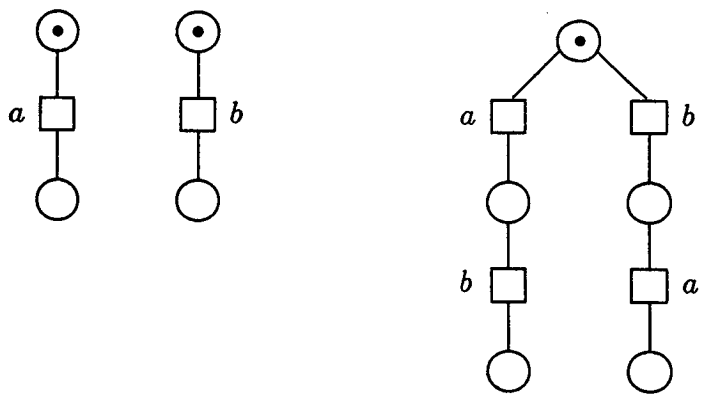
- *A* computation *of* N *is any of its firing sequences.*

As mentioned earlier, the behaviour of a net is usually taken to be the set of its computations. For labelled nets, however, one may wish to have a more refined notion of behaviour, in order to distinguish for example the two nets below, intuitively representing the terms $(ab + ac)$ and $a(b + c)$.



Since nets are now interpreted as labelled transition systems w.r.t. the transition relations $\xrightarrow{a}$, it is immediate to define a bisimulation equivalence on them. We then define the behaviour of a net to be its bisimulation equivalence class. It is easy to see that we obtain in this way an interleaving semantics for concurrency, as was the case for the calculus CCS. As an example, the two nets below, representing the terms $(a \mid b)$ and $(ab + ba)$, have the same behaviour.

Example 1



In fact U. Goltz and A. Mycroft give in [GM 84] a translation of (a subset of) CCS into Petri nets, and show the equivalence between CCS semantics and nets semantics as given by the simple firing rule.

The simple firing rule, in spite of its failing to represent the nonsequential behaviour of nets, has been for long the standard operational rule for nets, and in fact has proved a useful tool in the analysis of marked nets.

Step firing rule

The natural improvement on the simple firing rule consists in allowing more than one enabled transition to fire in one step. Of course, only *concurrent* transitions are allowed to occur together in the same step. In particular, a transition which is concurrent with itself may occur more than once in a step.

Thus a step will consist of a multiset rather than a set of transitions; a multiset of transitions in T being defined, as usual, as a mapping $f : T \longrightarrow \mathbb{N}$. If f is a multiset on T and $l : T \longrightarrow A$ a labelling on T, we denote by f_l the multiset of labels of f, defined as follows: $\forall a \in A,\ f_l(a) = \sum_{l(t)=a} f(t)$.

The new rule, called *step firing rule*, is given next.

Definition 3.7 (*Step firing rule*) *Let* $N = (S, T, F, M_0, l)$ *be a labelled marked net over* A, M, M' *be markings of* N, *and* $f : T \longrightarrow \mathbb{N}$. *Then:*

- f *is* a step from M to M', *in notation* $M\,[f>\,M'$, *iff:*

  i) *Each transition* t *of* f *is enabled at least* $f(t)$ *times in* M, *namely:* $\forall s \in S\ \ M(s) \geq \sum_{t \in s^\bullet} f(t)$.

  ii) *For any* $s \in S$, $M'(s) = M(s) - \sum_{t \in s^\bullet} f(t) + \sum_{t \in {}^\bullet s} f(t)$.

- *Let* L *be a multiset on* A. *Then the marking* M moves under L *to the marking* M', *in notation* $M \xrightarrow{L} M'$, *if and only if there exists a step* $f : T \longrightarrow \mathbb{N}$ *s.t.* $M\,[f>\,M'$ *and* $f_l = L$.

Computations of nets are now redefined in terms of the relations $\xrightarrow{L}$.

Definition 3.8 *Let* $N = (S, T, F, M_0, l)$ *be a labelled marked net over* A.

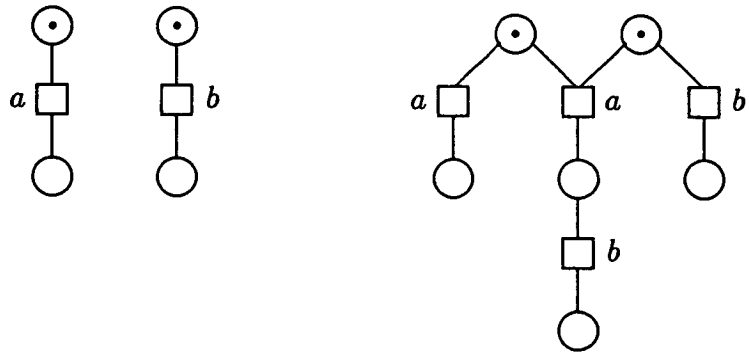
- *A finite or infinite sequence* $L_1, L_2, \ldots$ *of multisets on* A *is a* step sequence *of* N *if and only if there exist markings* $M_1, M_2, \ldots$ *such that* $M_0 \xrightarrow{L_1} M_1 \xrightarrow{L_2} M_2 \ldots$.

- *A* computation *of* N *is any of its step sequences.*

Let us now consider the bisimulation equivalence induced by the relations $\xrightarrow{L}$. This equivalence is stronger than the one induced by the simple firing rule. It

does not identify, for example, the two nets of example 1 above. On the other hand, the two nets below are still equivalent:

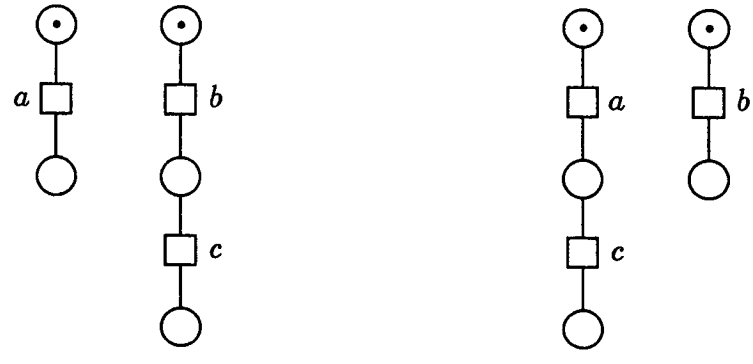
Example 2



Note: the nets of examples 1 and 2 here are just representations of the CL terms of examples 1 and 3 in the previous section. In fact G. Boudol et al. define in [BRS 85] a translation of nets into MEIJE terms, such that step bisimulation semantics for nets coincides with bisimulation semantics for the corresponding MEIJE terms.

Maximal steps. If we restrict ourselves to maximal steps only, whose definition we assume to be intuitively clear, we obtain a bisimulation equivalence which corresponds to the one induced by the rules of the calculus SCCS. The two nets below are for instance identified.

Example 3



Note on the other hand that the new equivalence distinguishes the two nets of example 2, because the left hand one cannot start by executing a transition

a alone. We conclude that this new equivalence is *not* comparable with the previous two ones.

Processes

In closing this section, we must at least mention the notion of *process* of a Petri net. Although processes are not constructed through an operational rule, and have a rather complicated definition based on occurrence nets, they appeared, from the very beginning, to be the "right" notion of behaviour for Petri nets [Petri 77] .

An *occurrence net* is essentially an acyclic net with no branching places. In such a net each transition may occur only once – whence the name, and there are no conflicts between transitions. The idea behind an occurrence net is that its transitions represent occurrences of transitions in a given computation. An occurrence net may then be regarded as a deterministic *unfolding* of some marked net. A *process* associates one of its unfoldings to a net. Formally, a process is a mapping from an occurrence net to a marked net, satisfying a number of properties.

The main advantage of processes with respect to firing rules, is that they seem to preserve all the information about concurrency and causality which is specified in the net.

However, processes as they stand are rather awkward. What seems essential of a process is the partial order of the transitions of the underlying occurrence net, what W. Reisig calls *abstract processes* [Rei 85] . By focussing on the transitions only, one seems to be able to study many relevant behavioural properties of a net. Recently, abstract processes have become – possibly under different names, *pomsets* [Pra 85, Gis 84] , *partial words* [Gra 81] – the object of several studies.

As a matter of fact, the idea of concentrating on the transitions of an occurrence net – although of a more complicated kind – was used already for building the theory of event structures, started by Nielsen, Plotkin and Winskel [NPW 79] and then developed mainly by Winskel [Win 87] .

Let us briefly recall the notion of occurrence net used in [NPW 79] : here an occurrence net is generalised in order to represent a possibly nondeterministic

computation. Thus forward branching is allowed in the net, while the idea that each transition (event) fires at most once is retained from the standard notion.

In occurrence nets, the relations between events may be determined statically: in Winskel's occurrence nets, two events may be causally related, concurrent or in conflict. Event Structures are derived from such occurrence nets by forgetting the conditions and keeping the events together with their relations.

Event structures appear on one side as a simple system model, where the relations of concurrency and causality among events are clearly expressed; on the other side they are close to abstract processes and thus to a behavioural representation of a net. This explains why they have received much attention as a model for concurrency in recent years [CFM 82, Win 82, BC 86, GV 87] .

In the next section we shall be concerned with labelled event structures, and try to extend to them our notion of abstraction homomorphism.

3.2 Labelled Event Structures

We recall here the definition of *Labelled Event Structure* (LES), a simple partial ordering model due to Winskel, Nielsen and Plotkin [NPW 81] . Labelled Event Structures are obtained from Petri nets – more specifically branching occurrence nets – by cancelling the conditions and retaining the events with their existing relations. We recall that in an occurrence net, events are in one of three relations: causal dependency, conflict or concurrency (see [NPW 81] for the definition). In particular, the relation of conflict (resp. concurrency) between causally independent events is determined by the existence (resp. the absence) of a common preceding condition. Henceforth in a LES, where conditions have been removed, the relation of conflict (or the complementary relation of concurrency) will have to be given explicitly.

Roughly speaking, LES's are directed acyclic graphs whose nodes represent events. They can also be regarded as a enrichment of labelled trees, where a new relation of *concurrency* may hold between unconnected events, and *confluence* is allowed after concurrent events. Here and in the rest of this chapter, we shall only be concerned with *finite* LES's. The definition follows.

Definition 3.9 *Let A be a non-empty set. A finite A-labelled event structure (A-LES) is a structure $(E, \leq, \#, \lambda)$ where*

i) *E is a finite set of events*

ii) *$\leq$ is a partial ordering on E, called the* causality *relation*

iii) *$\# \subseteq (E \times E) - (\leq \cup \geq)$ is a symmetric* conflict *relation, satisfying the property of* hereditariness: *$e \# e' \leq e''$ implies $e \# e''$*

iv) *$\lambda : E \longrightarrow A$ is the labelling function*

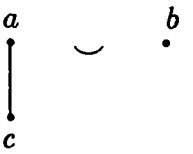
Here concurrency comes in as a derived relation: two events in E are said to be *concurrent*, noted $\smile$, if they are neither comparable nor in conflict. That is to say:

$$\smile =_{def} (E \times E) - (\leq \cup \geq \cup \#)$$

The relation $\smile$ is a symmetric irreflexive relation. Note that by definition the three relations $\leq \cup \geq$, $\#$, and $\smile$ induce a partition upon $(E \times E)$.

Since we have *labelled* structures, we shall mostly be interested in their isomorphism classes. In particular, we will always omit the names of events when drawing LES's. We denote $\rightleftharpoons$ the isomorphism relation on LES's.

In figures we will represent the ordering $\leq$ by arrows (or simple lines, in which case we assume that the order increases downwards) and we show just one of the remaining relations – whichever is most convenient. For instance the following:



is a structure with three events e, e' and e'' respectively labelled a, b and c, such that e causes e'', e, and e' are concurrent and e' and e'' are in conflict.

3.3 Interpretation of terms as LES's.

We shall now impose a structure of Σ-algebra on LES's. We will then have an interpretation of terms of CL as LES's. Similar structured LES's have been considered already by Winskel [Win 82] and for example by U.Goltz in [Gol 86] .

Informally, the unary operator $a:$ inserts a minimal a-labelled event r on top of a LES, while the binary operator $|$ (resp. $+$) juxtaposes two LES's and inserts a relation of concurrency (resp. conflict) between their sets of events.

In fact, there is a straight correspondence between the operators $a:$, $|, +$ of Σ and the connectives $\leq, \smile, \#$ of LES's, and the application of an operator results in an extension of the corresponding connective.

In the definition, we shall use the following notation. If V is a connective in $\{\smile, \#\}$, we write (V) for the corresponding operator. So we have $(\smile) = |$ and $(\#) = +$. Let now S_1, S_2 be LES's, with $S_i = (E_i, \leq_i, \#_i, \lambda_i)$. Then:

Definition 3.10 (*Operators on LES's*)

NIL is the empty LES, with no events

$a:S_1$ is the LES $S = (E, \leq, \#, \lambda)$, where

$$E = E_1 \cup \{r\}\,, \text{ for some } r \notin E_1$$

$$\leq = \leq_1 \cup \{(r, e) \mid e \in E_1\}$$

$$\lambda(e) = \begin{cases} a, & \text{if } e = r \\ \lambda_1(e), & \text{otherwise} \end{cases}$$

If $V \in \{\smile, \#\}$, then $S_1 (V) S_2$ is the LES $S = (E, \leq, \#, \lambda)$, with:

$$E = E_1 + E_2 \quad \text{(disjoint union)}$$

$$\leq = \leq_1 \cup \leq_2$$

$$\# = \begin{cases} (\#_1 \cup \#_2), & \text{if } V = \smile \\ (\#_1 \cup \#_2) \cup \{(e_i, e_j) \mid e_i \in E_i, e_j \in E_j,\ i \neq j,\ i, j \in \{1,2\}\}, & \\ & \text{if } V = \# \end{cases}$$

$$\lambda = \lambda_1 \cup \lambda_2$$

Equipped with these operators, LES's constitute a Σ-algebra, and thus a candidate model for our language CL. For any $t \in CL$, let S_t denote its interpretation as LES. We then have:

$$S_{\text{NIL}} = \text{NIL}$$

$$S_{a:t} = a : S_t$$

$$S_{t+t'} = S_t + S_{t'}$$

$$S_{t|t'} = S_t \,|\, S_{t'}$$

Having fixed an interpretation for terms of CL, we want now to characterise the *interpretation equality* $=_{\text{I}}$ on terms:

$$t =_{\text{I}} t' \quad \text{iff} \quad S_t \rightleftharpoons S_{t'}$$

In fact, it can be easily shown that $=_{\text{I}}$ is the equality generated by the set of axioms:

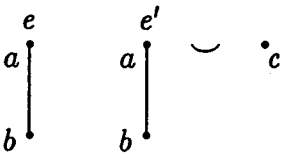
	$A1.$	$x + x' = x' + x$
$sum - laws$	$A2.$	$x + (x' + x'') = (x + x') + x''$
	$A3.$	$x + \mathrm{NIL} = x$
	$P1.$	$x \mid x' = x' \mid x$
$par - laws$	$P2.$	$x \mid (x' \mid x'') = (x \mid x') \mid x''$
	$P3.$	$x \mid \mathrm{NIL} = x$

The proof makes use of *normal forms* of the type $(t\ op\ t')$, where $op \in \{+, \mid\}$, t does not have op as its head operator, and t, t' are themselves normal forms different from NIL. Any term can be reduced to a normal form by means of the axioms A2, A3, P2, P3 (i.e. using associativity to shift parentheses to the right, and the identity laws to eliminate occurrences of NIL in subterms). Such normal forms can be easily associated to LES's which are interpretations of terms. Thereafter one shows that if two such LES's are isomorphic then the corresponding normal forms are equal up to A1, A2, P1, P2.

3.4 Abstraction homomorphisms on LES's

As was noted already, LES's can be regarded as a generalisation of labelled trees. Just like NDP's and labelled trees, Labelled Event Structures provide a basic, concrete model. Again, our intention is to abstract from the basic model to obtain a more interesting equality on terms, which identifies for example the terms t and $(t + t)$.

To this end, we shall try to extend to LES's the abstraction method introduced in Chapter 2. Let us look back at our definition of abstraction homomorphisms for NPS's. The general idea was to simplify the structure of a process by identifying similar states. In the case of NDP's, two states were considered similar if they carried the same label and had similar sets of subsequent states. For LES's we will need a stronger condition, since we want to distinguish for example the events e and e' in the following example:



on the grounds that e and e' have different parallel contexts and thus there is a "computation" that distinguishes them (informally, a computation of a LES S is a finite beginning of S which is free of conflicts, because choices have been solved. A formal definition will not be given until Chapter 6).

To ensure this kind of distinction we shall require that, for each event e, the set of events *concurrent* with e be also preserved by a homomorphism. Namely, if we let $conc(e) =_{def} \{e' \mid e \smile e'\}$, an a.h. h between two LES's S_1 and S_2 will now have to satisfy both the requirements:

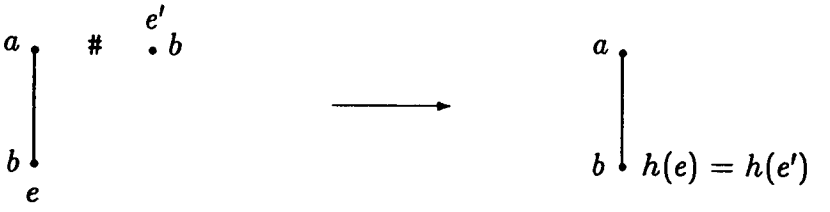
$$succ_2(h(q)) = h(succ_1(q))$$

$$conc_2(h(q)) = h(conc_1(q))$$

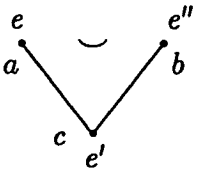
Furthermore, we need to change our condition on labels. For NDP's this was:

$$l_2(h(q)) = l_1(q)$$

where the label $l(q)$ represented the *sequence* of characters leading to q. On the other hand, in a LES events are labelled with simple characters rather than derivation sequences. Therefore the above condition on label preserving is going to be weaker for LES's than it was for NDP's. For example it will allow the simplification below:



As a matter of fact, we cannot really speak of derivation *sequences* for LES's. For consider the LES:



What would be here the "derivation sequence" for the event e' ? This question brings us back to the general problem of finding a notion of derivation for partial ordering models.

For the moment we shall simply get around this problem by keeping characters as labels, and seek some new condition to support the weaker one on labels.

To this purpose, let us consider once again the definition of abstraction homomorphism for NDP's. Note that, for an NDP P, the requirement $l(q) = l(q')$ for merging two states q and q' implies some similarity between their sets of *predecessors*. For NDP's with only observable transitions, this similarity can be stated explicitly as follows. For any state q of P, let $pred(q)$ denote the set of predecessors of q (defined in the obvious way), and $\lambda(q)$ be the last character in the sequence $l(q)$. Thus $\lambda(q)$ is the label of the last transition(s) entering q (note that, because transitions are assumed to be all observable, all transitions entering q must have the same label).

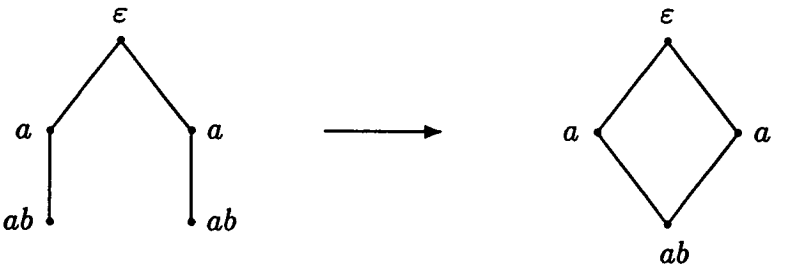
Now we can replace condition $i)$: $l_2(h(q)) = l_1(q)$ in our definition of a.h.'s (see page 22), where $l(q)$ is the derivation sequence for q, by the following two conditions:

a) $\lambda_2(h(q)) = \lambda_1(q)$

b) $pred_2(h(q)) = h(pred_1(q))$

without substantially changing the notion of a.h.. The only change regards confluence: with this new definition, an a.h. can never *introduce* confluence in an NDP, whereas this was possible with the definition of Chapter 2.

Example



The above simplification is not allowed by the new definition. In fact, a) and b) would be the conditions to adopt – rather than *i*) – for defining a.h.'s on labelled trees, in order to preserve their tree-structure.

To come back to our problem, we have now split condition *i*) of a.h.'s into two conditions a) and b). Now a) is just a character-preserving condition, similar to the one we have for LES's. Thus b) seems to be the additional condition we were looking for.

We come in this way to the following definition of abstraction homomorphism for LES's:

Definition 3.11 *If* $S_1 = (E_1, \leq_1, \#_1, l_1)$, $S_2 = (E_2, \leq_2, \#_2, l_2)$ *are LES's, a* surjective *function* $h : E_1 \longrightarrow E_2$ *is an abstraction homomorphism (a.h.) from* S_1 *to* S_2 *iff* $\forall e \in E_1$ *the following hold:*

i) $\lambda_2(h(e)) = \lambda_1(e)$

ii) $pred_2(h(e)) = h(pred_1(e))$

iii) $succ_2(h(e)) = h(succ_1(e))$

iv) $conc_2(h(e)) = h(conc_1(e))$

where:

$pred(e) =_{def} \{e' \mid e' > e\}$ is the set of proper predecessors of e;

$succ(e) =_{def} \{e' \mid e < e'\}$ is the set of proper successors of e;

$conc(e) =_{def} \{e' \mid e \smile e'\}$ is the set of events concurrent with e.

Before giving examples, let us list a few simple facts about a.h.'s. Let MIN(S) = $\{e \in E_S \mid (e' \leq e) \Longrightarrow (e' = e)\}$.

Fact 3.12 *If h is an a.h. from S_1 to S_2, then* $h(\mathrm{MIN}(S_1)) = \mathrm{MIN}(S_2)$.

Proof Uses surjectivity and property *ii)* of h. □

Fact 3.13 *If h is an a.h. from S_1 to S_2, then for any $e, e' \in E_1$:*

i) $e \smile e' \implies h(e) \smile h(e')$

ii) $e \leq e' \implies h(e) \leq h(e')$

iii) $h(e) \smile h(e') \implies \exists\, e'' \smile e' \quad s.t. \quad h(e) = h(e'')$

iv) $h(e) \leq h(e') \implies \exists\, e'' \leq e' \quad s.t. \quad h(e) = h(e'')$ □

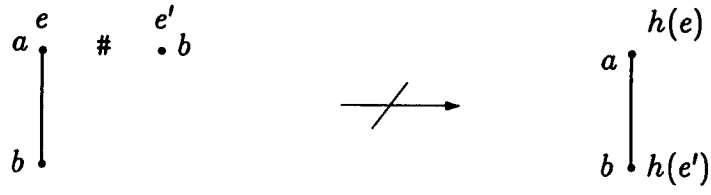
Let us now examine some examples.

Examples

1)

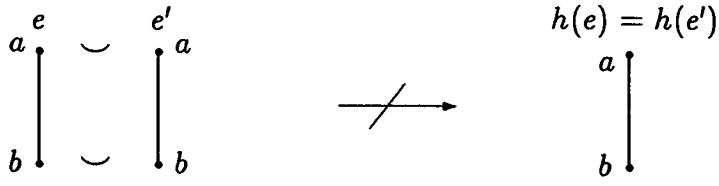
Counterexamples

2)



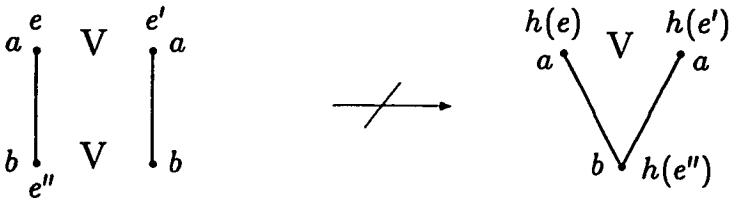
The mapping h here is not an a.h. because it violates condition *ii)* : $h(pred(e')) = \emptyset \neq \{h(e)\} = pred(h(e'))$. It can be noted from this example that condition *ii)* is not implied by condition *i)* (which is satisfied here).

3)



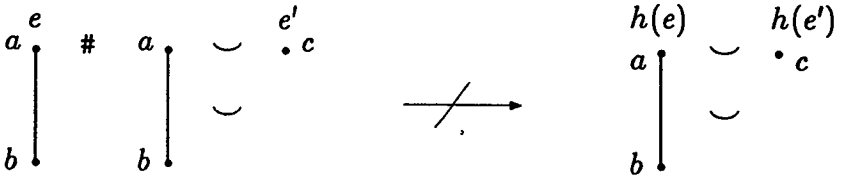
This is not allowed because $h(e') \in h(conc(e)) \neq conc(h(e)) = \emptyset$.

4)



where $V \in \{\#, \smile\}$. This is not allowed since we have for example: $h(pred(e'')) = \{h(e)\} \neq \{h(e), h(e')\} = pred(h(e''))$.

5)



The mapping h here is not an a.h. because $h(conc(e)) = \emptyset \neq h(e') = conc(h(e))$.

Note that we just gave one example of a.h.: the others are all counterexamples. We shall see in fact, in the following section, that the only simplification performed by a.h.'s is of the kind illustrated in ex. 1) (identification of isomorphic conflicting components).

But first, we want to establish some simple properties of a.h.'s. We shall proceed in the same way as we did for NDP's.

We start by defining the *reduction relation* $\xrightarrow{abs}$ on LES's. Let us again write $h : S_1 \longrightarrow S_2$ to indicate that h is an a.h. from S_1 to S_2.

Definition 3.14 $S_1 \xrightarrow{abs} S_2$ *iff* $\exists$ *a.h.* $h : S_1 \longrightarrow S_2$.

We can then easily show the following facts.

Property 3.15 *The relation* $\xrightarrow{abs}$ *is reflexive, transitive and antisymmetric (up to isomorphism).* □

Property 3.16 *The relation* $\xrightarrow{abs}$ *is preserved by the operators* $a:$, | *and* $+$. □

3.5 Characterisation of the abstraction relation

We have at this point an interpretation of terms of CL as LES's, and a reduction relation $S \xrightarrow{abs} S'$ on LES's. We want now to characterise by a set of axioms the relation on terms, which we also denote $\xrightarrow{abs}$, defined as:

$$t \xrightarrow{abs} t' \quad \Longleftrightarrow \quad S_t \xrightarrow{abs} S_{t'}$$

This section will be devoted to such a characterisation. More precisely, we shall give here a system of *reduction rules* which generates the relation $\xrightarrow{abs}$ on terms.

Consider the following set of reduction rules $\mathcal{R}$, where $\longleftrightarrow$ stands for $(\longrightarrow \cap \longrightarrow^{-1})$:

	$RA1.$	$x + x' \longleftrightarrow x' + x$
$sum - laws$	$RA2.$	$x + (x' + x'') \longleftrightarrow (x + x') + x''$
	$RA3.$	$x + \mathrm{NIL} \longrightarrow x$
	$RP1.$	$x \mid x' \longleftrightarrow x' \mid x$
$par - laws$	$RP2.$	$x \mid (x' \mid x'') \longleftrightarrow (x \mid x') \mid x''$
	$RP3.$	$x \mid \mathrm{NIL} \longrightarrow x$
$absorption$	$RABS.$	$x + x \longrightarrow x$

Note that only three of these rules express *proper* reductions, namely $RA3, RP3$ and $RABS$, among which $RABS$ is the only nontrivial one.

Let now $\longrightarrow_{\mathcal{R}}$ be the reduction relation generated by $\mathcal{R}$.

Our purpose is to show the characterisation result:

Theorem 3.17 $t \longrightarrow_{\mathcal{R}} t'$ *iff* $S_t \xrightarrow{abs} S_{t'}$

The *soundness* of the rules $\mathcal{R}$ (that is the *only if* part of the statement) can be proved very easily. To prove the *if* part, i.e. the *completeness* of $\mathcal{R}$, we need as usual to devise appropriate *normal forms* for terms. These should be such that their components are preserved by the relation $\xrightarrow{abs}$ on the corresponding LES's, so that if t, t' are normal forms the proof that $S_t \xrightarrow{abs} S_{t'} \Longrightarrow t \longrightarrow_{\mathcal{R}} t'$

can use induction on the sizes of $S_t, S_{t'}$. Moreover normal forms should bring one closer to semantic interpretation. In the present case, normal forms will be terms taken up to associativity and commutativity of $|$ and $+$, whose proper subterms are all different from NIL. As a matter of fact, such forms are just a notation for *isomorphism classes* of LES's. Formally:

Definition 3.18 *A* normal form *is either* NIL *or one of the following:*

- *A* guarded form $a:t$, *where* t *is a normal form.*
- *A* sumform $\sum_{i\in I, |I|\geq 2} t_i$, *where each* t_i *is a normal form which is either a guarded form or a parform.*
- *A* parform $\prod_{i\in I, |I|\geq 2} t_i$, *where each* t_i *is a normal form which is either a guarded form or a sumform.*

For convenience, we shall use the notation $t = \sum_{i\in I, |I|\geq 2} t_i$ for unspecified normal forms, intending that $t = \text{NIL}$ if $I = \emptyset$ and $t = t_i$ if $|I| = 1$. Also, we will occasionally extend the above terminology – guarded forms, etc. – to (isomorphism classes of) LES's which are interpretations of normal forms. We shall study now how our homomorphisms act on the interpretations of normal forms. It will turn out that, while guarded forms and parforms are preserved by a.h.'s, this is not true in general for sumforms. In other words, a sumform can be simplified to a normal form of a different kind: however its summands will always be preserved.

Our first task will be precisely to show that if $t = \sum_{i\in I, |I|\geq 2} t_i$ is a normal form, and h is an a.h. on S_t, then h preserves the summands S_{t_i} of S_t, and its restriction to any S_{t_i} is still an a.h.. To this purpose, we shall use the fact below, which characterises the relation among events belonging to the same summand. To avoid heavy notations, we adopt the following shorthand:

- If S is a LES, we write $e \in S$ to mean $e \in E_S$.
- If $h : S_1 \longrightarrow S_2$ is an a.h., then $h(S_1)$ denotes the structure S_2, and for any substructure $S_1' = S_1 \restriction E_1'$, $h(S_1')$ denotes the structure $S_2 \restriction h(E_1')$.

Fact 3.19 *Let* $t = \sum_{i\in I, |I|\geq 2} t_i$ *be a sumform, and* S_t *be the corresponding LES. Two events* $e, e' \in \mathrm{MIN}(S_t)$ *belong to the same summand* S_{t_i} *if and only if:*

$$e = e' \quad \text{or} \quad e \smile e' \quad \text{or} \quad \exists e'' \in \mathrm{MIN}(S_t) \quad s.t. \quad e \smile e'' \smile e'$$

Proof. Only if: Let for short $S = S_t$, $S_i = S_{t_i}$. Consider any S_i, and take $e, e' \in (\mathrm{MIN}(S) \cap S_i) = \mathrm{MIN}(S_i)$. Then $e \not< e', e' \not< e$. Two cases are possible:

1) $t_i = a : s$. Then $\mathrm{MIN}(S_i)$ is a singleton and thus $e = e'$ trivially.

2) $t_i = \prod_{j\in J, |J|\geq 2} t_{ij}$. Suppose $e \# e'$: this means that e, e' are in the same component S_{ij}. But then it will be: $e \smile e'' \smile e'$ for any e'' in a different component S_{ik} (there is at least one such S_{ik}, given that $|J| \geq 2$).

If : Let again $e, e' \in \mathrm{MIN}(S)$. Then, if $e = e'$ or $e \smile e'$, the two events e, e' are trivially in the same summand. Suppose now that $e \smile e'' \smile e'$ and $e \# e'$. Then e and e' are in the same summand because they are both in the same summand as e''. □

We are now able to prove the lemma:

Lemma 3.20 *Let* $t = \sum_{i\in I, |I|\geq 2} t_i$ *be a sumform, and* $t' = \sum_{j\in J} t'_j$ *be a normal form. Then any a.h.* $h : S_t \longrightarrow S_{t'}$ *between the corresponding LES's satisfies the following properties:*

i) $\forall i \in \{1, \ldots, n\}\ \exists j \in \{1, \ldots, m\}$ *s.t.* $h(S_{t_i}) = S_{t'_j}$ *(summands are preserved)*

ii) *The restriction of* h *to any summand:* $h_i : S_{t_i} \longrightarrow h(S_{t_i})$ *is still an a.h.*

Proof of i) : Let $S = S_t$, $S' = S_{t'}$, $S_i = S_{t_i}$, $S'_j = S_{t'_j}$. To prove our statement it is enough to show that $\forall i \in \{1, \ldots, n\}\ \exists j \in \{1, \ldots, m\}$ *s.t.* $h(\mathrm{MIN}(S_i)) = \mathrm{MIN}(S_j)$. Then the required result $h(S_i) = S'_j$ will follow, since:

$$\begin{aligned} & E_{S_i} && = \mathrm{MIN}(S_i) \cup succ(\mathrm{MIN}(S_i)) \\ \text{and thus :} \quad & h(E_{S_i}) && = h(\mathrm{MIN}(S_i)) \cup h(succ(\mathrm{MIN}(S_i)) \\ & && = \mathrm{MIN}(S'_j) \cup succ(\mathrm{MIN}(S'_j)) \\ & && = E'_{S_j} \end{aligned}$$

Let us then proceed to prove that $h(\mathrm{MIN}(S_i)) = \mathrm{MIN}(S_j)$. We know from Fact 3.12 that $h(\mathrm{MIN}(S)) = \mathrm{MIN}(S')$. Then, in view of fact 3.19, we may

prove $h(\mathrm{MIN}(S_i)) \subseteq \mathrm{MIN}(S_j)$ by showing that the relations $\smile$ and $\smile \cdot \smile$ are preserved on minimal elements. Let $e, e' \in \mathrm{MIN}(S) \cap S_i$. Fact 3.13 above gives us both $e \smile e' \implies h(e) \smile h(e')$ and $e \smile e'' \smile e' \implies h(e) \smile h(e'') \smile h(e')$. So we can conclude that:

$$\forall i \in \{1, \ldots, n\} \ \exists j \in \{1, \ldots, m\} \ s.t. \ h(\mathrm{MIN}(S_i)) \subseteq \mathrm{MIN}(S_j).$$

We still have to prove the reverse inclusion. Let $h(e), h(e') \in \mathrm{MIN}(S') \cap S'_j$, and suppose $e \in S_i$.

Consider first the simplest case: $h(e) \smile h(e')$. Then, by Fact 3.13 iii), we know that $\exists e'' \ s.t. \ e \smile e''$ and $h(e'') = h(e')$. Since $e \smile e''$, $e'' \in S_i$ by fact 3.19.

Take now the other case: $h(e) \smile h(e'') \smile h(e')$. Here, using Fact 3.13 as above, we deduce first that $\exists \bar{e} \ s.t. \ e \smile \bar{e}$ and $h(\bar{e}) = h(e'')$. Then we have $h(\bar{e}) \smile h(e')$, and by Fact 3.13 again, we deduce now that $\exists \bar{e}' \ s.t. \ \bar{e} \smile \bar{e}'$ and $h(\bar{e}') = h(e')$. To sum up, we have found two events $\bar{e}, \bar{e}'$ s.t. $e \smile \bar{e} \smile \bar{e}'$ and $h(\bar{e}) = h(e'')$, $h(\bar{e}') = h(e')$. Note that by fact 3.19, $e \smile \bar{e} \smile \bar{e}'$ implies that $\bar{e}, \bar{e}' \in S_i$.

We can conclude that in any case if $h(e), h(e') \in \mathrm{MIN}(S') \cap S'_j$ and $e \in S_i$, then there exists $\bar{e}' \in S_i$ such that $h(\bar{e}') = h(e')$. We have thus proved that:

$$\forall j \in \{1, \ldots, m\}, \forall i \in \{1, \ldots, n\} \ s.t. \ h(S_i) \cap S_j \neq \emptyset : \ \mathrm{MIN}(S_j) \subseteq h(\mathrm{MIN}(S_i))$$

and this ends the proof of part i).

Proof of ii) : we want now to show that $h_i =_{def} h \restriction S_i$ is an a.h. from S_i to the corresponding S'_j. We know from part i) that h_i is surjective. As for the other properties, note that $\forall e \in S_i$:

$$\begin{aligned} char_i(e) &= char(e) \\ pred_i(e) &= pred(e) \\ succ_i(e) &= succ(e) \\ conc_i(e) &= conc(e) \end{aligned}$$

where $f_i = f \restriction S_i$. This is because $e' \in S - S_i \implies e \# e'$ and thus $e' \notin pred(e)$, $e' \notin succ(e)$, $e' \notin conc(e)$.

Having established that, it is easy to see that $h_i = h \restriction S_i$ inherits properties 1) – 4) of a.h.'s from h. For example, if $h(S_i) = S'_j$, we have:

$$h_i(pred_i(e) = h(pred(e)) = pred(h(e)) = pred_j(h_i(e)) \qquad \square$$

We shall now prove for guarded forms and parforms a stronger result. Precisely we show that abstraction homomorphisms preserve guarded forms and parforms as well as their immediate components. We start with guarded forms.

Lemma 3.21 *Let $t = a{:}t_0$ be a guarded form, t' be a normal form, and $h : S_t \longrightarrow S_{t'}$ an a.h.. Then:*

i) *t' is a* guarded form.

ii) *the restriction of h to S_{t_0}, $h_0 : S_{t_0} \longrightarrow h(S_{t_0})$ is again an a.h.*

Proof. i): A normal form t is a guarded form if and only if $\mathrm{MIN}(S_t)$ is a singleton. This implies that $h(\mathrm{MIN}(S_t)) = \mathrm{MIN}(S_{t'})$ is a singleton, i.e. that t' is guarded.

Proof. ii): We use again the shorthands $S = S_t$, $S' = S_{t'}$, etc.. Let now e_0 be the unique event in $\mathrm{MIN}(S)$. Then $e'_0 = h(e_0)$ is the unique minimal event of S'. Moreover, if $E_0 = E_{S_0}$, $E'_0 = E'_{S_0}$, we have:

$$h_0(E_0) = h(E_0) = h(succ(e_0)) = succ(h(e_0)) = E'_0$$

So h_0 is surjective.

As for the other properties, note that S_0 inherits the functions *char*, *succ* and *conc* from S exactly as they are, and similarly does S'_0 from S'. What is left to check is that for any $e \in S_0$: $h_0(pred_0(e)) = pred_0{}'(h_0(e))$ (where to be precise we use $pred_0$ to refer to S_0, and $pred'_0$ to refer to S'_0).

Now for any $e \in S_0$: $pred_0(e) = pred(e) - e_0$. Moreover, if $e \in S_0$ then $h(e) \neq e'_0$, because $pred(e) \neq \emptyset$ implies $pred'(h(e)) \neq \emptyset$.

We then have, for any $e \in S_0$:

$$\begin{aligned} h_0(pred_0(e)) &= h(pred(e) - e_0) &&= h(pred(e)) - h(e_0) \\ &= pred'(h(e)) - e'_0 &&= pred_0{}'(h_0(e)) \end{aligned}$$

which is what we wanted to show. We conclude that h_0 is indeed an a.h. from S_0 to S'_0. $\square$

To prove that parforms are preserved by a.h.'s, we need some more definitions. Essentially we need to consider parforms whose components may be parforms again. This is because, as we shall see next, a.h.'s preserve parforms but their components are not necessarily mapped to individual components of the image. Consider the example:

$$[\,(a \mid b) + (a \mid b)\,] \mid c \quad \longrightarrow \quad a \mid b \mid c$$

Here the first parallel component of the left member is mapped to $(a \mid b)$, which is not an individual component of the right member. We thus need a notion of generalised parform.

Convention: We use the notation $||_{i\in I,|I|\geq 2}\, t_i$ to denote a parform whose components t_i may again be parforms. We call $||_{i\in I,|I|\geq 2}\, t_i$ a *generalised parform.*

We then set:

Definition 3.22 *Let S be a LES, and $S_1,\ldots,S_n$ be substructures of S s.t. $\{E_{S_1},\ldots,E_{S_n}\}$ is a partition on E_S. We say that $\{S_1,\ldots,S_n\}$ is a* copartition *on S if and only if:* $(e \in S_i, i \neq j \text{ and } e' \in S_j) \implies (e \smile e')$.

The two concepts are related by the following statement.

Fact 3.23 $t = ||_{i\in I}\, t_i \iff \{S_{t_1},\ldots,S_{t_n}\}$ *is a copartition of* S_t. □

We can now prove our lemma about parforms:

Lemma 3.24 *Let* $t = \prod_{i\in I,|I|\geq 2} t_i$ *be a parform,* t' *be a normal form, and* $h : S_t \longrightarrow S_{t'}$ *an a.h.. Then:*

i) t' *is a* parform.

ii) *the restriction of* h *to any* S_{t_i}, $h_i : S_{t_i} \longrightarrow h(S_{t_i})$ *is an a.h.*

Proof of i): If t is a parform, it is a sumform with just one summand. Thus, using exactly the same reasoning as for lemma 3.20, we can show that this summand is preserved by h. This implies that t' also has a unique summand. Now this summand cannot be a guarded form since then $\mathrm{MIN}(S_{t'})$ would be

a singleton, whereas MIN(S_t) contains at least two elements, say e, e', with $e \smile e'$ implying $h(e) \smile h(e')$ (and thus $h(e) \neq h(e')$). Thus $t' = \prod_{j \in J, |J| \geq 2} t'_j$.

Proof of ii): Let as usual S_i, S'_j stand for S_{t_i}, $S_{t'_j}$ and E_i stand for E_{S_i} etc.. From part i) we know that $t' = \prod_{j \in J, |J| \geq 2} t'_j$. We shall prove now that $\{h(S_i) \mid i \in I\}$ is a *copartition* of S'. We know that:

- $\bigcup_{i \in I} h(S_i) = S'$ (because of h's surjectivity)
- $h(S_i) \cap h(S_j) = \emptyset$ for $i, j \in I$, $i \neq j$, since $\forall e_i \in S_i$, $e_j \in S_j$ we have: $e_i \smile e_j \implies h(e_i) \smile h(e_j)$.

So $\{h(S_i) \mid i \in I\}$ is indeed a *copartition* of S'. It follows from Fact 3.23 that there exists a partition $\{J_i \mid i \in I\}$ of J such that $t' = \mid\mid_{i \in I} t'_{J_i}$ and $S'_{J_i} = h(S_i)$. Supposing $I \cap J = \emptyset$, we rename for simplicity each S'_{J_i} as S'_i, so that for any $i \in I$, $S'_i = h(S_i)$. We want now to show that for any i, the restriction $h_i : S_i \longrightarrow S'_i$ is again an a.h..

Note that, for any $e \in S_i$, we have:

$$\begin{aligned} char_i(e) &= char(e) \\ pred_i(e) &= pred(e) \\ succ_i(e) &= succ(e) \\ conc_i(e) &= conc(e) - \bigcup_{k \neq i} S_k \end{aligned}$$

Similarly for any $e' \in S'_i = h_i(S_i)$. In particular we have:

$$conc'_i(e') = conc'(e') - \bigcup_{k \neq i} h(S_k)$$

It can be easily seen that h_i preserves all the functions f_i. For example:

$$\begin{aligned} h_i(pred_i(e)) &= h(pred(e)) = pred'(h(e)) = pred'_i(h_i(e)) \\ h_i(conc_i(e)) &= h(conc(e)) - h(\textstyle\bigcup_{k \neq i} S_k) = conc'(h(e)) - \bigcup_{k \neq i} h(S_k) \\ &= conc'_i(h_i(e)) \end{aligned}$$

□

We are now ready to prove our completeness result. This relies as usual on a *normalisation* lemma, which we state without proof.

Lemma 3.25 *(Normalisation)*
For any term $t \in CL$ there exists a normal form $\hat{t}$ such that $t \longrightarrow_{\mathcal{R}} \hat{t}$. □

We can then prove the main result of this section.

Theorem 3.26 *(Completeness)*
Let t, t' *be terms of* CL. *Then* $S_t \xrightarrow{abs} S_{t'} \iff t \longrightarrow_{\mathcal{R}} t'$.

Proof: By induction on the sum of the sizes of S_t, $S_{t'}$. Notation: we let $S = S_t$, $S' = S_{t'}$, $S_i = S_{t_i}$, etc. In view of the normalisation lemma, and given the soundness of the axioms, we can assume t, t' to be normal forms.

We need distinguish three cases:

1) $t = a{:}t_0$. Here we know, by lemma 3.21 , that t' will also be of the form $a{:}t'_0$, and that the restriction $h_0 =_{def} h \restriction S'_0$ is still an a.h.. We can then apply induction to get $t_0 \longrightarrow_{\mathcal{R}} t'_0$, and thus, by substitutivity of $\longrightarrow_{\mathcal{R}}$, $a{:}t_0 \longrightarrow_{\mathcal{R}} a{:}t'_0$.

2) $t = \prod_{i \in I, |I| \geq 2} t_i$. Here, by lemma 3.24 and fact 3.23 we know that there exists a generalised parform $t' = \|_{i \in I, |I| \geq 2} t'_i$ such that $S'_i = h(S_i)$ and the restriction $h_i : S_i \longrightarrow S'_i$ is an a.h.. Again, since $|I| \geq 2$, we can apply induction to get $t_i \longrightarrow_{\mathcal{R}} t'_i$. Whence, using substitutivity of $\longrightarrow_{\mathcal{R}}$ together with rules $RP1, RP2$, we obtain $t \longrightarrow_{\mathcal{R}} t'$.

3) $t = \sum_{i \in I, |I| \geq 2} t_i$. Then, by lemma 3.20, there are two possibilities for $t' = \sum_{j \in J} t'_j$:

i) t' is a proper sumform, that is $|J| \geq 2$. Then for any $i \in I$ there exists a $j \in J$ s.t. $h(S_i) = S'_j$ and $S_i \xrightarrow{abs} S'_j$.

ii) t' has only one summand. Then for any $i \in I$ we have $h(S_i) = S'$ and $S_i \xrightarrow{abs} S'$.

Case i): Take $i \in I$, and assume $h(S_i) = S'_j$ and $S_i \xrightarrow{abs} S'_j$. By induction we have: $t_i \longrightarrow_{\mathcal{R}} t'_j$. Define now, for any $j \in J$, the set $I_j =_{def} \{i \in I | h(S_i) = S'_j\}$. Note that $\{I_j \mid j \in J\}$ is a partition on I. Then, using substitutivity of $\longrightarrow_{\mathcal{R}}$ and the absorption rule $RABS$, we obtain:

$$\sum_{i \in I_j} t_i \longrightarrow_{\mathcal{R}} \underbrace{(t'_j + \cdots + t'_j)}_{I_j \ times} \longrightarrow_{RABS} t'_j$$

and thus, by substitutivity again:

$$t = \sum_{i\in I, |I|\geq 2} t_i = \sum_{i\in I_j, j\in J} t_i \longrightarrow_{\mathcal{R}} \sum_{j\in J} t'_j = t'.$$

Case ii): Since $h(S_i) = S'$ and $S_i \xrightarrow{abs} S'$ for all $i \in I$, and the size of each S_i is smaller than the size of S, we can apply induction to get for any $i \in I$ $t_i \longrightarrow_{\mathcal{R}} t'$. Whence, by substitutivity of $\longrightarrow_{\mathcal{R}}$:

$$t = \sum_{i\in I, |I|\geq 2} t_i \longrightarrow_{\mathcal{R}} \underbrace{(t' + \cdots + t')}_{|I| \ times} \longrightarrow_{RABS} t'$$

which is what we wanted to show. □

We have thus established that our reduction relation on LES's is a rather strong one: the only proper reductions (implying a reduction of the number of events) it performs on LES's are those corresponding to the absorption law: $x + x \longrightarrow x$.

We define now an abstraction equivalence $\sim_{abs}$ on LES's, as follows:

Definition 3.27 $\sim_{abs} =_{def} [\xrightarrow{abs} \cup \xleftarrow{abs}]^*$

A corollary of our characterisation theorem is that the equivalence $\sim_{abs}$ is the congruence generated by:

	$A1.$	$x + x' = x' + x$
$sum - laws$	$A2.$	$x + (x' + x'') = (x + x') + x''$
	$A3.$	$x + \mathrm{NIL} = x$
$absorption\ law$	$A4.$	$x + x = x$
	$P1.$	$x \mid x' = x' \mid x$
$par - laws$	$P2.$	$x \mid (x' \mid x'') = (x \mid x') \mid x''$
	$P3.$	$x \mid \mathrm{NIL} = x$

Note that we have not defined $\sim_{abs}$ as reducibility to a same LES here. In fact, we shall not develop the study of $\sim_{abs}$ as we did in chapter 2 here.

We want to establish first whether there is an operational counterpart for $\sim_{abs}$. We know from Chapter 2 that a.h.'s are closely related to bisimulations, We may then try now to derive an operational semantics for terms of CL), using our new notion of abstraction homomorphism as a reference.

This new operational semantics will be the subject of next chapter.

Chapter 4

Distributed Bisimulations

In this chapter we propose an interpretation of concurrent processes as *distributed* labelled transition systems (DLTS's). In such systems, each transition gives rise to a compound residual, made out of a *local* component and a remote *concurrent* component. This description reflects a view of concurrent processes as distributed in space – whence the name, and is based on the same intuition as our abstraction homomorphisms for labelled event structures.

Distributed LTS's give a more intensional description of processes than do usual LTS's (like those examined in the previous chapter). In particular, in a DLTS the global residual after a transition may be retrieved by combining the two – local and concurrent – residuals. On the other hand separating the components allows us, intuitively, to distinguish causality – relating an action to its local residual – from concurrency – relating it to the nonlocal residual.

Distributed LTS's will be used to model the simple algebraic language CL introduced in the preceding chapter, whose operators are prefixing, sum and parallel composition.

Based on the new transitions, we define a *distributed bisimulation* equivalence on DLTS's, which acts recursively on both components of the residual. We show that this equivalence preserves the concurrent structure of processes, while allowing some nontrivial identifications among them. In particular, the new equivalence turns out to be weaker (more interesting?) than the one induced by abstraction homomorphisms, although based on a similar intention. In fact, distributed bisimulation allows more complicated absorptions than the one expressed by the idempotence law.

Our semantics can be easily extended to a language with communication. We adopt the communication discipline of CCS: a communication action results from the simultaneous execution of complementary actions by parallel processes. Any such action is denoted by a symbol τ, and cannot contribute to a further communication. For the rest we shall not, in this chapter, try to distinguish communications from other actions. In other words, we study here the *strong* distributed bisimulation, while the corresponding *weak* bisimulation, where actions τ are considered as unobservable, will be treated in the next chapter.

Our distributed LTS's are somewhat nonstandard, in that the result of a transition is not itself a state of the LTS, but rather a pair of states. We thus propose an alternative formulation of our semantics, where each transition yields a unique (global) residual, while an additional information – the local residual – is annexed to the label. This appears to be a more satisfactory description, because the local information varies at each step of execution, just like the executed action, and is used by the bisimulation in a similar way.

We show that the two semantic formulations are equivalent. The latter is more intuitive, and closer to traditional transition system semantics, whereas the first is simpler to manipulate and better suited for conducting proofs.

The algebraic characterisation of our new equivalence is not easy to derive. We propose here a complete finite axiomatisation, which makes use of two auxiliary operators, an asymmetric parallel operator $\between$ and a communication operator $\mid_c$.

4.1 Distributed transition systems

In this section we introduce the basics of our formalism. We shall keep to the approach of algebraic calculi of processes, as was delineated in the previous chapter.

We start by defining syntax and transition rules for processes. To begin with, let us take the simple syntax Σ considered already:

$$\Sigma = A \cup \{\mathrm{NIL}, +, \mid\}$$

Processes are terms of $\mathcal{P} = T_\Sigma$. We we shall prefer to use the name $\mathcal{P}$ when treating processes as computational objects. Let $p, q, r, \ldots$ range over $\mathcal{P}$, and

$a, b, \ldots$ denote actions of A. We will often abbreviate $a:p$ to ap, and $a:\mathrm{NIL}$ simply to a.

As anticipated in the introduction, we shall interpret processes as *distributed transition systems.* Before formalising this notion, let us illustrate it with a simple example. Consider the parallel process $p \mid q$, where $p = a:p'$. With the operational interpretation of CCS, we have:

$$p \mid q \xrightarrow{a} p' \mid q$$

We aim here at giving a more "distributed" description of a parallel process. We regard $p \mid q$ as consisting of two independent subprocesses p and q, which are placed at different localities, say l_1 and l_2. Then the action a above takes place at locality l_1, yielding the *local residual* p'. The process q is independent from a, and may evolve concurrently with it: we say that q is the *concurrent residual* of the transition. We describe this situation by means of an arrow:

$$p \mid q \xrightarrow{a} <p', q>$$

We may then hope, by examining the two residuals separately, to achieve a distinction between causality and concurrency.

Let us now give the set of rules specifying the behaviour of processes of $\mathcal{P}$ as distributed transition systems. For any $a \in A$, let $\xrightarrow{a}$ be the least relation in $\mathcal{P} \times (\mathcal{P} \times \mathcal{P})$ satisfying the following set of rules $\mathcal{R}$.

RULES $\mathcal{R}$

R1. $a:\ p \xrightarrow{a} <p, \mathrm{NIL}>$

R2. $p \xrightarrow{a} <p',\ p''>$ implies $p + q \xrightarrow{a} <p',\ p''>$

$q + p \xrightarrow{a} <p',\ p''>$

R3. $p \xrightarrow{a} <p',\ p''>$ implies $p \mid q \xrightarrow{a} <p',\ p'' \mid q>$

$q \mid p \xrightarrow{a} <p',\ q \mid p''>$

As mentioned already, the interpretation for $p \xrightarrow{a} <p', p''>$ is that p may perform an action a, thereby producing a local residual p' and a concurrent

residual p''. Rule **R1** states that a term $a:p$ has a transition $\xrightarrow{a}$ yielding a local residual p and a concurrent residual NIL: nothing can happen in parallel with the action a. Rule **R2** is the ordinary rule for sum. The most interesting rule is **R3**, which adds new components to the concurrent residual. It follows from this rule that the concurrent residual of a transition $p \xrightarrow{a} <p', p''>$ is a term of the form $p_1 | \cdots | p_n$, where one of the p_i, equal to NIL, has been inserted at the very first step by rule **R1**, while all the others have been introduced by successive applications of rule **R3**.

Let us examine some examples.

Examples

$$
\begin{aligned}
1)\quad ab + (a|b) \quad &\xrightarrow{a} <b, \mathrm{NIL}> \\
&\xrightarrow{a} <\mathrm{NIL}, \mathrm{NIL} \mid b> \\
&\xrightarrow{b} <\mathrm{NIL}, a|\mathrm{NIL}>
\end{aligned}
$$

$$
\begin{aligned}
2)\quad (ab|c + d)|e \quad &\xrightarrow{a} <b, (\mathrm{NIL}\,|c)|e> \\
&\xrightarrow{c} <\mathrm{NIL}, (ab \mid \mathrm{NIL})|e> \\
&\xrightarrow{d} <\mathrm{NIL}, \mathrm{NIL}\,|e> \\
&\xrightarrow{e} <\mathrm{NIL}, (ab|c + d)| \mathrm{NIL}>
\end{aligned}
$$

The first example shows the difference between a transition $\xrightarrow{a}$ coming from a sequential subterm ab and an homonymous transition coming from a parallel subterm $a|b$: in the first case the concurrent residual is NIL, whereas in the latter it contains the parallel component b.

Looking at the second example, on the other hand, one may remark that the concurrent residual of a transition $p \xrightarrow{a} <p', p''>$ – taken modulo simplification of NIL components – is not in general a subterm of p, whereas the local residual always is. The reason is that building the concurrent residual may imply the resolution of some choices. This point is in fact rather important, and will be raised again in the last section of this chapter, where we discuss the algebraic characterisation of our behavioural equivalence. In the next section we shall give the definition of this equivalence and examine some of its properties.

4.2 Distributed bisimulation equivalence

In this section we define a behavioural equivalence on $\mathcal{P}$, called *distributed* because at each step it compares both the local and the concurrent residuals of two processes. Its definition is based on the notion of *distributed bisimulation*, which we introduce first.

Definition 4.1 *A distributed bisimulation* (d-bisimulation) *is a relation* $R \subseteq (\mathcal{P} \times \mathcal{P})$ *satisfying, for any* $(p,q) \in R$, *the following property of* d-invariance:

i) $p \xrightarrow{a} <p',p''> \implies q \xrightarrow{a} <q',q''>$, *with* $p' \,R\, q'$ *and* $p'' \,R\, q''$

ii) $q \xrightarrow{a} <q',q''> \implies p \xrightarrow{a} <p',p''>$, *with* $p' \,R\, q'$ *and* $p'' \,R\, q''$

If we let $D(R)$ *be the set of all pairs* (p,q) *satisfying clauses* i) *and* ii), *we may summarise the above definition as: a relation* $R \subseteq (\mathcal{P} \times \mathcal{P})$ *is a d-bisimulation iff* $R \subseteq D(R)$.

Now our behavioural equivalence over $\mathcal{P}$, which we call *distributed bisimulation equivalence* and denote by $\sim_d$, is defined to be the union of all d-bisimulations:

Definition 4.2 *Let* $p,q \in \mathcal{P}$ .Then $p \sim_d q$ *iff* $\exists$ *d-bisimulation* R *s.t.* $p \,R\, q$.

Note that, apart from the separate recursion on the two parts of the residual, the definitions of d-bisimulation and d-bisimulation equivalence do not otherwise differ from the usual definitions. In fact it is easy to check that the relation $\sim_d$ satisfies the following properties, which we state without proof.

Property 4.3

i) $\sim_d$ *is an equivalence relation.*

ii) $\sim_d$ *is the largest solution of* $R \subseteq D(R)$ $(and\ R = D(R))$.

By virtue of Property 4.3 ii), we can use for $\sim_d$ the same convenient proof rule as for standard bisimulation equivalences: to prove that $p \sim_d q$, it is enough to construct a d-bisimulation R such that $p \,R\, q$, since then we have $R \subseteq \sim_d$ because $\sim_d$ is the largest d-bisimulation. In what follows, we shall often avail

of this proof technique. As a first application, we use it to show that $\sim_d$ is preserved by the operators of Σ, namely:

Property 4.4 *$\sim_d$ is a Σ-congruence.*

Proof: Trivial for the operators $a:$ and $+$. To show that $\sim_d$ is preserved by the operator $|$, we seek a d-bisimulation R such that $(p|r, q|r) \in R$ whenever $p \sim_d q$. Now it is immediate to check that the relation:

$$R = \{ (p|r, q|r) \mid p \sim_d q \} \cup \sim_d$$

is such a d-bisimulation. The proof that $(r|p, r|q) \in \sim_d$ is of course symmetric. □

Let us consider some examples.

Example 3) $\quad a\,b + (a|b) \quad \not\sim_d \quad (a|b)$

These two processes are not equivalent because the first one has a transition $\xrightarrow{a}$ with concurrent residual NIL , while the the second has a unique transition $\xrightarrow{a}$ with a concurrent residual obviously not equivalent to NIL (the transitions of the processes are described in Ex. 1 above).

Example 4) $\quad (a|b) + (a|b) \quad \sim_d \quad (a|b)$

This is just an ordinary absorption, the typical identification effected by bisimulation equivalences. However Example 3) suggests that $\sim_d$ may be more discriminating than the (strong) bisimulation equivalence $\sim$ of CCS. We shall see in fact, in in section 4.11, that $\sim_d$ is strictly included in $\sim$.

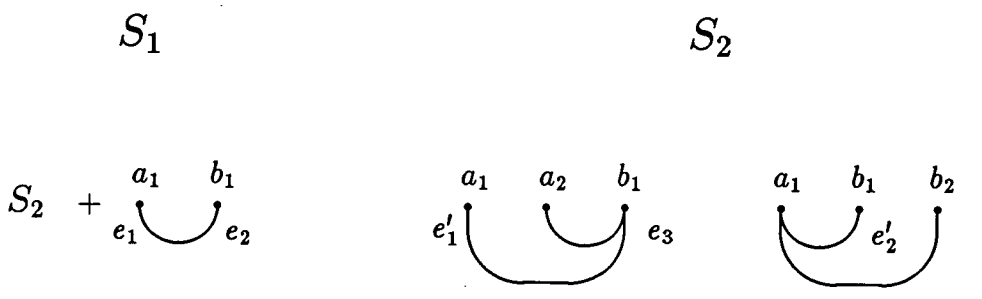
Note on the other hand that both examples 3) and 4) are valid for the abstraction equivalence $\sim_{abs}$ considered in the previous chapter. The obvious question is now: do the two equivalences $\sim_{abs}$ and $\sim_d$ coincide on our language T_Σ? We answer by giving a counterexample to $\sim_d \implies \sim_{abs}$.

Example 5)

$$(a_1 + a_2)|b_1 + a_1|(b_1 + b_2) + (a_1|b_1) \quad \sim_d \quad (a_1 + a_2)|b_1 + a_1|(b_1 + b_2)$$

One may easily check that the two transitions $\xrightarrow{a_1}$ and $\xrightarrow{b_1}$ of the additional term $(a_1 \mid b_1)$ in the left member are matched respectively by the first and the second term in the right member.

Note on the other hand that this identification is not allowed by the equivalence $\sim_{abs}$. For consider the corresponding labelled event structures S_1 and S_2, where for sake of space we represent S_1 as $S_2 + S_1'$:



We may envisage here to reduce S_1 to S_2 by mapping the part S_2 of S_1 isomorphically to S_2, and the additional events e_1, e_2 to e_1', e_2' respectively. However the resulting mapping h is not an abstraction homomorphism because:

$$h(conc(e_1)) = \{h(e_2)\} = \{e_2'\} \neq \{e_3\} = conc(e_1') = conc(h(e_1))$$

This example shows that the equivalence $\sim_{abs}$ is more *intensional* than our d-bisimulation equivalence $\sim_d$. Intuitively, this is because abstraction homomorphisms – as well as the functions $conc, succ$, etc – are defined globally on a fixed set of events, whereas bisimulations are defined recursively on *isomorphism classes* of LES's (i.e. representations of terms). Let us try to explain this point better. Consider the equivalence $\sim_d$ on LES-representations of terms:

$$S_{t_1} \sim_d S_{t_2} \quad \text{iff} \quad t_1 \sim_d t_2$$

Note moreover that each occurrence of an event e in S_t corresponds to a transition of (some subterm of) t. Then $S \upharpoonright succ(e)$ represents the local residual of the transition, and $S \upharpoonright conc(e)$ its concurrent residual.

Now consider the occurrence of e_1 in S_1. Then S_2 may match this with the occurrence of e_1'. It is easy to see that the two pairs of residuals are equivalent:

$$S_1 \upharpoonright succ(e_1) \ \sim_d \ S_2 \upharpoonright succ(e_1')$$

$$S_1 \upharpoonright conc(e_1) \ \sim_d \ S_2 \upharpoonright conc(e_1')$$

whereas for the mapping h considered above we had:

$$h(conc(e_1)) \neq conc(e_1')$$

We have thus established that $\sim_d \not\subseteq \sim_{abs}$. We will show now, on the other hand, that $\sim_{abs} \subseteq \sim_d$. For consider the set of properties:

	$A1.$	$x + x' = x' + x$
$sum - laws$	$A2.$	$x + (x' + x'') = (x + x') + x''$
	$A3.$	$x + \mathrm{NIL} = x$
$absorption\ law$	$A4.$	$x + x = x$
	$P1.$	$x \mid x' = x' \mid x$
$par - laws$	$P2.$	$x \mid (x' \mid x'') = (x \mid x') \mid x''$
	$P3.$	$x \mid \mathrm{NIL} = x$

We know from the previous chapter that $\sim_{abs}$ is precisely the congruence induced by these laws. Now it is easy to see that these laws are *sound* for $\sim_d$, namely:

Property 4.5 *The equivalence* $\sim_d$ *satisfies properties* $A1 - A4$, $P1 - P3$.

Proof: Let $p = q$ be an instance of one of the laws $A1 - A4$. Then (p, q) belongs to the relation: $R = \{(p, q)\} \cup \mathrm{Id}$, (where Id is the identity relation on $\mathcal{P}$) which is obviously a d-bisimulation. Whence $p \sim_d q$.

Let now $p = q$ be an instance of Pi, for $i \in \{1, 2, 3\}$. Then the relation R_i, where:

$$R_1 = \{(r \mid s,\ s \mid r) \mid r, s \in \mathcal{P}\} \cup Id$$

$$R_2 = \{(r \mid (s \mid u),\ (r \mid s) \mid u) \mid r, s, u \in \mathcal{P}\} \cup Id$$

$$R_3 = \{(r \mid \mathrm{NIL},\ r) \mid r \in \mathcal{P}\} \cup Id$$

is a d-bisimulation containing (p, q). Thus $p \sim_d q$, and this ends the proof. □

In conclusion, the relation between $\sim_{abs}$ and $\sim_d$ sums up to:

Corollary 4.6 $\sim_{abs} \subset \sim_d$ ($\sim_{abs}$ *is strictly contained in* $\sim_d$).

Proof: Since $\sim_d$ is a congruence, it follows from property 4.5 that $\sim_{abs} \subseteq \sim_d$. On the other hand example 5) shows that $\sim_{abs} \neq \sim_d$. □

Before ending this section, let us recapitulate what we have obtained so far concerning our language CL. We have defined two notions of equivalence on CL: the first, $\sim_{abs}$, is based on the system model of labelled event structures, and seems to be the strongest possible equivalence one can adopt (as long as one does not want to see the names and the number of locations); the other, $\sim_d$, is based on an operational criterion, and corresponds to a more abstract concept of behaviour. In particular, $\sim_d$ allows some interesting interferences between sum and parallel composition.

Hence, although we have failed in our search for an operational counterpart of $\sim_{abs}$, we have obtained a new notion of equivalence which seems worth investigating. In the rest of this chapter we will no longer be concerned with $\sim_{abs}$. Instead, we shall undertake a detailed study of $\sim_d$, for an extended version of CL including communication.

4.3 Adding communication

In this section we extend the definitions of distributed transition system and d-bisimulation to deal with communication. To add this new feature to our calculus, we simply reinterpret the parallel operator $|$ so that its arguments may synchronise on complementary actions. More precisely, we adopt the communication model of CCS: we assume our set of actions A to be of the form $\Lambda \cup \bar{\Lambda}$, where Λ is some set of labels and $\bar{\Lambda} = \{\bar{a} \mid a \in \Lambda\}$ is the set of their complements. By convention $\bar{\bar{a}} = a$, for any $a \in A$. Now communication is defined to be the simultaneous execution of two complementary actions. A communication action is denoted by a special symbol τ, and is the only action not to have a complement. Thus a τ cannot be used for a further synchronisation: communication is always twofold. The new rule for the operator $|$ is the following:

$$p \xrightarrow{a} <p',\ p''>,\ q \xrightarrow{\bar{a}} <q', q''> \quad \text{imply} \quad p|q \xrightarrow{\tau} <p'|q',\ p''|q''>$$

If we let A_τ stand for $A \cup \{\tau\}$, our syntax for processes remains practically unchanged:

$$\Sigma' = A_\tau \cup \{\text{NIL}, +, |\}$$

Let $\mathcal{P}' = T_{\Sigma'}$ denote our new class of processes. The transitions of processes of $\mathcal{P}'$ are specified by the rules $\mathcal{R}'$, where the new rule for $|$ has been inserted, and the labels a, μ of transitions range respectively over A and A_τ.

RULES $\mathcal{R}'$

R1. μ: $p \xrightarrow{\mu} <p, \text{NIL}>$

R2. $p \xrightarrow{\mu} <p',\ p''>$ implies $p+q \xrightarrow{\mu} <p',\ p''>$, $q+p \xrightarrow{\mu} <p',\ p''>$

R3a. $p \xrightarrow{\mu} <p',\ p''>$ implies $p|q \xrightarrow{\mu} <p',\ p''|q>$, $q|p \xrightarrow{\mu} <p',\ q|p''>$

R3b. $p \xrightarrow{a} <p', p''>,\ q \xrightarrow{\bar{a}} <q', q''>$ imply $p|q \xrightarrow{\tau} <p'|q',\ p''|q''>$

Note that **R3b.** is the only rule where both components contribute to the local residual. This indicates that, as long as they are involved in a communication, two parallel components are seen as forming just *one* local process.

We can now redefine d-bisimulations in terms of the new transitions $p \xrightarrow{\mu} <p', p''>$. We shall still denote by $\sim_d$ the resulting equivalence. It is easy to check that properties 4.3 and 4.4 continue to hold for $\sim_d$ in our extended calculus.

As is to be expected, on the other hand, the relation $\sim_d$ allows now new identifications between processes. We have for example:

Example 6) $\quad (a\,b \mid \bar{a}\,c) \quad \sim_d \quad (a\,b \mid \bar{a}\,c) \;+\; \tau\,(b \mid c)$

We shall see more examples later in this chapter, when discussing the algebraic properties of $\sim_d$.

4.4 Alternative formulations of the semantics

We have established a behavioural equivalence on processes of $\mathcal{P}'$, based on the transitions $p \xrightarrow{\mu} <p', p''>$. Note however that the rules $\mathcal{R}'$ do not really provide an execution model for processes, since they only describe their first transitions. We have not specified how a process continues its execution after giving rise to a compound residual $<p', p''>$. We may now remedy this deficiency by adding the following rules for pairs $<p, q>$ of processes.

R4a. $p \xrightarrow{\mu} <p',\ p''>$ implies $<p, q> \xrightarrow{\mu} <p',\ p'' \mid q>$

$<q, p> \xrightarrow{\mu} <p',\ q \mid p''>$

R4b. $p \xrightarrow{a} <p', p''>,\ q \xrightarrow{\bar{a}} <q', q''>$ imply $<p, q> \xrightarrow{\tau} <p' \mid q',\ p'' \mid q''>$

According to these rules, after each transition $p \xrightarrow{\mu} <p', p''>$ the execution resumes with the composition of the two residuals: the pair $<p',\ p''>$ has exactly the same behaviour as the process $p' \mid p''$. In fact we will show now that $p' \mid p''$ coincides, up to a rearrangement of the components, with the *global* residual of the corresponding CCS transition.

We have the following result – where we use the same arrows $\xrightarrow{\mu}$ for CCS transitions for ours, since they may be distinguished by the form of their residuals (see Appendix for the definition of CCS transitions on $T_{\Sigma'}$):

Fact 4.7 (*Conversion*) *Let* $p \in \mathcal{P}'$. *Then:*

1. $p \xrightarrow{\mu} <p', p''> \implies \exists\, q \equiv p' \mid p''$ *s.t.* $p \xrightarrow{\mu} q$.
2. $p \xrightarrow{\mu} q \implies \exists\, p', p''$ *s.t.* $q \equiv p' \mid p''$ *and* $p \xrightarrow{\mu} <p', p''>$.

where $\equiv$ *is the congruence induced by the laws P1 - P3 at p. 97.*

Proof: By induction on the length of the proof of transitions. □

We may now extend the definition of d–bisimulation to pairs of processes. We redefine our class of semantic objects to be $\mathcal{P}'' =_{def} \mathcal{P}' \cup \{(p,q) \mid p, q \in \mathcal{P}'\}$, and let P, Q, etc. range over $\mathcal{P}''$. We then assume our extended equivalence – which we still denote by $\sim_d$ – to be the union of all (extended) d–bisimulations defined as follows:

Definition 4.8 *An extended d–bisimulation is a relation* $R \subseteq (\mathcal{P}'' \times \mathcal{P}'')$ *satisfying, for any* $(P,Q) \in R$ *and* $\mu \in A_\tau$, *the following property:*

i) $P \xrightarrow{\mu} <p', p''> \implies Q \xrightarrow{\mu} <q', q''>$, *with* $p' \,R\, q'$ *and* $p'' \,R\, q''$

ii) $Q \xrightarrow{\mu} <q', q''> \implies P \xrightarrow{\mu} <p', p''>$, *with* $p' \,R\, q'$ *and* $p'' \,R\, q''$

It is then easy to show that for any processes p, q in $\mathcal{P}'$ the pair $<p, q>$ is bisimilar to the process $p \mid q$.

Fact 4.9 $\forall\, p, q \in \mathcal{P}' : \quad <p, q> \;\sim_d\; p \mid q$.

Proof: The following is a d–bisimulation:

$$R = \{\, (<p, q>,\, p \mid q) \mid p, q \in \mathcal{P}'' \,\} \;\cup\; \sim_d$$

□

Now it is clear that adding a condition $<p', p''> \; R \; <q', q''>$ to each clause would not affect the resulting equivalence. In fact $\sim_d$ is a congruence, and thus: $p' \sim_d q'$, $p'' \sim_d q''$ imply $<p', p''> \sim_d p' \mid p'' \sim_d q' \mid q'' \sim_d <q', q''>$.

To sum up, in defining d-bisimulations we may dispense entirely with the new rules **R4**. In other words, we may dispense from looking at the "global residual" once we have examined the local and concurrent residuals.

An interesting question, on the other hand, is whether we can *replace* one of the two (local and concurrent) residuals by the global residual, and still obtain an equivalent semantics.

In order to answer this question, we introduce now a new semantic definition for processes. Essentially, this is obtained from our original definition at p. 92 by substituting the *global residual* for the concurrent one. However, the new transitions will be presented in a different notation, closer to the standard: the result of a transition will just be its global residual, while the local residual appears as an additional information associated to the label. A typical transition will have the form: $p \xrightarrow{\mu, p'} p''$.

Let us now give the rules specifying the new semantics. For any $\mu \in A_\tau$, we define $\xrightarrow{\mu}$ to be the least relation on $\mathcal{P} \times (\mathcal{P} \times \mathcal{P})$ satisfying the rules $\mathcal{U}$.

RULES $\mathcal{U}$

U1. μ: $p \xrightarrow{\mu, p} p$

U2. $p \xrightarrow{\mu, p'} p''$ implies $p + q \xrightarrow{\mu, p'} p''$

$q + p \xrightarrow{\mu, p'} p''$

U3a. $p \xrightarrow{\mu, p'} p''$ implies $p \mid q \xrightarrow{\mu, p'} p'' \mid q$

$q \mid p \xrightarrow{\mu, p'} q \mid p''$

U3b. $p \xrightarrow{a, p'} p''$, $q \xrightarrow{\bar{a}, q'} q''$ imply $p \mid q \xrightarrow{\tau, p' \mid q'} p'' \mid q''$

As we said already, the residual p'' of a transition $p \xrightarrow{\mu, p'} p''$ records the global evolution of the process p after an atomic action μ. Now this is precisely what CCS interleaving semantics describes (again, we refer to the Appendix for the

definition of CCS semantics). In fact, if we forget about the local residual above the arrow, our rules $\mathcal{U}$ are just Milner's transition rules for CCS.

We have the following (obvious) conversion lemma.

Lemma 4.10 (*Conversion*) *Let* $p \in \mathcal{P}'$. *Then:*

1. $p \xrightarrow{\mu, p'} p'' \implies p \xrightarrow{\mu} p''$.
2. $p \xrightarrow{\mu} p'' \implies \exists p' \ s.t. \ p \xrightarrow{\mu, p'} p''$.

We may now define a new d-bisimulation equivalence $\simeq_d$ on $\mathcal{P}'$ as follows: we set $p \, \mathcal{D}'(\mathcal{R}) q$ if and only if:

i) $p \xrightarrow{\mu, p'} p'' \implies q \xrightarrow{\mu, q'} q''$, *with* $p' \ \mathcal{R} \ q'$ *and* $p'' \ \mathcal{R} \ q''$

ii) $q \xrightarrow{\mu, q'} q'' \implies p \xrightarrow{\mu, p'} p''$, *with* $p' \ \mathcal{R} \ q'$ *and* $p'' \ \mathcal{R} \ q''$

Define $\simeq_d$ to be the largest relation $\mathcal{R}$ such that $\mathcal{R} \subseteq \mathcal{D}'(\mathcal{R})$. Then it is intuitively clear that $\simeq_d$ will be more discriminating than the (strong) bisimulation equivalence $\sim$ of CCS, since it requires an additional condition (similarity of local residuals). We prove next that $\sim$ is an *extension* of $\simeq_d$.

Theorem 4.11 *For any* $p, q \in \mathcal{P}'$:

1. $p \simeq_d q$ *implies* $p \sim q$.
2. *If* p, q *do not contain occurrences of* $|$, *then* $p \sim q$ *implies* $p \simeq_d q$.

Proof: **1.** We show that $\simeq_d$ is a bisimulation relation, and is thus included in the largest bisimulation $\sim$. We use to this end the conversion lemma 4.10. Suppose that $p \simeq_d q$. We want to show that for any CCS transition $p \xrightarrow{\mu} p''$ of p there exists a corresponding transition $q \xrightarrow{\mu} q''$ of q such that $p'' \simeq_d q''$. By the conversion lemma we have: $p \xrightarrow{\mu} p'' \implies \exists p' \ s.t. \ p \xrightarrow{\mu, p'} p''$. Then, since $p \simeq_d q$ there must exist $q', q'' \ s.t. \ q \xrightarrow{\mu, q'} q''$, with $p' \simeq_d q'$ and $p'' \simeq_d q''$. We can now apply the conversion lemma the other way round to get: $q \xrightarrow{\mu, q'} q'' \implies q \xrightarrow{\mu} q''$. Since $p'' \simeq_d q''$, we have proved that $\simeq_d$ is a bisimulation.

2. Let $\mathcal{P}'_{seq}$ be the class of processes which do not use the operator $|$. We want to show that $\sim$ is a d-bisimulation relation on $\mathcal{P}'_{seq}$.

Note that for processes of $\mathcal{P}'_{seq}$ the local and global residuals always coincide, and thus the conversion lemma 4.10 may be strenghtened to:

$$p \xrightarrow{\mu} r \quad \text{if and only if} \quad p \xrightarrow{\mu, r} r$$

Moreover, since r is a subterm of p, it will necessarily belong to $\mathcal{P}'_{seq}$. This is sufficient to prove our statement. For suppose that $p \sim q$, with $p, q \in \mathcal{P}'_{seq}$, and $p \xrightarrow{\mu, r} r$. By the conversion property we have $p \xrightarrow{\mu} r$. Then, since $p \sim q$, there exists s such that $q \xrightarrow{\mu} s$ and $r \sim s$. Now, by conversion again, we have $q \xrightarrow{\mu, s} s$. Since $r \sim s$, we have proved that $\sim$ is a d-bisimulation. □

We will show now that $\simeq_d$ is an alternative formulation for $\sim_d$. To this purpose we need a third (and last) conversion lemma.

Lemma 4.12 (*Conversion*) *Let* $p \in \mathcal{P}'$. *Then:*

1. $p \xrightarrow{\mu} <p', p''> \implies \exists\, q \equiv p' \,|\, p''$ *s.t.* $p \xrightarrow{\mu, p'} q$.

2. $p \xrightarrow{\mu, p'} q \implies \exists\, p''$ *s.t.* $q \equiv p' \,|\, p''$ *and* $p \xrightarrow{\mu} <p', p''>$.

where $\equiv$ *is the congruence induced by the laws P1 - P3 at p. 97.*

Proof: By induction on the length of proofs of transitions. □

This result, together with the synchronisation and simplification lemmas below, will give us the elements to prove that the two equivalences $\sim_d$ and $\simeq_d$ coincide.

Lemma 4.13 (*Synchronisation lemma*) *For any* $p \in \mathcal{P}$:

$$p \xrightarrow{a} <p', p''> \text{ and } p'' \xrightarrow{\bar{a}} <r', r''> \text{ imply } p \xrightarrow{\tau} <p' \,|\, r', r''>.$$

Proof: By structural induction on p. There are only two cases to consider:

i) $p = p_1 + p_2$. Then we get the result by a simple induction on the summand which performs the action.

ii) $p = p_1 \mid p_2$. Suppose that the action a comes from the component p_1, that is $p_1 \mid p_2 \xrightarrow{a} <p', p''>$ because $p_1 \xrightarrow{a} <p', s>$ and $p'' = s \mid p_2$. Now we know that $p'' \xrightarrow{\bar{a}} <r', r''>$. There are then two possibilities, depending on which component of $p'' = s \mid p_2$ performs the action.

a) $s \xrightarrow{\bar{a}} <r', s''>$. Then $r'' = s'' \mid p_2$. Since p_1 meets the hypotheses of the lemma, we may use induction to get $p_1 \xrightarrow{\tau} <p' \mid r',\ s''>$. Whence we deduce $p_1 \mid p_2 \xrightarrow{\tau} <p' \mid r',\ s'' \mid p_2>$, which is the required move since $s'' \mid p_2 = r''$.

b) $p_2 \xrightarrow{\bar{a}} <r', u>$. Then $r'' = s \mid u$. Now $p_1 \xrightarrow{a} <p', s>$ and $p_2 \xrightarrow{\bar{a}} <r', u>$ imply, by the communication rule **R3b**, that $p_1 \mid p_2 \xrightarrow{\tau} <p' \mid r',\ s \mid u>$, which is the move we were after, since $s \mid u = r''$. □

Lemma 4.14 (*Simplification*)

For any $p, q, r \in \mathcal{P}'$: $p \mid r \sim_d q \mid r$ *implies* $p \sim_d q$.

Proof: By induction on the sizes of p, q. The proof makes use of the synchronisation lemma 4.13.

We show that for any move of p there exists a corresponding move of q.

Assume $p \xrightarrow{\mu} <p', p''>$. Then $p \mid r \xrightarrow{\mu} <p',\ p'' \mid r>$. Now $q \mid r$ can match this in three different ways.

1) $q \mid r \xrightarrow{\mu} <q', q'' \mid r>$, with $p' \sim_d q'$ and $p'' \mid r \sim_d q'' \mid r$, because $q \xrightarrow{\mu} <q', q''>$. Then by induction we have $p'' \sim_d q''$, and thus $q \xrightarrow{\mu} <q', q''>$ is the required move of q.

2) $q \mid r \xrightarrow{\mu} <r',\ q \mid r''>$, with $p' \sim_d r'$ and $p'' \mid r \sim_d q \mid r''$, because $r \xrightarrow{\mu} <r', r''>$. We will show now that:

(*) $p'' \mid r \sim_d q \mid r''$ and $r \xrightarrow{\mu} <r', r''>$ imply $q \xrightarrow{\mu} <q', q''>$, with $r' \sim_d q'$ and $p'' \sim_d q''$.

We prove (*) by induction on the number n of actions μ occurring in r.

Basis: $n = 1$. Then $r'' \not\xrightarrow{\mu}$. Thus $q \mid r''$ will have to match the transition $p'' \mid r \xrightarrow{\mu} <r',\ p'' \mid r''>$ by letting the component q move, possibly together with the component r'' if $\mu = \tau$. We have therefore two cases to examine:

a) q moves alone: $q \xrightarrow{\mu} <q', q''>$, and thus $q \mid r'' \xrightarrow{\mu} <q', q'' \mid r''>$, with $r' \sim_d q'$ and $p'' \mid r'' \sim_d q'' \mid r''$. Then by induction on the simplification lemma we get $p'' \sim_d q''$.

b) $\mu = \tau$ and $q \mid r'' \xrightarrow{\tau} <q' \mid s', q'' \mid s''>$, with $r' \sim_d q' \mid s'$ and $p'' \mid r'' \sim_d q'' \mid s''$, because $q \xrightarrow{a} <q', q''>$ and $r \xrightarrow{\bar{a}} <s', s''>$. Now, from $p'' \mid r'' \sim_d q'' \mid s''$ and $r \xrightarrow{\bar{a}} <s', s''>$ we can deduce, by induction, on (*),that $q'' \xrightarrow{\bar{a}} <u', u''>$, with $s' \sim_d u'$ and $p'' \sim_d u''$. By the synchronisation lemma we finally obtain $q \xrightarrow{\tau} <q' \mid u', u''>$, where we know that $r' \sim_d q' \mid s' \sim_d q' \mid u'$ and $p'' \sim_d u''$.

Inductive step: Here $q \mid r''$ can match the move $p'' \mid r \xrightarrow{\mu} <r', p'' \mid r''>$ in three possible ways: a) and b) as in basis case. The new case is:

c) $q \mid r'' \xrightarrow{\mu} <s', q \mid s''>$, with $r' \sim_d s'$ and $p'' \mid r'' \sim_d q \mid r''$, because $r'' \xrightarrow{\mu} <s', s''>$. We can then apply induction to get $q \xrightarrow{\mu} <q', q''>$, with $r' \sim_d q'$ and $p'' \sim_d q''$. This is the required move of q, since $r' \sim_d s' \sim_d q'$ and $p'' \sim_d q''$.

This ends the proof of statement (*). From which it follows that $q \xrightarrow{\mu} <q', q''>$ is the matching move for $p \xrightarrow{\mu} <p', p''>$, since $p' \sim_d r' \sim_d q'$ and $p'' \sim_d q''$.

3) $\mu = \tau$ and $q \mid r \xrightarrow{\tau} <q' \mid r', q'' \mid r''>$, with $p' \sim_d q' \mid r'$ and $p'' \mid r \sim_d q'' \mid r''$, because $q \xrightarrow{a} <q', q''>$ and $r \xrightarrow{\bar{a}} <r', r''>$. Now $p'' \mid r \sim_d q'' \mid r''$ and $r \xrightarrow{\bar{a}} <r', r''>$ imply, by the same reasoning as in part **2)**, that $q'' \xrightarrow{\bar{a}} <s', s''>$, with $r' \sim_d s'$ and $p'' \sim_d s''$. We may now use the synchronisation lemma to obtain $q \xrightarrow{\tau} <q' \mid s', s''>$. Since $p' \sim_d q' \mid r' \sim_d q' \mid s'$ and $p'' \sim_d s''$, this is the required move of q. □

We may now prove the result:

Theorem 4.15 (*Equivalence of the two semantics*)

For any $p, q \in \mathcal{P}'$: $\quad p \sim_d q \iff p \simeq_d q$.

Proof: The proof rests on two lemmas: the above conversion lemma 4.12 and the simplification lemma 4.14.

We recall that $\sim_d$ is the largest solution of $R \subseteq D(R)$. Similarly, if we let $D'(R)$ be the set of pairs (p,q) satisfying clauses $i)$ and $ii)$ (with R replacing $\simeq_d$) at page 103 , we have that $\simeq_d$ is the is the largest solution of $R \subseteq D'(R)$. We shall prove that $\sim_d$ coincides with $\simeq_d$ by showing first $\sim_d \subseteq D'(\sim_d)$ (whence $\sim_d \subseteq \simeq_d$) and then $\simeq_d \subseteq D(\sim_d) = \sim_d$.

1. $\sim_d \subseteq D'(\sim_d)$.

Let $p \sim_d q$. Suppose now that $p \xrightarrow{\mu, p'} r$. By the conversion lemma, there exists p'' such that $r \equiv p' \,|\, p''$ and $p \xrightarrow{\mu} <p', p''>$. Then, since $p \sim_d q$, there must be q', q'' such that $q \xrightarrow{\mu} <q', q''>$, with $p' \simeq_d q'$ and $p'' \simeq_d q''$. We may now use the conversion lemma in the other way to get: $\exists\, s \equiv q' \,|\, q''$ such that $q \xrightarrow{\mu, q'} s$. We know already that $p' \sim_d q'$ and $p'' \sim_d q''$. Moreover, since $\sim_d$ is a congruence and $\equiv \subseteq \sim_d$, we have also: $r \sim_d p' \,|\, p'' \sim_d q' \,|\, q'' \sim_d s$. We conclude that $(p,q) \in D'(\sim_d)$.

2. $\simeq_d \subseteq D(\sim_d) = \sim_d$.

We use here an induction on the sizes of p, q. We need this induction to be able to apply our simplification lemma 4.14.

Let $p \simeq_d q$, and suppose that $p \xrightarrow{\mu} <p', p''>$. By the conversion lemma, there is $r \equiv p' \,|\, p''$ such that $p \xrightarrow{\mu, p'} r$. Since $p \simeq_d q$, there exist q', s such that $q \xrightarrow{\mu, q'} s$, with $p' \simeq_d q'$ and $r \simeq_d s$. By the conversion lemma again, there is q'' such that $s \equiv q' \,|\, q''$ and $q \xrightarrow{\mu} <q', q''>$. Now we know that $p' \simeq_d q'$ and $p' \,|\, p'' \simeq_d r \simeq_d s \simeq_d q' \,|\, q''$. By induction, $p' \simeq_d q'$ and $p' \,|\, p'' \simeq_d q' \,|\, q''$ imply respectively $p' \sim_d q'$ and $p' \,|\, p'' \sim_d q' \,|\, q''$. We may then apply the simplification lemma to get $p'' \sim_d q''$. We conclude that indeed $(p,q) \in D(\sim_d) = \sim_d$.

□

We have thus established that our two semantics, based on the transitions $p \xrightarrow{\mu} <p', p''>$ and $p \xrightarrow{\mu, p'} p''$ respectively, are equivalent as regards the resulting d-bisimulation equivalence. As we just saw, the latter formulation has some advantages over the first: it is closer to standard execution models and allows an easy comparison with CCS interleaving semantics. On the other hand, the first formulation is generally easier to work with: in particular, it is the one we will use in the next section for proving algebraic properties of $\sim_d$.

Note to conclude that, since the local and concurrent residuals play symmetric roles (in composing the global residual), we could as well use the concurrent residual in the definition of $\simeq_d$. As it were, any choice of a pair among the local, concurrent and global residuals appears to lead to the same behavioural equivalence $\sim_d$.

4.5 Algebraic characterisation

The aim of this section is to establish an axiomatisation for the behavioural equivalence $\sim_d$. We know already from previous sections that $\sim_d$ is strictly contained between the abstraction equivalence $\sim_{abs}$ and Milner's strong bisimulation equivalence $\sim$. Hence in order to obtain an axiomatisation of $\sim_d$ we will need to add new laws to those of $\sim_{abs}$.

We noticed as well – in section 4.2 – that the expansion theorem (IN) of CCS, expressing the simulation of concurrency by nondeterministic interleaving:

(IN) If $x = \sum_{i\in I} a_i x_i$, $y = \sum_{j\in J} b_j y_j$, then

$$x \,|\, y = \sum_{i\in I} a_i (x_i \,|\, y) + \sum_{j\in J} b_j (x \,|\, y_j)$$

is not valid for $\sim_d$.

This does not mean, however, that $\sim_d$ will not allow interesting dependencies between the operators. We will have for example the following identifications:

Example 7) $p + (r_1 \,|\, s_1) \sim_d p$ *where*

$$p = (r_1 + r_2) \,|\, s_1 + r_1 \,|\, (s_1 + s_2).$$

Example 8) $q + (r_1 + r_2) \,|\, (s_1 + s_2) \sim_d q$ *where*

$$q = (r_1 + r_2) \,|\, s_1 + (r_1 + r_2) \,|\, s_2 + r_1 \,|\, (s_1 + s_2) + r_2 \,|\, (s_1 + s_2).$$

We can gather here that $\sim_d$ allows more complicated absorptions than the one expressed by the idempotence law A4. Both 7) and 8) are in fact *absorption laws*: for instance 7) states that the term $(r_1 \,|\, s_1)$ may be absorbed into the term $r_1 \,|\, (s_1 + s_2) + (r_1 + r_2) \,|\, s_1$.

As a matter of fact, we can find arbitrarily complex absorption laws, which are all independent. Before giving more examples, let us formalise what we mean by *absorption* of a term into another.

Definition 4.16 *Let* $p \subsetneq_d q \iff (p+q) \sim_d q$.

If $p \subsetneq_d q$, we say that p is *absorbed* into q. It is clear that $p \sim_d q$ if and only if $p \subsetneq_d q$ and $q \subsetneq_d p$.

Thus in the examples above we have $(r_1 \mid s_1) \subsetneq_d p$ and $(r_1+r_2) \mid (s_1+s_2) \subsetneq_d q$. Another example (where we use $\mid$ up to associativity) is:

$$(r_1 \mid s_1 \mid u_1) \subsetneq_d (r_1 + r_2) \mid s_1 \mid u_1 + r_1 \mid (s_1 + s_2) \mid u_1 + r_1 \mid s_1 \mid (u_1 + u_2)$$

and one may easily modify example 8) to a similar but independent one in which $(r_1 + r_2 + r_3) \mid (s_1 + s_2 + s_3)$ is absorbed.

In view of this variety of absorption phenomena, it seems doubtful that there might exist a *finite* set of axioms – or even axiom schemata – which will account for all cases.

We shall get round this difficulty by introducing an auxiliary operator $\between$. In order to understand the role of this operator, we must look back at our behavioural equivalence $\sim_d$ (here and throughout this section we shall refer to the equivalence $\sim_d$ based on the transitions $p \xrightarrow{\mu} <p', p''>$).

We recall that, whenever $p \xrightarrow{\mu} <p', p''>$, this is because p contains a subterm $\mu p'$ and p'' represents, intuitively, the term which is concurrent to $\mu p'$ in p: what we called earlier the *concurrent residual* of the transition, and will rename here, since we are reasoning about terms, the *coterm* of $\mu p'$ in p.

Now the operational behaviour of a term p is exactly determined by the set of its (initial) subterms $\mu p'$ together with their coterms p''. Unfortunately such coterms are not, in general, subterms of p. For example, in:

$$p = ((\mu p' \mid s_1) + r) \mid s_2$$

the term $\mu p'$ has coterm $p'' = s_1 \mid s_2$, which is not a subterm of p. Intuitively, this is because the action μ eliminates all subterms in alternative with $\mu p'$ (the subterm r in the above example).

As a consequence, we will not be able to prove the equality of two terms by first comparing their subterms $\mu p'$ and the respective coterms, as would be the natural way. Now the reason we introduce our new operator $\between$ is precisely to be able to express a term p in its *explicit form* $\sum_{i\in I} a_i p_i \between p'_i$, where for each $i \in I$ p'_i is the coterm of $a_i p_i$ (and a subterm of p).

The operational meaning of $\between$ is specified by the rule:

R5. $\quad p \xrightarrow{\mu} \langle p', p''\rangle \quad$ implies $\quad p \between q \xrightarrow{\mu} \langle p', p'' | q\rangle$

As this rule indicates, the operator $\between$ has some similarity with $|$. In fact it is easy to see that $p \between q$ is absorbed into $p|q$, that is:

$$p \between q \quad \subseteq_d \quad p|q$$

In absence of communication, that is when $|$ is defined by rule **R3a.** only, the operator $\between$ is all we need to derive a complete axiomatisation for $\sim_d$. In this case $p|q$ is absorbed into $(p \between q + q \between p)$. Indeed we have the following *expansion law* for $|$:

$$p|q \;=\; p \between q \;+\; q \between p \qquad \text{(PE)}$$

Thus $\between$ may be viewed as a sort of *asymmetric parallel operator*: in the term $p \between q$ the components p and q are concurrent but somehow p has an initial dominance over q. In fact, the introduction of $\between$ may seem to bring us back to an interleaving semantics. Fortunately this is not the case, since the equivalence $\sim_d$ – extended to the language with $\between$ – does *not* satisfy the law:

$$\mu p \between q \;=\; \mu(p|q) \qquad (\between\text{IN})$$

We have, for example: $\quad a \between b \not\sim_d ab \quad$ and also:

$$a|b \;\sim_d\; a \between b + b \between a \;\not\sim_d\; ab + ba$$

An important difference between $\between$ and $|$ is that $\between$ satisfies a distributive law:

$$(p+q) \between r = (p \between r) + (q \between r) \qquad \text{(LP1)}$$

whereas it is well-known that this is not the case for $|$ (at least in the theory of bisimulations).

The law LP1 will help us a great deal in syntactic manipulations: in particular it will be crucial to reduce a term p to its explicit form $\sum a_i p_i \between p'_i$.

The operator $\between$ is not associative. Instead, it satisfies the property:

$$(p \between q) \between r = p \between (q \mid r) \tag{LP2}$$

The three axioms PE, LP1, LP2 are very convenient. They may be used to derive the laws of $\mid$:

$$p \mid q = q \mid p \tag{P1}$$

$$p \mid (q \mid r) = (p \mid q) \mid r \tag{P2}$$

as well as absorption laws of the kind we saw above (in our examples at the beginning of section). The other main property of $\mid$:

$$p \mid \mathrm{NIL} = p \tag{P3}$$

follows from (PE) above and the additional two axioms:

$$p \between \mathrm{NIL} = p \tag{LP3}$$

$$\mathrm{NIL} \between p = \mathrm{NIL} \tag{LP4}$$

Let now $\mathcal{D}$ denote the laws A1–A4, PE, LP1–LP4, as listed in Figure 4.1.

If Σ_1 denotes the enlarged syntax:

$$\Sigma_1 = A_\tau \cup \{\mathrm{NIL}, +, \mid, \between\}$$

and we extend $\sim_d$ to the new calculus, we have the following result:

Theorem 4.17 *In absence of communication, the equivalence $\sim_d$ is the Σ_1-congruence generated by the axioms $\mathcal{D}$.*

We shall not give the proof of this theorem here, since it is essentially a simpler version of the forthcoming theorem 4.20, which covers the case of communication too.

Let us now turn to the general case, where $\mid$ is defined by both rules **R3a** and **R3b**. Note that the operator $\between$ does not have a communication rule like **R3b**.

Axioms $\mathcal{E} = \mathcal{D}\backslash PE \cup \Delta$

Axioms $\mathcal{D}$

$(A1)$ $\quad x + (y + z) = (x + y) + z$

$(A2)$ $\quad x + y = y + x$

$(A3)$ $\quad x + \mathrm{NIL} = x$

$(A4)$ $\quad x + x = x$

(PE) $\quad x \mid y = x \between y + y \between x$

$(LP1)$ $\quad (x + y) \between z = x \between z + y \between z$

$(LP2)$ $\quad (x \between y) \between z = x \between (y \mid z)$

$(LP3)$ $\quad x \between \mathrm{NIL} = x$

$(LP4)$ $\quad \mathrm{NIL} \between x = \mathrm{NIL}$

Axioms Δ

(CPE) $\quad x \mid y = x \between y + y \between x + x \mid_c y$

$(CP1)$ $\quad (x + y) \mid_c z = (x \mid_c z) + (y \mid_c z)$

$(CP2)$ $\quad x \mid_c y = y \mid_c x$

$(CP3)$ $\quad x \mid_c \mathrm{NIL} = \mathrm{NIL}$

$(CP4)$ $\quad (\mu x \between x') \mid_c (\nu y \mid_c y') = \begin{cases} \tau\,(x \mid y) \between (x' \mid y'), & \text{if } \mu = \bar{\nu} \\ \mathrm{NIL}, & \text{otherwise} \end{cases}$

Figure 4.1: Axiomatisation of $\sim_d$.

which *enforces* communication between its arguments. The rule for $|_c$ is the following:

R5. $p \xrightarrow{a} \langle p', p'' \rangle$, $q \xrightarrow{\bar{a}} \langle q', q'' \rangle$ imply $p \,|_c\, q \xrightarrow{\tau} \langle p' \,|\, q',\ p'' \,|\, q'' \rangle$

In fact the operator $|_c$ is not strictly needed as long as we do not want to abstract from communication actions, what we shall do in the next chapter (where this point is further explained, see page 141). At this stage, we could well do without $|_c$, by adding a communication rule for the operator $\between$. However, to keep our treatment of communication as uniform as possible, we prefer to introduce $|_c$ from the start.

Having defined $|_c$, we now replace the law PE by:

$$p \,|\, q \;=\; (p \between q) \;+\; (q \between p) \;+\; (p \,|_c\, q) \qquad \text{(CPE)}$$

The operator $|_c$ may be seen as a (restricted) *synchronous product*, similar to the product of SCCS. It obeys the following laws (which are valid also for SCCS product):

$$(p + q) \,|_c\, r \;=\; (p \,|_c\, r) \;+\; (q \,|_c\, r) \qquad \text{(CP1)}$$

$$p \,|_c\, q \;=\; q \,|_c\, p \qquad \text{(CP2)}$$

$$p \,|_c\, \mathrm{NIL} \;=\; \mathrm{NIL} \qquad \text{(CP3)}$$

On the other hand, $|_c$ only allows synchronisation of complementary actions:

$$(\mu p \between p') \,|_c\, (\nu q \between q') \;=\; \begin{cases} \tau\,(p \,|\, q) \;\between\; (p' \,|\, q'), & \text{if } \mu = \bar{\nu} \\ \mathrm{NIL}, & \text{otherwise} \end{cases} \qquad \text{(CP4)}$$

We will denote by Δ the set of laws CPE, CP1 – CP4, and by $\mathcal{E}$ the whole set of axioms $\mathcal{E} = \mathcal{D} \setminus \mathrm{PE} \cup \Delta$, as shown in Figure 4.1.

Let Σ_2 be our final syntax:

$$\Sigma_2 \;=\; A_\tau \cup \{\mathrm{NIL}, +, |, \between, |_c\}$$

and $\sim_d$ denote now the extended d-bisimulation equivalence.

We are then ready to prove our characterisation theorem. The clue of this theorem is as usual the reduction of terms to normal forms. Here we adopt as normal forms the *explicit forms* that we mentioned earlier on. Formally:

Definition 4.18 *A normal form (nf) is a term - defined modulo the axioms A1, A2, A3 - of the form:*

$$\hat{p} = \sum_{i \in I} \mu_i p_i \between p'_i$$

where for each $i \in I$ p_i *and* p'_i *are again nf's and by convention* $\hat{p} = \text{NIL}$ *if* $I = \emptyset$.

Note that all transitions of a normal form $\hat{p}$ are of the kind $\hat{p} \xrightarrow{\mu_i} <p_i, p'_i>$, for some $i \in I$. We have the following:

Lemma 4.19 *Normalisation lemma:*

For any $p \in T_{\Sigma_2}$ *there exists a nf* $\hat{p} = \sum_{i \in I} \mu_i p_i \between p'_i$ *such that* $p =_\mathcal{E} \hat{p}$.

Proof: The proof of this lemma, by induction on the size of p, makes use of all axioms in $\mathcal{E}$ except for the idempotence law A4. Axioms A1, A2, A3 are used implicitly throughout.

i) NIL is a normal form.

ii) $p = \mu q$. We let here $\hat{p} = \mu \hat{q} \between \text{NIL}$. We have then $p =_\mathcal{E} \hat{p}$ by induction and LP3.

iii) $p = q + r$. We let here $\hat{p} = \hat{q} + \hat{r}$. Then $p =_\mathcal{E} \hat{p}$ by simple induction.

iv) $p = q \between r$. If $q = \sum_{i \in I} \nu_i q_i \between q'_i$, we define: $\hat{p} = \sum_{i \in I} \nu_i q_i \between \widehat{(q'_i \between r)}$. Now, if $\hat{q} = \text{NIL}$ then $I = \emptyset$ and also $\hat{p} = \text{NIL}$. In this case we get, using induction and axiom LP4 :

$$p =_\mathcal{E} \hat{q} \between r = \text{NIL} \between r =_\mathcal{E} \text{NIL} =_\mathcal{E} \hat{p}$$

Otherwise we have:

$$p = \sum_{i \in I} (\nu_i q_i \between q'_i) \between r =_\mathcal{E} \sum_{i \in I} \left[(\nu_i q_i \between q'_i) \between r \right] =_\mathcal{E} \sum_{i \in I} \nu_i q_i \between \widehat{(q'_i \between r)} =_\mathcal{E} \hat{p}$$

using induction and the laws LP1, LP2.

v) $p = q\,|_c\, r$. If $\hat{r} = \mathrm{NIL}$ we let $\hat{p} = \mathrm{NIL}$. Then, by induction and axiom CP3 we have: $p =_\mathcal{E} q\,|_c\,\hat{r} = q\,|_c\,\mathrm{NIL} =_\mathcal{E} \mathrm{NIL}$. Same thing, using CP2 first, when $\hat{q} = \mathrm{NIL}$.

Otherwise, if $\hat{q} = \sum_{i\in I} \mu_i q_i \parallel q'_i$ and $\hat{r} = \sum_{j\in J} \nu_j r_j \parallel r'_j$, we define $\hat{p} = \sum_{\mu_i=\nu_j} \tau\,(\widehat{q_i\,|\,r_j}) \parallel (\widehat{q'_i\,|\,r'_j})$. Then, using induction and axioms CP1, CP2, CP4, we obtain:

$$p =_\mathcal{E} (\sum_{i\in I} \mu_i q_i \parallel q'_i)\,|_c\,(\sum_{j\in J} \nu_j r_j \parallel r'_j) =_\mathcal{E} \sum_{i\in I}\sum_{j\in J} (\mu_i q_i \parallel q'_i)\,|_c\,(\nu_j r_j \parallel r'_j)$$
$$=_\mathcal{E} \sum_{\mu_i=\nu_j} \tau\,(\widehat{q_i\,|\,r_j}) \parallel (\widehat{q'_i\,|\,r'_j}) = \hat{p}$$

vi) $p = q\,|\,r$. Let $p_1 = q \parallel r$, $p_2 = r \parallel q$ and $p_3 = q\,|_c\,r$. We set $\hat{p} = \hat{p}_1 + \hat{p}_2 + \hat{p}_3$, where $\hat{p}_1$ and $\hat{p}_2$ are defined as in case **iv)**, and $\hat{p}_3$ as in case **v)**. Then, using induction and the expansion law CPE, we obtain:

$$p =_\mathcal{E} p_1 + p_2 + p_3 =_\mathcal{E} \hat{p}_1 + \hat{p}_2 + \hat{p}_3 =_\mathcal{E} \hat{p} \qquad \square$$

We give now our main result.

Theorem 4.20 *The equivalence* $\sim_d$ *is the* Σ_2*-congruence* $=_\mathcal{E}$ *generated by the axioms* $\mathcal{E}$.

Proof. The proof is naturally divided in two parts, the *soundness* of the axioms, which is easy – but somewhat tedious – to show, and their *completeness*, which is straightforward once we have proved the normalisation lemma.

1) Soundness: $p =_\mathcal{E} q \implies p \sim_d q$. Check that if $p = q$ is an instance of an axiom of $\mathcal{E}$, then $p \sim_d q$.

2) Completeness: $p \sim_d q \implies p =_\mathcal{E} q$. We prove this statement by induction on the sizes of p, q. The only nontrivial axiom used explicitly in the proof is the absorption law A4 .

Suppose that $p \sim_d q$. We may assume p, q to be normal forms:

$$p = \sum_{i\in I} \mu_i p_i \parallel p'_i, \qquad q = \sum_{j\in J} \nu_j q_j \parallel q'_j$$

We show now that $q + p =_\mathcal{E} q$. Then, by a symmetric argument, we have also $p + q =_\mathcal{E} p$, and by combining these two equalities we obtain the required result: $p =_\mathcal{E} p + q =_\mathcal{E} q$.

To prove $q + p =_\mathcal{E} q$, it is sufficient to show that $\forall i \in I$:

$$q + \mu_i p_i \between p_i' =_\mathcal{E} q$$

Note that for any $\mu_i p_i \between p_i'$ we have $p \xrightarrow{\mu_i} \langle p_i, p_i' \rangle$. Since $p \sim_d q$, there must be $j \in J$ s.t. $q \xrightarrow{\nu_j} \langle q_j, q_j' \rangle$, with $\mu_i = \nu_j$ and $p_i \sim_d q_j$, $p_i' \sim_d q_j'$. By induction $p_i =_\mathcal{E} q_j$, $p_i' =_\mathcal{E} q_j'$, whence by substitution and A4 we obtain:

$$q + \mu_i p_i \between p_i' =_\mathcal{E} q + \nu_j q_j \between q_j' =_\mathcal{E} q$$

which is what we wanted to show. □

We should point out here that the recourse to an asymmetric parallel operator is not unusual. The operator $\between$ was used by M. Hennessy in [Hen 80] for a language similar to ours, but with a (formally) different operational semantics. Also, the operator $\between$ may be used – and has been used extensively by Bergstra and Klop – in [BK 84] and related works – to give a *finite* axiomatisation for finitary CCS. Note in fact that the axiomatisation proposed by Milner and Hennessy makes use of the expansion theorem (IN), which is an axiom schema. Instead, one can use an operator $\between$, called *left-merge* by Bergstra and Klop, which obeys the same laws as ours and moreover:

$$\mu p \between q = \mu (p \mid q) \qquad (\between \text{IN})$$

To treat communication one takes in addition the operator $|_c$, with exactly the same properties CPE, CP1 – CP4 that we have here. The operational rules are those of CCS plus two new rules for $\between$ and $|_c$, the following:

R4'. $p \xrightarrow{\mu} p'$ implies $p \between q \xrightarrow{\mu} p' \mid q$

R5'. $p \xrightarrow{a} p'$, $q \xrightarrow{\bar{a}} q'$ imply $p |_c q \xrightarrow{\tau} p' \mid q'$

Now it turns out that the laws $\mathcal{D} \cup \{\between \text{IN}\}$ provide a *finite* complete axiomatisation for the strong bisimulation $\sim$ on the language T_{Σ_1} without communication, while the laws $\mathcal{E} \cup \{\between \text{IN}\}$ characterise $\sim$ on the language T_{Σ_2} (with communication). This point is rather interesting: it shows that the whole difference between Milner's semantics and ours comes down to a single axiom, the law $(\between \text{IN})$.

One last remark concerning $\sim_d$, before we end the chapter. We said earlier that we could have avoided introducing the operator $|_c$ at this stage, by allowing communication across $\between$. In fact, if we do not insist on the *finiteness* of the axiomatisation, there is another alternative to the use of $|_c$, which does not require a communication rule for $\between$. This consists in replacing the expansion law (CPE) and the remaining axioms of $|_c$ by the following *expansion theorem*:

(EXP) If $x = \sum_{i\in I} \mu_i x_i \between x'_i$ and $y = \sum_{j\in J} \nu_j y_j \between y'_j$, then

$$x|y = \sum_{i\in I} \mu_i x_i \between (x'_i|y) + \sum_{j\in J} \nu_j y_j \between (x|y'_j) + \sum_{\mu_i=\nu_j} \tau(x_i|y_j) \between (x'_i|y'_j)$$

Then it may be proved (although we shall not do it here) that $\mathcal{D}\setminus\{\text{PE}\}\cup\{\text{EXP}\}$ is a complete axiomatisation for $\sim_d$ on the language T_{Σ_1}.

To sum up, we have studied in this chapter what we could call the *strong d-bisimulation equivalence* on the language T_{Σ_2}. In particular, we saw that the introduction of communication leads to new identifications between processes. One has typically the absorption of example 6) above:

$$\tau(b|c) \quad \subsetneq_d \quad (ab \mid \bar{a}c)$$

Note on the other hand that:

Example 9) $\qquad \tau\tau(c|d) \quad \not\subseteq_d \quad (abc \mid \bar{a}\bar{b}d)$

Similarly, we have:

Example 10) $\qquad \tau\tau(p|q|r|s) \quad \not\subseteq_d \quad (ap|\bar{a}q) \mid (br|\bar{b}s)$

If we adopt now Milner's interpretation of τ actions as *unobservable*, as we did previously in chapter 2, then it is clear that the two absorptions in 9) and 10) should become valid as well.

Our next concern will be precisely to examine the *weak* version $\approx_d$ of our d-bisimulation equivalence. We shall see that our results for $\sim_d$ (the connection with CCS semantics, the existence of a complete axiomatisation) carry over to the *weak d-bisimulation* $\approx_d$. The study of $\approx_d$ will be the subject of the coming chapter.

Chapter 5

Weak distributed bisimulations

In this chapter we develop the theory of distributed bisimulations in the presence of *unobservable* actions. Following Milner, we will assume τ actions – that is actions arising from communication – to be *internal* to a process and thus unobservable. In fact, the treatment of unobservable actions is a problem in its own right, which may be studied (as we did in Chapter 2) independently of *how* such actions occur.

We shall therefore consider here, besides a calculus with unobservable communication actions, also a simpler one with unobservable actions but no communication. With the now familiar technique, we abstract from internal actions by allowing them to be absorbed into observable transitions. More precisely, any sequence of transitions including only one observable transition is now allowed to occur in one step. A sequence of this kind is rendered as a *weak transition* $\stackrel{a}{\Longrightarrow}$, labelled with the (unique) observable action in the sequence. For technical reasons, it is necessary to introduce also weak transitions of the form $\stackrel{\tau}{\Longrightarrow}$, representing a finite sequence of τ transitions.

We may now use *weak distributed transitions* to describe the operational behaviour of processes. A weak distributed transition (d–transition) has exactly the same form as a strong d–transition: it is labelled by a single *action* and generates a pair of residuals (local and concurrent). The main novelty is that a weak d–transition may affect different parallel components; while an observable action is taking place in some component, internal actions may occur independently in other components.

As is to be expected, the *weak d-bisimulation equivalence* $\approx_d$ based on the new transitions is not a congruence (the critical operator being once again the sum). We therefore concentrate on the *substitutive* version of $\approx_d$, denoted $\approx_d^c$.

The structure of this chapter is closely modelled on that of chapter 4. Although the study of $\approx_d^c$ is somewhat more laborious than that of $\sim_d$, we obtain for $\approx_d$ and $\approx_d^c$ the same kind of results as for $\sim_d$. We show for example that $\approx_d$ is an extension of Milner's weak bisimulation equivalence $\approx$. In section 5.3 we prove that, when no communication is involved, our alternative semantic definition of chapter 4 carries over to weak d-transitions. We have not been able, as yet, to generalise this result to the complete calculus (with communication).

Section 5.4 is devoted to the study of the behavioural congruence $\approx_d^c$. Finally in section 5.5 we give a complete axiomatisation for $\approx_d^c$, both in the calculus without communication and in the complete calculus.

5.1 Weak distributed transition systems

Let us now introduce our new calculi. We assume the usual syntax for processes:

$$\Sigma' = A_\tau \cup \{\,\mathrm{NIL}, +, \mid\,\}$$

Note that Σ' does not include the auxiliary operators $\between$ and $\mid_c$. These will not be inserted until the last section, where they are needed to derive our axiomatisation for $\approx_d^c$.

Let again $\mathcal{P}' = T_{\Sigma'}$ denote the class of processes, considered as computational objects. We seek an operational description for processes of $\mathcal{P}'$ that abstracts to some extent from τ-actions.

To obtain this, we replace the transitions $\xrightarrow{a}$ by the corresponding *weak* transitions $\stackrel{a}{\Longrightarrow}$. Informally, the meaning of a weak transition $p \stackrel{a}{\Longrightarrow} <p', p''>$ is that p may evolve internally for some time, then perform an action a and thereafter still possibly move internally to reach the state $<p', p''>$. Thus a weak transition $\stackrel{a}{\Longrightarrow}$ involves a transition $\xrightarrow{a}$ as well as a finite number, possibly equal to zero, of transitions $\xrightarrow{\tau}$ before and after it.

In fact, because of internal actions, a weak transition $\stackrel{a}{\Longrightarrow}$ may have an effect on different components. For example we expect:

Example 1) $\quad ap \mid (\tau q + r) \stackrel{a}{\Longrightarrow} <p, \text{NIL} \mid q>$

since the internal action τ of the component $(\tau q + r)$ may occur while the action a is taking place in the component ap.

As was the case already for CCS, weak *unobservable* transitions of the form $\stackrel{\tau}{\Longrightarrow}$ are needed to obtain an equivalence which is substitutive w.r.t. the operators $\mu:$ and $\mid$. The following example shows that an equivalence based on transitions $\stackrel{a}{\Longrightarrow}$ only would not be preserved by prefixing.

Example 2) The terms $a + b$ and $a + \tau b$ have the same weak observable transitions. However this is no longer true if we prefix them by an action c, since the move $c(a + \tau b) \stackrel{c}{\Longrightarrow} b$ has no counterpart in $c(a + b)$.

For unobservable weak transitions we choose here the simple form $p \stackrel{\tau}{\Longrightarrow} q$ rather than $p \stackrel{\tau}{\Longrightarrow} <p', p''>$. This means that an *unobservable* action τ is regarded as *global*, and may be localised only indirectly, if it affects the observable behaviour of the component where it occurs. For example the locality of the action τ will not be observable in the process $p \mid \tau q$, whereas it will be in the process $p \mid (\tau q + r)$, since here the τ action may prevent a local observer of $(\tau q + r)$ from obtaining an action of r.

We now proceed to formally define the relations $\stackrel{\mu}{\Longrightarrow}$ on $\mathcal{P}'$, via a set of rules $\mathcal{S}$ similar to those for $\stackrel{\mu}{\longrightarrow}$. We start by specifying the relation $\stackrel{\tau}{\longrightarrow}$. The relation $\stackrel{\tau}{\Longrightarrow}$ is then defined as an abbreviation for $\stackrel{\tau^n}{\longrightarrow}$, $n \geq 0$.

The rules for $\stackrel{\tau}{\longrightarrow}$ are given at top of figure 5.1. Note that they coincide with CCS rules if we except the communication rule $\tau 4$, which makes use of strong d–transitions. However, after our discussion of Chapter 4, it should be clear that the rule $\tau 4$ is in fact equivalent to the communication rule of CCS.

Consider now the rules for $\stackrel{a}{\Longrightarrow}$, shown at bottom of figure 5.1. There is nothing new as concerns rules S1 – S3. Rules S4 – S5 allow us to collapse unobservable transitions either before or after an observable transition.

The communication rule S7 deserves some comment. Note that a communication action following an action a preserves the locality of the a action, although the local residual is partly recombined with the concurrent one. Let us consider a simple instance of this rule.

Rules S

RULES for $\xrightarrow{\tau}$

τ 1. τ: $p \xrightarrow{\tau} p$

τ 2. $p \xrightarrow{\tau} p'$ implies $p + q \xrightarrow{\tau} p'$

$q + p \xrightarrow{\tau} p'$

τ 3. $p \xrightarrow{\tau} p'$ implies $p \mid q \xrightarrow{\tau} p' \mid q$

$q \mid p \xrightarrow{\tau} q \mid p'$

τ 4. $p \xrightarrow{a} <p', p''>$, $q \xrightarrow{\bar{a}} <q', q''>$ imply $p \mid q \xrightarrow{\tau} (p' \mid q') \mid (p'' \mid q'')$

RULES for $\stackrel{a}{\Longrightarrow}$

S1. a: $p \stackrel{a}{\Longrightarrow} <p, \mathrm{NIL}>$

S2. $p \stackrel{a}{\Longrightarrow} <p', p''>$ implies $p + q \stackrel{a}{\Longrightarrow} <p', p''>$

$q + p \stackrel{a}{\Longrightarrow} <p', p''>$

S3. $p \stackrel{a}{\Longrightarrow} <p', p''>$ implies $p \mid q \stackrel{a}{\Longrightarrow} <p', p'' \mid q>$

$q \mid p \stackrel{a}{\Longrightarrow} <p', q \mid p''>$

S4. $p \xrightarrow{\tau} q$, $q \stackrel{a}{\Longrightarrow} <q', q''>$ imply $p \stackrel{a}{\Longrightarrow} <q', q''>$

S5. $p \stackrel{a}{\Longrightarrow} <p', p''>$, $p' \xrightarrow{\tau} q$ imply $p \stackrel{a}{\Longrightarrow} <q, p''>$

S6. $p \stackrel{a}{\Longrightarrow} <p', p''>$, $p'' \xrightarrow{\tau} q$ imply $p \stackrel{a}{\Longrightarrow} <p', q>$

S7. $p \stackrel{a}{\Longrightarrow} <p', p''>$, $p' \xrightarrow{c} <r', r''>$, $p'' \xrightarrow{\bar{c}} <s', s''>$

imply $p \stackrel{a}{\Longrightarrow} <r' \mid s', r'' \mid s''>$

Figure 5.1: Weak distributed transitions

Example 3) $(acb|d) \mid (\bar{c}e|f) \stackrel{a}{\Longrightarrow} <b|e, d|f>.$

Here the synchronisation of the actions c and $\bar{c}$ introduces a causal dependency between the action a of the first component and the subprocess e of the second component. This explains why the process e, previously placed in the concurrent residual of the transition $\stackrel{a}{\longrightarrow}$, is now shifted to the local residual of the sequence $\stackrel{a}{\longrightarrow}\stackrel{\tau}{\longrightarrow}$. Note on the other hand that the components d and f, independent both from the action a and from the subsequent communication, are left in the concurrent residual.

As a general consequence of rule S7, the local residual of a transition $\stackrel{a}{\Longrightarrow}$ will no longer be, as was the case for $\stackrel{a}{\longrightarrow}$, a subterm of the component which performs the action a.

5.2 Weak distributed bisimulation equivalence

Having introduced the weak transitions $\stackrel{a}{\Longrightarrow}$ and $\stackrel{\tau}{\Longrightarrow}$, we may use them to define a new d–bisimulation equivalence $\approx_d$ on processes of $\mathcal{P}'$. We start by giving the notion of weak d–bisimulation relation.

Definition 5.1 *A weak d–bisimulation is a relation $R \subseteq (\mathcal{P}' \times \mathcal{P}')$ satisfying, for any $(p,q) \in R$ and $a \in A$, the following property:*

i) $p \stackrel{\tau}{\Longrightarrow} p' \implies q \stackrel{\tau}{\Longrightarrow} q'$, *with* $p' \, R \, q'$

ii) $q \stackrel{\tau}{\Longrightarrow} q' \implies p \stackrel{\tau}{\Longrightarrow} p'$, *with* $p' \, R \, q'$

iii) $p \stackrel{a}{\Longrightarrow} <p', p''> \implies q \stackrel{a}{\Longrightarrow} <q', q''>$, *with* $p' \, R \, q'$ *and* $p'' \, R \, q''$

iv) $q \stackrel{a}{\Longrightarrow} <q', q''> \implies p \stackrel{a}{\Longrightarrow} <p', p''>$, *with* $p' \, R \, q'$ *and* $p'' \, R \, q''$

If we let $WD(R)$ be the set of all pairs (p,q) satisfying clauses $i) - iv)$, we have the more compact definition: a relation $R \subseteq (\mathcal{P}' \times \mathcal{P}')$ is a weak d–bisimulation iff $R \subseteq WD(R)$.

A weak d–bisimulation will be sometimes called wd–bisimulation. Our new behavioural equivalence over $\mathcal{P}'$, which we call *weak distributed bisimulation equivalence* and denote by $\approx_d$, is now defined to be the union of all weak d-bisimulations:

Definition 5.2 $\forall p,q \in \mathcal{P}' : \; p \approx_d q \;$ *iff* $\exists$ *wd-bisimulation* R *s.t.* $p \, R \, q$.

We have for $\approx_d$ the usual property of (maximal) bisimulation equivalences.

Property 5.3

i) $\approx_d$ *is an equivalence relation.*

ii) $\approx_d$ *is the largest solution of* $R \subseteq WD(R)$ *(and* $R = WD(R)$*).*

□

The equivalence $\approx_d$ is preserved by most operators of Σ', namely:

Property 5.4 $\approx_d$ *is preserved by the operators* $\mu:$ *and* $|$.

Proof: Trivial for the operator $\mu:$. To show that $\approx_d$ is preserved by $|$ we seek a wd-bisimulation R such that $(p|r, q|r) \in R$ whenever $p \approx_d q$. Now it is easy to check that the relation:

$$R \;=\; \{(p|r, q|r) \mid p \approx_d q\} \cup \approx_d$$

is such a wd-bisimulation. □

We give next some examples of identifications modulo $\approx_d$.

Example 4) $\qquad \tau\, a \mid b \;\approx_d\; \tau\,(a \mid b)$

Note that this identification would not be valid if we could observe the locality of the τ–transition in the first process.

Example 5) $\qquad \tau\,(c \mid d) \;\underset{\approx_d}{\subseteq}\; (a\,b\,c \mid \bar{a}\,\bar{b}\,d)$

where $\underset{\approx_d}{\subseteq}$ is the absorption relation corresponding to $\approx_d$, defined by: $p \underset{\approx_d}{\subseteq} q \iff (p+q) \approx_d q$. Similarly, we have:

Example 6) $\tau(p|q|r|s) \quad \underset{\approx_d}{\subset} \quad (ap|\bar{a}q) \mid (br|\bar{b}s)$

One last example of absorption, illustrating the effect of rule **S7**:

Example 7) $a(b|d) \quad \underset{\approx_d}{\subset} \quad (acb \mid \bar{c}d)$

We conclude this section with a simple result: the weak d-bisimulation equivalence $\approx_d$ is an extension – although not a conservative one, as shown by the examples above – of the strong d-bisimulation $\sim_d$ of chapter 4. Moreover the two relations coincide whenever there are no internal moves involved.

Theorem 5.5

i) $p \sim_d q$ *implies* $p \approx_d q$.

ii) *If* p, q *contain no occurrences of* a, *then* $p \approx_d q$ *implies* $p \sim_d q$. □

5.3 Alternative formulation of the semantics

We proceed now, as we did in chapter 4, to extend our semantics to pairs of processes $<p, q>$. The rules for pairs are given below. Note that τ–transitions transform a pair in another pair: nevertheless they are *global* moves in the sense that they do not create a new local residual, but merely preserve the one created by a preceding observable transition.

τ 5. $p \xrightarrow{\tau} p'$ implies $<p, q> \xrightarrow{\tau} <p', q>$

$<q, p> \xrightarrow{\tau} <q, p'>$

τ 6. $p \xrightarrow{a} <p', p''>$, $q \xrightarrow{\bar{a}} <q', q''>$ imply $<p, q> \xrightarrow{\tau} <p'|q', p''|q''>$

S8. $p \overset{a}{\Longrightarrow} <p', p''>$ implies $<p, q> \overset{a}{\Longrightarrow} <p', p''|q>$

$<q, p> \overset{a}{\Longrightarrow} <p', q|p''>$

Having defined the transitions $\xrightarrow{\tau}$ for pairs of processes allows us to unify the three rules S5 – S7 in figure 5.1 into just one rule:

S9. $p \stackrel{a}{\Longrightarrow} \langle p', p'' \rangle \stackrel{\tau}{\longrightarrow} \langle q', q'' \rangle$ imply $p \stackrel{a}{\Longrightarrow} \langle q', q'' \rangle$

Note again the similarity between the transitions of the pair $\langle p, q \rangle$ and those of the process $p \mid q$. We shall see next that $\langle p, q \rangle$ and $p \mid q$ have indeed the same behaviour, modulo the (extended) weak d–bisimulation equivalence $\approx_d$.

The extension of $\approx_d$ to pairs of processes is straightforward. If $\mathcal{P}'' =_{def} \mathcal{P}' \cup \{(p,q) \mid p, q \in \mathcal{P}'\}$, and P, Q, etc. range over $\mathcal{P}''$, we define the extended equivalence – wich we still denote by $\approx_d$ – as follows.

Definition 5.6 *For any $P, Q \in \mathcal{P}''$, let $P \approx_d Q$ if and only if for any $a \in A$ the following hold:*

i) $P \stackrel{\tau}{\Longrightarrow} P' \implies Q \stackrel{\tau}{\Longrightarrow} Q'$, *with* $P' \approx_d Q'$

ii) $Q \stackrel{\tau}{\Longrightarrow} Q' \implies P \stackrel{\tau}{\Longrightarrow} P'$, *with* $P' \approx_d Q'$

iii) $P \stackrel{a}{\Longrightarrow} \langle p', p'' \rangle \implies Q \stackrel{a}{\Longrightarrow} \langle q', q'' \rangle$, *with* $p' \approx_d q'$ *and* $p'' \approx_d q''$

iv) $Q \stackrel{a}{\Longrightarrow} \langle q', q'' \rangle \implies P \stackrel{a}{\Longrightarrow} \langle p', p'' \rangle$, *with* $p' \approx_d q'$ *and* $p'' \approx_d q''$

Again, one may easily show (the proof is the same as for $\sim_d$) that for any processes p, q in $\mathcal{P}'$ the pair $\langle p, q \rangle$ is bisimilar to the process $p \mid q$.

Fact 5.7 $\forall\, p, q \in \mathcal{P}' : \quad \langle p, q \rangle \approx_d p \mid q$. □

Once more, adding a condition $\langle p', p'' \rangle \; R \; \langle q', q'' \rangle$ at the end of clauses *iii*) and *iv*) in the above definition would not change the resulting wd–bisimulation equivalence.

In fact, by property 5.4, $\approx_d$ is preserved by the operator $\mid$. This ensures us that when $p' \approx_d q'$ and $p'' \approx_d q''$ then $\langle p', p'' \rangle \approx_d p' \mid p'' \approx_d q' \mid q'' \approx_d \langle q', q'' \rangle$.

We may conclude that the rules for pairs $\tau 5$, $\tau 6$, S8 are superfluous for the definition of $\approx_d$ on processes. However, as we shall see in the remainder of this section, these rules may be used to give an alternative formulation of our semantics along the lines of chapter 4.

We have just seen that the compound residual $<p', p''>$ of a transition behaves like the parallel process $p' \mid p''$. We may then envisage, as we did for our "strong" semantics, to describe the behaviour of a process by means of transitions of a different kind, yielding a single *global residual* and carrying a new information – the local residual – above the arrow. Such transitions will thus have the form $p \stackrel{a,p'}{\Longrightarrow} q$, and the relative bisimulation will recur separately on the local and global residuals.

However, using $p' \mid p''$ in place of $<p', p''>$ gives us a slight problem when trying to reformulate rules S5, S6 and S7 . Consider rules S5, S6 first. Intuitively, whenever $p \stackrel{a,p'}{\Longrightarrow} q$ the global residual q includes the local residual p'. Now, if p does a further action τ, we must make sure that this action modifies consistently the local and global residuals. Thus we cannot for example replace rule S5 by:

$S5'.$ $\quad p \stackrel{a,p'}{\Longrightarrow} q$, $p' \stackrel{\tau}{\longrightarrow} p''$ imply $p \stackrel{a,p''}{\Longrightarrow} q$

since this would create a mismatch between our local and global description (the local residual would no more divide the global residual). A similar argument holds for rule S6.

As for rule S7, we want to be sure that after a transition $p \stackrel{a,p'}{\Longrightarrow} q$ the two synchronising actions c and $\bar{c}$ come respectively from the parts p' and $q \backslash p'$ of the residual q.

One way of tackling this difficulty is the one adopted in [CH 87] : we introduce contexts of the form $C[\ ]$, which are simply terms with a "hole" in them (a definition may be found at page 130). Then the transitions take the form $p \stackrel{a}{\Longrightarrow} C[p']$, where p' is the local residual and $C[\ ]$ is the global environment for p', corresponding to $q \backslash p'$ above. The bisimulation recurs on p' and $C[p']$.

In this case Rule S5 may be reformulated as:

$S5''.$ $\quad p \stackrel{a}{\Longrightarrow} C[p']$, $p' \stackrel{\tau}{\longrightarrow} p''$ imply $p \stackrel{a}{\Longrightarrow} C[p'']$

But of course the simplest way to get round the problem is to keep the "notation" $<p', p''>$ for the global residual $p' \mid p''$, while disallowing the bisimulation from taking apart the components of a pair. More precisely, we shall keep our

present transitions $p \stackrel{a}{\Longrightarrow} <p', p''>$, and define our new wd–bisimulation $\cong_d$ to recur on the local residual p' *and* on the pair $<p', p''>$. The definition follows:

Definition 5.8 *For any relation* $R \subseteq (\mathcal{P}'' \times \mathcal{P}'')$ *we let* $p\ WD'(R)\ q$ *if and only if for any* $a \in A$*:*

i) $p \stackrel{\tau}{\Longrightarrow} p' \implies q \stackrel{\tau}{\Longrightarrow} q'$, *with* $p'\,R\,q'$

ii) $q \stackrel{\tau}{\Longrightarrow} q' \implies p \stackrel{\tau}{\Longrightarrow} p'$, *with* $p'\,R\,q'$

iii) $p \stackrel{a}{\Longrightarrow} <p', p''> \implies q \stackrel{a}{\Longrightarrow} <q', q''>$, *with* $p'R\,q'$ *and* $<p', p''>\ R\ <q', q''>$

iv) $q \stackrel{a}{\Longrightarrow} <q', q''> \implies p \stackrel{a}{\Longrightarrow} <p', p''>$, *with* $p'R\,q'$ *and* $<p', p''>\ R\ <q', q''>$

We then define $\cong_d$ to be the union of all such relations:

Definition 5.9 *Let* $p, q \in \mathcal{P}''$. Then :

$$p \cong_d q \quad \textit{iff} \quad \exists\ R \subseteq (\mathcal{P}'' \times \mathcal{P}'')\ \ s.t.\ \ p\,R\,q\ \textit{and}\ R \subseteq WD'(R).$$

We want now to establish the relation between $\cong_d$ and our previous bisimulation equivalence $\approx_d$. Remember that the corresponding strong equivalences $\simeq_d$ and $\sim_d$ were proven to coincide in chapter 4. Unfortunately we do not have, for now, a similar result for $\approx_d$ in our complete calculus (with communication). However we can prove that the two equivalences $\cong_d$ and $\approx_d$ coincide on the calculus without communication – i.e. the calculus restricted to rules $\tau 1$–$\tau 3$, S1–S6. The proof is based as usual on a simplification result.

To establish our simplication result we actually need to prove three statements:

Lemma 5.10 (*Simplification, in the restricted calculus*) *If* $p, q, r \in \mathcal{P}'$:

i) $r \stackrel{\tau}{\Longrightarrow} r'$ *and* $p|r \approx_d q|r'$ *imply* $q \stackrel{\tau}{\Longrightarrow} q' \approx_d p$.

ii) $r \stackrel{a}{\Longrightarrow} <r', r''>$ *and* $p|r \approx_d q|r''$ *imply* $q \stackrel{a}{\Longrightarrow} <q', q''>$, *with* $r' \approx_d q'$ *and* $p \approx_d q''$.

iii) Simplification: $p|r \approx_d q|r$ *implies* $p \approx_d q$.

Proof: The three statements are proved simultaneously, by induction on the sum of sizes of p, q, r.

i) Here $p|r \stackrel{\tau}{\Longrightarrow} p|r'$ and thus $q|r'$ must have a matching move $q|r' \stackrel{\tau}{\Longrightarrow} q'|r'' \approx_d p|r'$, for some q', r'' s.t. $q \stackrel{\tau}{\Longrightarrow} q'$ and $r' \stackrel{\tau}{\Longrightarrow} r''$.

Now, if $r'' = r'$ we may apply induction on part *iii)* to get $q' \approx_d p$.

Otherwise we apply induction on part *i)* to obtain $q' \stackrel{\tau}{\Longrightarrow} q'' \approx_d p$.

ii) Here $p|r \stackrel{a}{\Longrightarrow} <r', p|r''>$. Now $q|r''$ can match this move in two ways.

a) $q|r'' \stackrel{a}{\Longrightarrow} <q', q''|r'''>$ because $q \stackrel{a}{\Longrightarrow} <q', q''>$ and $r'' \stackrel{\tau}{\Longrightarrow} r'''$. In this case we have $r' \approx_d q'$ and $p|r'' \approx_d q''|r'''$.

To the latter equation we may now apply induction, part *i)*, to obtain $q'' \stackrel{\tau}{\Longrightarrow} q''' \approx_d p$. Thus $q \stackrel{a}{\Longrightarrow} <q', q'''>$ is the required move of q.

b) $q|r'' \stackrel{a}{\Longrightarrow} <s', q'|s''>$ because $r'' \stackrel{a}{\Longrightarrow} <s', s''>$ and $q \stackrel{\tau}{\Longrightarrow} q'$. Then $r' \approx_d s'$ and $p|r'' \approx_d q'|s''$. We may now apply induction on part *ii)* to get $q' \stackrel{a}{\Longrightarrow} <q'', q'''>$, with $r'' \approx_d q''$ and $p \approx_d q'''$.

iii) To prove $p \approx_d q$, we show that for any move of p there exists a corresponding move of q. The converse will then follow by symmetry.

a) $p \stackrel{\tau}{\Longrightarrow} p'$. Then $p|r \stackrel{\tau}{\Longrightarrow} p'|r$ and $q|r$ must have a matching move $q|r \stackrel{\tau}{\Longrightarrow} q'|r' \approx_d p'|r$, where $q \stackrel{\tau}{\Longrightarrow} q'$ and $r \stackrel{\tau}{\Longrightarrow} r'$.

Now, if $r' = r$ we apply induction on part *iii)* to get $q' \approx_d p$.

Otherwise we apply induction on part *i)* to obtain $q' \stackrel{\tau}{\Longrightarrow} q'' \approx_d p'$.

b) $p \stackrel{a}{\Longrightarrow} <p', p''>$. Then $p|r \stackrel{a}{\Longrightarrow} <p', p''|r>$. Again, $q|r$ has two ways to match this move.

1) $q|r \stackrel{a}{\Longrightarrow} <q', q''|r'>$ because $q \stackrel{a}{\Longrightarrow} <q', q''>$ and $r \stackrel{\tau}{\Longrightarrow} r'$. In this case we have $p' \approx_d q'$ and $p''|r \approx_d q''|r'$.

To the latter equation we apply now induction, part *i)*, to obtain $q'' \stackrel{\tau}{\Longrightarrow} q''' \approx_d p''$. Thus $q \stackrel{a}{\Longrightarrow} <q', q'''>$ is the required move of q.

2) $q|r \stackrel{a}{\Longrightarrow} <r', q'|r''>$ because $r \stackrel{a}{\Longrightarrow} <r', r''>$ and $q \stackrel{\tau}{\Longrightarrow} q'$. Then $p' \approx_d r'$ and $p''|r \approx_d q'|r''$. We now apply induction on part *ii)* to get $q' \stackrel{a}{\Longrightarrow} <q'', q'''>$, with $r' \approx_d q''$ and $p'' \approx_d q'''$. Since $p' \approx_d r'$, this is the required move of q. □

Theorem 5.11 (*Equivalence of the two semantics, in the restricted calculi*)

For any $p,q \in \mathcal{P}'$: $\quad p \approx_d q \iff p \cong_d q$.

Proof: Essentially the same as that of theorem 4.15 for $\sim_d$. The proof of $\approx_d \subset \cong_d$ uses the substitutivity of $\approx_d$ w.r.t. $|$, while the proof of $\cong_d \subset \approx_d$ requires the simplification lemma above.

Note that we do not need a conversion lemma here, since $\cong_d$ and $\approx_d$ are based on the same transitions $p \stackrel{a}{\Longrightarrow} <p',p''>$ and $p \stackrel{\tau}{\Longrightarrow} p'$.

□

It is now easy to establish the relation between our semantics and the classical semantics of CCS. We show that Milner's weak bisimulation $\approx$ is an extension of $\approx_d$. We use the following conversion lemma for $\stackrel{a}{\Longrightarrow}$ transitions (our $\stackrel{\tau}{\Longrightarrow}$ transitions being the same as in CCS), which we state without proof.

Lemma 5.12 (*Conversion*) *Let* $p \in \mathcal{P}'$. *Then:*

1. $p \stackrel{a}{\Longrightarrow} <p',p''> \implies \exists\, q \equiv p' \,|\, p''$ *s.t.* $p \stackrel{a}{\Longrightarrow} q$.
2. $p \stackrel{a}{\Longrightarrow} q \implies \exists\, p', p''$ *s.t.* $q \equiv p' \,|\, p''$ *and* $p \stackrel{a}{\Longrightarrow} <p',p''>$.

where $\equiv$ *is the congruence induced by the laws P1 - P3 at p. 97.*

We may then show that $\approx$ is an extension of $\approx_d$.

Theorem 5.13 *For any* $p,q \in \mathcal{P}'$:

1. $p \approx_d q$ *implies* $p \approx q$.
2. *If* p,q *do not contain occurrences of* $|$, *then* $p \approx q$ *implies* $p \approx_d q$.

Proof: Essentially the same as that of the analog theorem 4.11 for $\sim_d$.

In the next section we introduce a behavioural congruence $\approx_d^c$ based on $\approx_d$. The rest of the chapter is devoted to the study of properties of $\approx_d^c$.

5.4 Behavioural congruence

We saw in section 5.2 that the equivalence $\approx_d$ is preserved by most operators of Σ'. However $\approx_d$ is not preserved by the operator $+$, as shown by the usual example: $a \approx_d \tau a$ but $a + b \not\approx_d \tau a + b$. We shall therefore strengthen $\approx_d$ to a congruence $\approx_d^c$, as we did already for $\approx$ in Chapter 2.

We first define *contexts* $C[\,]$ to be generated by the following grammar:

$$C[\,] ::= [\,] \mid \mu{:}C[\,] \mid p + C[\,] \mid C[\,] + p \mid p \mid C[\,] \mid C[\,] \mid p$$

Then our behavioural equivalence $\approx_d^c$ is taken to be the closure of $\approx_d$ w.r.t. all contexts:

Definition 5.14 *Let* $p, q \in \mathcal{P}'$. *Then* $p \approx_d^c q$ *iff* $\forall\, C[\,] : C[p] \approx_d C[q]$.

The relation $\approx_d^c$ so defined is the *largest* congruence included in $\approx_d$ (this is a standard result). Note that sum–contexts are the only relevant ones in the definition of $\approx_d^c$, since $\approx_d$ is preserved by all the other operators. As a matter of fact, if we define:

Definition 5.15 $p \approx_d^+ q$ if for some a not in p, q : $a + p \approx_d a + q$

we have the following useful characterisation for $\approx_d^c$:

Theorem 5.16 $\approx_d^+$ *coincides with* $\approx_d^c$ *over* $\mathcal{P}'$.

Outline of proof: The nontrivial part is $\approx_d^+ \subseteq \approx_d^c$. We prove separately:

i) $\approx_d^+ \subseteq \approx_d$.

ii) $\approx_d^+$ is a congruence.

Then the result will follow, since $\approx_d^c$ is the largest congruence included in $\approx_d$.

To prove point i) it is enough to show that $\approx_d^+$ is a wd–bisimulation. To prove point ii), check that the following relations are wd–bisimulations:

$$R_1 = \{(\mu p + a, \mu q + a) \mid p \approx_d^+ q, a \notin \mu p, \mu q\} \cup \approx_d$$

$$R_2 = \{((p+r) + a, (q+r) + a) \mid p \approx_d^+ q, a \notin p, q, r\} \cup \approx_d$$

$$R_3 = \{((p \mid r) + a, (q \mid r) + a) \mid p \approx_d^+ q, a \notin p, q, r\} \cup \approx_d \quad \square$$

Let us now define – resuming the notation of chapter 2 – a transition relation:

$$\stackrel{\tau}{\Longmapsto} =_{def} \stackrel{\tau^n}{\longrightarrow}, \quad n > 0$$

Thus $\stackrel{\tau}{\Longmapsto}$ involves *at least one* transition $\stackrel{\tau}{\longrightarrow}$.

Now it is immediate to check that $\approx_d^+$ satisfies the property:

Property 5.17 *Let* $p \approx_d^+ q$. *Then* $p \stackrel{\tau}{\Longmapsto} p'$ *implies* $q \stackrel{\tau}{\Longmapsto} q'$ *for some* q' *such that* $p' \approx_d q'$.

Proof: Let $p + a \approx_d q + a$ for some $a \notin p, q$. If $p \stackrel{\tau}{\Longmapsto} p'$ then $p + a \stackrel{\tau}{\Longmapsto} p'$. Now the corresponding move of $q + a$ cannot be $q + a \stackrel{\tau}{\Longrightarrow} q + a$, since $p' \not\stackrel{a}{\Longrightarrow} q'$. We thus have $q + a \stackrel{\tau}{\Longmapsto} q' \sim_d p'$ because $q \stackrel{\tau}{\Longmapsto} q'$. □

Property 5.18 $p \approx_d q$ *if and only if one of the following holds:*

i) $p \approx_d^c q$

ii) $p \approx_d^c \tau q$

iii) $\tau p \approx_d^c q$

Proof: The nontrivial part is the *only if* one. So suppose $p \approx_d q$. We want to show that we are in one of the three cases $i), ii), iii)$. We proceed by case analysis.

a. Suppose that $p \stackrel{\tau}{\Longmapsto} p'$ and the corresponding move of q is $q \stackrel{\tau}{\Longrightarrow} q$, with $p' \approx_d q$. In this case we have $p \approx_d^c \tau q$. In fact it is easy to check that, if $a \notin p, q$, then $p + a \approx_d \tau q + a$: corresponding to the move $p + a \stackrel{\tau}{\Longrightarrow} p'$ of the first term we pick the move $\tau q + a \stackrel{\tau}{\Longrightarrow} q$ of the second term.

b. Symmetrically, if $q \stackrel{\tau}{\Longmapsto} q'$ and p replies with $p \stackrel{\tau}{\Longrightarrow} p$, we have $\tau p \approx_d^c q$.

c. If we are in neither of cases $i)$ and $ii)$, then we have $p + a \approx_d q + a$, that is $p \approx_d^c q$. □

Note that the three alternatives $i), ii), iii)$ of the above property may be summarised as $(\approx_d^c)_a$, where R_a is the function on relations defined in chapter 2 (at page 36). The notation R_a will be used again in the following.

The next section is devoted to the search of an axiomatisation for $\approx_d^c$. The relation $\approx_d^+$ will be extensively used as an alternative formulation of $\approx_d^c$.

5.5 Algebraic Characterisation

In this section we present a complete set of axioms for the weak behavioural equivalence $\approx_d^c$. First, we find that the three τ-laws of [Mil 80, HM 85] are valid for $\approx_d^c$:

$(I1)$ $\qquad x + \tau x = \tau x$

$(I2)$ $\qquad \mu \tau x = \mu x$

$(I3)$ $\qquad \mu(x + \tau y) + \mu y = \mu(x + \tau y)$

These three properties can be easily proven for the equivalence $\approx_d^+$ that was introduced in the previous section as an alternative formulation for $\approx_d^c$. Furthermore, we have a law expressing the globality of τ – actions:

$(I4)$ $\qquad \tau(x \mid y) = \tau x \mid y$

Once more, however, to obtain a complete axiomatisation for our behavioural equivalence we need to recourse to the auxiliary operators $\between$ and $\mid_c$. Thus, for example, the equation (I4) will be derivable from the following two laws of $\between$:

(NI1) $\qquad \tau x \between y = \tau(x \mid y)$

(NI2) $\qquad x \between y = x \between \tau y$

One further law is required for $\between$, which is similar in structure to I3:

(NI3) $\qquad x \between (y + \tau z) + x \between z = x \between (y + \tau z)$

In absence of communication, that is when we let aside rules $\tau 4$ and S7, the operator $\between$ is all we need to derive a complete axiomatisation for $\approx_d$. The behavioural rules for $\between$ are the following:

τ6. $\quad p \xrightarrow{\tau} p'$ implies $p \between q \xrightarrow{\tau} p' \mid q$

S10. $\quad p \stackrel{a}{\Longrightarrow} \langle p', p'' \rangle$ implies $p \between q \stackrel{a}{\Longrightarrow} \langle p', p'' \mid q \rangle$

Let now $\mathcal{F}$ denote the set of axioms $\mathcal{D}$ extended with the τ-laws (I1) – (I3), (NI1) – (NI3). These laws are all grouped together in Figure 5.2.

Axioms $\mathcal{F} = \mathcal{D} \cup \tau-$laws

$(A1)$	$x + (y + z) = (x + y) + z$	Axioms $\mathcal{D}$
$(A2)$	$x + y = y + x$	
$(A3)$	$x + \mathrm{NIL} = x$	
$(A4)$	$x + x = x$	
(PE)	$x \mid y = x \between y + y \between x$	
$(LP1)$	$(x + y) \between z = x \between z + y \between z$	
$(LP2)$	$(x \between y) \between z = x \between (y \mid z)$	
$(LP3)$	$x \between \mathrm{NIL} = x$	
$(LP4)$	$\mathrm{NIL} \between x = \mathrm{NIL}$	
$(I1)$	$x + \tau x = \tau x$	$\tau-$laws
$(I2)$	$\mu\tau x = \mu x$	
$(I3)$	$\mu(x + \tau y) + \mu y = \mu(x + \tau y)$	
$(NI1)$	$\tau x \between y = \tau(x \mid y)$	
$(NI2)$	$x \between y = x \between \tau y$	
$(NI3)$	$x \between (y + \tau z) + x \between z = x \between (y + \tau z)$	

Figure 5.2: Axiomatisation of $\approx_d$, in absence of communication.

We shall state here, as we did for the strong equivalence $\sim_d$, two separate characterisation results, one for the calculus without communication and one for the complete calculus.

Let first Σ_1 denote the enlarged syntax:

$$\Sigma_1 = A_\tau \cup \{\mathrm{NIL}, +, |, \between\}$$

If we extend $\approx_d^c$ to the new calculus, we have the following result:

Theorem 5.19 *In absence of communication, the equivalence $\approx_d^c$ is the Σ_1-congruence generated by the axioms $\mathcal{F}$.*

Again, we do not give the proof of this first theorem, since it is essentially a simpler version of the general theorem 5.23 which follows.

Consider now the whole language T_{Σ_2}, where

$$\Sigma_2 = A_\tau \cup \{\mathrm{NIL}, +, |, \between, |_c\}$$

and $|_c$ is specified by the rule:

S11. $p \xrightarrow{a} <p', p''>$, $q \xrightarrow{\bar{a}} <q', q''>$ imply $p|_c q \xrightarrow{\tau} (p'|p'') \mid (q'|q'')$

The other operators are specified by the whole set of rules $\tau 1$– $\tau 4$, S1–S7. Let us now extend our equivalence $\approx_d^c$ to the new language. Then the laws Δ for $\sim_d$ remain valid for $\approx_d^c$. Corresponding to the rule S7, we need an additional axiom for communication, which we state in the form of an absorption law (where $p \sqsubseteq q$ means $p + q = q$):

$$a(x|y) \between (x'|y') \sqsubseteq a(cx \between x' + v) \between (\bar{c}y \between y' + w) \qquad \text{(CP5)}$$

Let $\Delta' = \Delta \cup \{\mathrm{CP5}\}$. If we denote by $\mathcal{G}$ the set of laws $\mathcal{F}\backslash \mathrm{PE} \cup \Delta'$, as shown in Figure 5.3, we have all the elements to state our general characterisation theorem. We give first the normalisation lemma.

Our *normal forms* are now terms (defined modulo the axioms A1, A2, A3) of the form:

$$\hat{p} = \sum_{i \in I} a_i\, p_i \between p_i' + \sum_{j \in J} \tau\, p_j$$

where $I \cap J = \emptyset$ and for each $i \in I$, $j \in J$, the terms p_i, p_i' and p_j are again *nf's*. By convention $\hat{p} = \mathrm{NIL}$ if both $I = \emptyset$ and $J = \emptyset$.

We have then the following:

Lemma 5.20 (*Normalisation lemma*) $\forall p \in T_{\Sigma_2} \quad \exists\, nf \quad \hat{p} =_{\mathcal{G}} p.$

Proof: By induction on the size of p. The proof makes use of all axioms in $\mathcal{G}$ except for the idempotence law A4 and the set of τ-laws (I1) – (I3), (NI2), (NI3). As usual, we use axioms A1, A2, A3 without mentioning them.

We proceed by case analysis. The main change w.r.t. the analogous proof for $\sim_d$ regards cases v) and vi).

i) NIL is a normal form.

ii) $p = a\,q$. Define $\hat{p} = a\,\hat{q} \between \mathrm{NIL}$. Then $p =_{\mathcal{G}} \hat{p}$ by induction and LP3.

iii) $p = \tau\,q$. Define $\hat{p} = \tau\,\hat{q}$. Then $p =_{\mathcal{G}} \hat{p}$ by induction.

iv) $p = q + r$. We let here $\hat{p} = \hat{q} + \hat{r}$. Then $p =_{\mathcal{G}} \hat{p}$ by simple induction.

v) $p = q \between r$. If $\hat{q} = \mathrm{NIL}$ define $\hat{p} = \mathrm{NIL}$. Then, using induction and axiom LP4 , we get: $p =_{\mathcal{G}} \hat{q} \between r = \mathrm{NIL} \between r =_{\mathcal{G}} \mathrm{NIL} =_{\mathcal{G}} \hat{p}$.

Otherwise, if $\hat{q} = \sum_{i\in I} a_i q_i \between q'_i + \sum_{j\in J} \tau\, q_j \neq \mathrm{NIL}$, we define:

$\hat{p} = \sum_{i\in I} a_i q_i \between (\widehat{q'_i \mid r}) + \sum_{j\in J} \tau\,(\widehat{q_j \mid r})$. We then have:

$$
\begin{aligned}
p \;&=_{\mathcal{G}}\; (\textstyle\sum_{i\in I} a_i q_i \between q'_i + \sum_{j\in J} \tau q_j) \between r \\
&=_{\mathcal{G}}\; \textstyle\sum_{i\in I} \left[(a_i q_i \between q'_i) \between r\right] + \sum_{j\in J} (\tau q_j \between r) \\
&=_{\mathcal{G}}\; \textstyle\sum_{i\in I} a_i q_i \between (q'_i \mid r) + \sum_{j\in J} (\tau q_j \between r) \\
&=_{\mathcal{G}}\; \textstyle\sum_{i\in I} a_i q_i \between (\widehat{q'_i \mid r}) + \sum_{j\in J} \tau\,(\widehat{q_j \mid r}) \;=\; \hat{p}
\end{aligned}
$$

using induction and the laws LP1, LP2, NI1.

vi) $p = q\,|_c\, r$. If $\hat{r} = \mathrm{NIL}$ we let $\hat{p} = \mathrm{NIL}$. Then, by induction and axiom CP3 we have: $p =_{\mathcal{G}} q\,|_c\,\hat{r} = q\,|_c\,\mathrm{NIL} =_{\mathcal{G}} \mathrm{NIL}$. We proceed similarly, using CP2 first, when $\hat{q} = \mathrm{NIL}$.

Otherwise, if we have both $\hat{q} = \sum_{i\in I} a_i q_i \between q'_i + \sum_{j\in J} \tau q_j \neq \mathrm{NIL}$ and $\hat{r} = \sum_{k\in K} b_k r_k \between r'_j + \sum_{l\in L} \tau r_l \neq \mathrm{NIL}$, we define $\hat{p} = \sum_{a_i = b_k} \tau\,(\widehat{q_i \mid q'_i \mid r_k \mid r'_k})$. Then, using induction and axioms CP1, CP2, CP3, CP4, NI1 we obtain:

$$
\begin{aligned}
p &=_{\mathcal{G}} \left(\Sigma_{i\in I}\, a_i q_i \between q'_i + \Sigma_{j\in J}\, \tau q_j\right) \mid_c \left(\Sigma_{k\in K}\, b_k r_k \between r'_k + \Sigma_{l\in L}\, \tau r_l\right) \\
&=_{\mathcal{G}} \Sigma_{i\in I}\, a_i q_i \between q'_i \mid_c \Sigma_{k\in K}\, b_k r_k \between r'_k + \Sigma_{j\in J}\, \tau q_j \mid_c \Sigma_{l\in L}\, \tau r_l + \\
&\quad + \Sigma_{j\in J}\, \tau q_j \mid_c \Sigma_{k\in K}\, b_k r_k \between r'_k + \Sigma_{i\in I}\, a_i q_i \between q'_i \mid_c \Sigma_{l\in L}\, \tau r_l \\
&=_{\mathcal{G}} \Sigma_{a_i=\bar{b}_k}\, \tau\,(q_i \mid r_k) \between (q'_i \mid r'_k) + \mathrm{NIL} + \mathrm{NIL} + \mathrm{NIL} \\
&=_{\mathcal{G}} \Sigma_{a_i=\bar{b}_k}\, \tau\,(q_i \mid q'_i \mid r_k \mid r'_k) \;=_{\mathcal{G}}\; \hat{p}
\end{aligned}
$$

vii) $p = q \mid r$. Let $p_1 = q \between r$, $p_2 = r \between q$ and $p_3 = q \mid_c r$. Define $\hat{p} = \hat{p}_1 + \hat{p}_2 + \hat{p}_3$, where $\hat{p}_1$ and $\hat{p}_2$ and $\hat{p}_3$ are given by cases v) and vi). Then, using induction and the expansion law CPE, we obtain:

$$p =_{\mathcal{G}} p_1 + p_2 + p_3 =_{\mathcal{G}} \hat{p}_1 + \hat{p}_2 + \hat{p}_3 =_{\mathcal{G}} \hat{p} \qquad \square$$

The proof of our completeness result is more complicated than for $\sim_d$. Besides a normalisation lemma, it also requires two *absorption lemmas*, which we prove next. The first one is a *τ–absorption* lemma, which holds in the same form for weak CCS transitions, see [HM 85]. We recall that $\stackrel{\tau}{\Longmapsto} = \stackrel{\tau^n}{\longrightarrow}$, $n > 0$.

Lemma 5.21 (*τ-absorption lemma*)

If p is a nf then: $p \stackrel{\tau}{\Longmapsto} p'$ *implies* $p + \tau p' =_{\mathcal{G}} p$

Proof: By induction on the size of p. The proof uses axioms A4 and I1.

If $p = \Sigma_{i\in I}\, a_i p_i \between p'_i + \Sigma_{j\in J}\, \tau p_j$, then the transition $\stackrel{\tau}{\Longmapsto}$ is due to the part $\Sigma_{j\in J}\, \tau p_j$ of p. Now, there are two possibilities:

1. $p_j = p'$, for some $j \in J$. Then $p =_{\mathcal{G}} p + \tau p'$ by A4.

2. $p_j \stackrel{\tau}{\Longmapsto} p'$, for some $j \in J$. By induction $p_j =_{\mathcal{G}} p_j + \tau p'$. Then using axiom I1 we obtain:

$$p =_{\mathcal{G}} p + \tau p_j =_{\mathcal{G}} p + \tau p_j + p_j =_{\mathcal{G}} p + \tau p_j + p_j + \tau p' =_{\mathcal{G}} p + \tau p'$$

$\square$

The second absorption lemma, which is slightly more involved, is the analogue for weak distributed transitions of the *generalised absorption* lemma we gave in Chapter 2 for weak CCS transitions.

Lemma 5.22 (*Generalised absorption lemma*)

If p *is a nf then:* $p \stackrel{a}{\Longrightarrow} \langle p', p'' \rangle$ *implies* $p + ap' \between p'' =_{\mathcal{G}} p$

Proof: By induction on the length of the proof that $p \stackrel{a}{\Longrightarrow} \langle p', p'' \rangle$. It uses axioms A4, I1, LP1, CP5 and the above τ–absorption lemma.

Let $p = \sum_{i \in I} a_i p_i \between p_i' + \sum_{j \in J} \tau p_j$. There are then two main possibilities for $p \stackrel{a}{\Longrightarrow} \langle p', p'' \rangle$.

1. $p \stackrel{a}{\Longrightarrow} \langle p', p'' \rangle$ because for some $j \in J$: $\tau p_j \stackrel{\tau}{\longrightarrow} p_j \stackrel{a}{\Longrightarrow} \langle p', p'' \rangle$. By induction $p_j + ap' \between p'' =_{\mathcal{G}} p_j$. Whence we deduce, using I1:

$$\begin{aligned} p + ap' \between p'' &= p + \tau p_j + ap' \between p'' \\ &=_{\mathcal{G}} p + \tau p_j + p_j + ap' \between p'' \\ &=_{\mathcal{G}} p + \tau p_j + p_j =_{\mathcal{G}} p \end{aligned}$$

2. $p \stackrel{a}{\Longrightarrow} \langle p', p'' \rangle$ because for some $i \in I$ we have $a = a_i$ and $p \stackrel{a}{\longrightarrow} \langle p_i, p_i' \rangle \stackrel{\tau}{\Longrightarrow} \langle p', p'' \rangle$. Now, consider the chain of τ–transitions leading from $\langle p_i, p_i' \rangle$ to $\langle p', p'' \rangle$. If this chain is empty, we have $a = a_i$, $p' = p_i$, $p'' = p_i'$ for some $i \in I$, and get our result by simple absorption A4.

Otherwise, consider the last application of one of the rules S5, S6, S7 in the derivation of $p \stackrel{a}{\Longrightarrow} \langle p', p'' \rangle$. The general form of this application is:

$$p \stackrel{a}{\Longrightarrow} \langle q', q'' \rangle \stackrel{\tau}{\longrightarrow} \langle p', p'' \rangle \quad \text{implies} \quad p \stackrel{a}{\Longrightarrow} \langle p', p'' \rangle$$

where we know by induction that $p + aq' \between q'' =_{\mathcal{G}} p$.

Now, according to which of the rules S5, S6, S7 is actually applied, we have three different cases. For each case we will just show that :

$$aq' \between q'' = aq' \between q'' + ap' \between p''$$

since from this we can deduce:

$$p =_{\mathcal{G}} p + aq' \between q'' =_{\mathcal{G}} p + aq' \between q'' + ap' \between p'' =_{\mathcal{G}} p + ap' \between p''$$

Let us now examine the three cases:

i) The rule applied is S5: that is $<q', q''> \xrightarrow{\tau} <p', p''>$ because $q' \xrightarrow{\tau} p'$ and $q'' = p''$. Then we have $q' =_{\mathcal{G}} q' + \tau p'$ by τ–absorption lemma. Whence we deduce, using axiom I3:

$$aq' =_{\mathcal{G}} a(q' + \tau p') =_{\mathcal{G}} a(q' + \tau p') + ap' =_{\mathcal{G}} aq' + ap'$$

We then have, using axiom LP1:

$$\begin{aligned} aq' \between q'' &=_{\mathcal{G}} (aq' + ap') \between q'' \\ &=_{\mathcal{G}} aq' \between q'' + ap' \between q'' \\ &= aq' \between q'' + ap' \between p'' \end{aligned}$$

ii) The rule applied is S6: $<q', q''> \xrightarrow{\tau} <p', p''>$ because $q'' \xrightarrow{\tau} p''$ and $q' = p'$. Again we may use the τ–absorption lemma to get $q'' =_{\mathcal{G}} q'' + \tau p''$. This implies, by axiom I3:

$$aq'' =_{\mathcal{G}} a(q'' + \tau p'') =_{\mathcal{G}} a(q'' + \tau p'') + ap'' =_{\mathcal{G}} aq'' + ap''$$

We may now deduce, using axiom NI3:

$$\begin{aligned} aq' \between q'' &=_{\mathcal{G}} aq' \between (q'' + \tau p'') \\ &=_{\mathcal{G}} aq' \between (q'' + \tau p'') + aq' \between p'' \\ &=_{\mathcal{G}} aq' \between q'' + aq' \between p'' \\ &= aq' \between q'' + ap' \between p'' \end{aligned}$$

iii) The rule applied is S7: here $<q', q''> \xrightarrow{\tau} <p', p''>$ because $q' \xrightarrow{c} <r, r'>$ and $q'' \xrightarrow{\bar{c}} <s, s'>$, with $r \mid s = p'$ and $r' \mid s' = p''$. By induction $q' =_{\mathcal{G}} q' + cr \between r'$ and $q'' =_{\mathcal{G}} q'' + \bar{c}s \between s'$. We then have, using the law CP5:

$$\begin{aligned} aq' \between q'' &=_{\mathcal{G}} a(q' + cr \between r') \between (q'' + \bar{c}s \between s') \\ &=_{\mathcal{G}} a(q' + cr \between r') \between (q'' + \bar{c}s \between s') + a(r \mid s) \between (r' \mid s') \\ &= aq' \between q'' + ap' \between p'' \end{aligned}$$

□

With the help of these lemmas we can finally prove our characterisation result:

Theorem 5.23 *The equivalence* $\approx_d^c$ *is the* Σ_2*-congruence* $=_{\mathcal{G}}$ *generated by the axioms* $\mathcal{G}$.

Proof. As for theorem 4.20 the *soundness* of the axioms, i.e. the implication: $p =_{\mathcal{G}} q \implies p \approx_d^c q$ is relatively easy to prove.

We give here the proof of *completeness*: $p \approx_d^c q$ implies $p =_{\mathcal{G}} q$. The proof is by induction on the sum of sizes of p, q. We shall here, to simplify the notation, write $\approx^c$ instead of $\approx_d^c$. Suppose then that $p \approx^c q$. By virtue of the normalisation lemma we may assume p, q to be normal forms:

$$p = \sum_{i\in I} a_i\, p_i \between p_i' + \sum_{j\in J} \tau p_j, \qquad q = \sum_{n\in N} b_n\, q_n \between q_n' + \sum_{m\in M} \tau q_m$$

We proceed with our usual method. We show that $q + p =_{\mathcal{G}} q$ and the result will follow by symmetry. We prove separately:

i) $q + \tau p_j =_{\mathcal{G}} q \qquad \forall j \in J$

ii) $q + a_i p_i \between p_i' =_{\mathcal{G}} q \qquad \forall i \in I$

Proof of i). We have $p \overset{\tau}{\longmapsto} p_j$. Correspondingly, since $p \approx^c q$, there exists r s.t. $q \overset{\tau}{\longmapsto} r$ and $p_j \approx_a^c r$. By induction $p_j\ (=_{\mathcal{G}})_a\ r$ and thus, by axiom I2, $\tau p_j =_{\mathcal{G}} \tau r$. We may now use the τ–absorption lemma to get: $q =_{\mathcal{G}} q + \tau r =_{\mathcal{G}} q + \tau p_j$.

Proof of ii). We have here $p \overset{a_i}{\Longrightarrow} <p_i, p_i'>$. Since $p \approx^c q$, there must exist r s.t. $q \overset{a_i}{\Longrightarrow} <r, r'>$, with $p_i \approx_a^c r$ and $p_i' \approx_a^c r'$. By induction $p_i\ (=_{\mathcal{G}})_a\ r$ and $p_i'\ (=_{\mathcal{G}})_a\ r'$. Then using axiom I2 we get $\tau p_i =_{\mathcal{G}} \tau r$ and $\tau p_i' =_{\mathcal{G}} \tau r'$. At this point we may use axioms I2 and NI2 to obtain:

$$\begin{aligned} a_i p_i \between p_i' &=_{\mathcal{G}} a_i\, \tau\, p_i \between p_i' &&=_{\mathcal{G}} a_i\, \tau\, r \between p_i' \\ &=_{\mathcal{G}} a_i\, r \between p_i' &&=_{\mathcal{G}} a_i\, r \between \tau\, p_i' \\ &=_{\mathcal{G}} a_i\, r \between \tau\, r' &&=_{\mathcal{G}} a_i\, r \between r' \end{aligned}$$

whence, by the generalised absorption lemma, we finally deduce:

$$q =_{\mathcal{G}} q + a_i r \between r' =_{\mathcal{G}} q + a_i p_i \between p_i' \qquad \square$$

Axioms $\mathcal{G} = \mathcal{F} \setminus PE \cup \Delta'$

$$
\begin{array}{lll}
(A1) & x + (y + z) = (x + y) + z & \left.\begin{array}{l} \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \end{array}\right. \\
(A2) & x + y = y + x & \\
(A3) & x + \mathrm{NIL} = x & \\
(A4) & x + x = x & \\
(LP1) & (x + y) \between z = x \between z + y \between z & \\
(LP2) & (x \between y) \between z = x \between (y \mid z) & \\
(LP3) & x \between \mathrm{NIL} = x & \\
(LP4) & \mathrm{NIL} \between x = \mathrm{NIL} & \mathcal{F} \setminus PE \\
(I1) & x + \tau x = \tau x & \\
(I2) & \mu \tau x = \mu x & \\
(I3) & \mu (x + \tau y) + \mu y = \mu (x + \tau y) & \\
(NI1) & \tau x \between y = \tau (x \mid y) & \\
(NI2) & x \between y = x \between \tau y & \\
(NI3) & x \between (y + \tau z) + x \between z = x \between (y + \tau z) & \left.\begin{array}{l} \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \end{array}\right.
\end{array}
$$

$$
\begin{array}{lll}
(CPE) & x \mid y = x \between y + y \between x + x \mid_c y & \left.\begin{array}{l} \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \end{array}\right. \\
(CP1) & (x + y) \mid_c z = (x \mid_c z) + (y \mid_c z) & \\
(CP2) & x \mid_c y = y \mid_c x & \\
(CP3) & x \mid_c \mathrm{NIL} = \mathrm{NIL} & \\
(CP4) & (\mu x \between x') \mid_c (\nu y \between y') = \begin{cases} \tau (x \mid y) \between (x' \mid y'), & \text{if } \mu = \bar{\nu} \\ \mathrm{NIL}, & \text{otherwise} \end{cases} & \Delta' \\
(CP5) & a (x \mid y) \between (x' \mid y') \sqsubseteq a (cx \between x' + v) \between (\bar{c} y \between y' + w) & \left.\begin{array}{l} \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \end{array}\right.
\end{array}
$$

Figure 5.3: Axiomatisation of $\approx_d$, with communication.

From our axioms $\mathcal{G}$ we may deduce the following *expansion theorem* (GEXP):

$$\textit{Let } x = \Sigma_{i\in I}\, a_i x_i \between x'_i + \Sigma_{j\in J}\, \tau z_j\,,\ y = \Sigma_{n\in N}\, b_n y_n \between y'_n + \Sigma_{m\in M}\, \tau w_m$$

$$\begin{aligned}\textit{Then } x \mid y \;=\; & \Sigma_{i\in I}\, a_i\, x_i \between (x'_i \mid y) + \Sigma_{n\in J}\, b_n\, y_n \between (x \mid y'_n) \\ & + \Sigma_{j\in J}\, \tau\,(z_j \mid y) \;+\; \Sigma_{m\in M}\, \tau\,(x \mid w_j) \\ & + \sum_{a_i = \bar{b}_n} \tau\,(x_i \mid y_n \mid x'_i \mid y'_n)\end{aligned}$$

To sum up, we have established a finite complete axiomatisation for our behavioural congruence $\approx^c_d$.

Here again, as for $\sim_d$, we could dispense with the communication operator $|_c$. In fact, if we replace all axioms involving $|_c$ in figure 5.3 by the expansion theorem (GEXP), we obtain an axiomatisation for $\approx^c_d$ which is still complete – although not finite.

On the other hand it would be out of question to eliminate $|_c$ and simply introduce a communication rule for $\between$. As it were, allowing communication across $\between$ would render *unsound* our axiom LP2 – which is needed, among other things, for the reduction of terms to normal forms.

For assume we had a new rule for $\between$, similar to $\tau 4$. Then the two processes in the following example would not have the same behaviour, since the second would have a transition $\stackrel{\tau}{\Longrightarrow}$ leading to (a process equivalent to) NIL, which the first one could not match:

Example 8) $\quad (a \between b) \between (\bar{b} \between \bar{a}) \;=_{\mathrm{LP2}}\; a \between (b \mid (\bar{b} \between \bar{a})$

One last remark. We justified the use of global unobservable transitions $p \stackrel{\tau}{\Longrightarrow} p'$ rather than distributed ones $p \stackrel{\tau}{\Longrightarrow} <p', p''>$ by saying that not only the presence, but also the *locality* of an action τ should not be directly observable. Now, in the light of the previous study, we may argue that there is also a technical reason for taking $\stackrel{\tau}{\Longrightarrow}$ to be global. A weak d–bisimulation $\equiv_d$ based on distributed transitions $p \stackrel{\tau}{\Longrightarrow} <p', p''>$ would not be preserved by $\mid$, as shown by the following example:

$$a \equiv_d \tau a \quad \text{but} \quad a \mid b \not\equiv_d \tau a \mid b$$

since the first process would have only one τ–transition, namely: $a \mid b \stackrel{\tau}{\Longrightarrow} <a \mid b, \mathrm{NIL}>$, whereas the second would have a further possibility: $\tau\, a \mid b \stackrel{\tau}{\Longrightarrow} <a, \mathrm{NIL} \mid b>$.

On the other hand, we saw that the substitutivity of $\approx_d$ w.r.t. $\mid$ was crucial for some of our results, notably for the equivalence of our two semantics.

We conclude here our treatment of distributed bisimulations. In the next chapter, we shall present a somewhat more standard notion of bisimulation, called *pomset* bisimulation, which also leads to a non–interleaved model for our concurrent language.

Chapter 6

Pomset transition systems

In this chapter we present a new behavioural equivalence for concurrent processes, called *pomset bisimulation equivalence* in that it is based on transitions labelled by partially ordered multisets (*pomsets*, in the terminology of Pratt and Gischer [Pra 82, Gis 84]).

The idea underlying this new equivalence is very simple. We noted already that in a labelled transition system S the action η labelling a transition

$$S \xrightarrow{\eta} S'$$

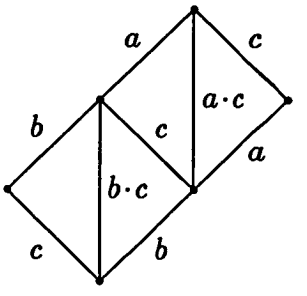
may be seen to represent a *computation* of the system S. In the case of sequential systems, a computation may always be decomposed in a *sequence* of elementary steps: thus transitions labelled by *atomic* actions are enough to describe the behaviour of such systems.

On the other hand we know from our discussion in chapter 3 that elementary transitions are not sufficient to account for nonsequential behaviours. At least we need to enrich the transitions with some information about the concurrent structure of processes. This was achieved in our distributed transition systems of chapters 4 and 5 by associating with each elementary transition an indication of "where" the action had been performed.

We shall follow here a different approach. If we want actions to represent nonsequential computations, we need to endow them with some internal *structure*. In doing this, we will relax the requirement that actions be atomic – both in time and in space.

We saw in chapter 3 that the calculi MEIJE (and SCCS) allow compound actions made out of concurrent atomic actions. By introducing *concurrency* in actions, one gives up their spatial atomicity. However actions are still atomic in time: each parallel component may contribute *at most one* action to a computational step.

Consider for example the process $p = (a:b) \mid c$ (described in our usual syntax). This is interpreted in MEIJE as the following transition system:



Here the action c may occur in parallel with the action a or in in parallel with the action b, but there is no way of saying that c happens in parallel with the whole sequence $a:b$.

In other words the calculus MEIJE, while allowing a complete asynchrony among concurrent *atomic* actions, still retains a notion of *global time* that creates artificial dependencies among actions.

In fact, if we look at the diagram above, we will notice that each computation introduces some sequentiality among actions which is not prescribed by the syntax of p. For example, if we use $\bullet$ to denote the sequencing of transitions (e.g. $a \bullet b$ will mean $\xrightarrow{a}\xrightarrow{b}$) we have a computation $a \bullet b \bullet c$ introducing dependencies between a and c, b and c, another computation $a \bullet (b \cdot c)$ introducing a dependency between a and c, and so on. As a consequence we will have the following identification:

Example 1) $\qquad (a:b) \mid c \;=\; (a:b) \mid c \;+\; a:b:c \;+\; a:(b \mid c)$

Intuitively, we would like to distinguish here the causal sequences $a:b:c$ and $a:(b \mid c)$ from the computational sequences $a \bullet b \bullet c$ and $a \bullet (b \cdot c)$ – which may be due to the term $(a:b) \mid c$.

The simplest way to obtain this is to allow the use of prefixing (as well as concurrent composition) inside actions. Of course an occurrence of " : " within an action will only reflect an occurrence of " : " in the process.

We will then be able to say that a sequence of actions occurs as a whole in one step. So for instance the right member of example 1), call it q, will have a transition $q \xrightarrow{a:c} b$, whereas the left member p only has a sequence of transitions $p \xrightarrow{a} \xrightarrow{c} b$. In other words, we may now distinguish the "transition of a sequence" from a sequence of transitions.

We come in this way to a *generalised* notion of *action*, not necessarily atomic neither in space nor in time. In fact, by dropping the requirement of atomicity for actions, we lose much of the distinction between actions and processes. If we decide to denote by $|$ the concurrent composition of actions, an action becomes itself a process, built using only the operators : and $|$.

Hence an action is now a deterministic process. This appears to agree with our concept of nonsequential computation as *partial order* of atomic actions or events. It is therefore not surprising that, if we reconsider our interpretation of processes as *labelled event structures* as was given in chapter 3, we find an exact correspondence between the actions of a process and the computations (or configurations) of the associated event structure.

This chapter is organised as follows. In section 6.1 we present our new operational semantics for processes, based on the generalised transitions. The syntax for processes is essentially the same as in chapter 4, the operators being prefixing, sum and parallel composition. However the calculus is simpler here, in that we do not consider communication nor τ–actions (an extension of this semantics to communication is proposed in [BC 87b]).

In section 6.2, we show that our semantics agrees with an (intuitive) transition system semantics for labelled event structures, where transitions are labelled by configurations.

We then turn to the study of the new bisimulation $\asymp$, called *pomset bisimulation equivalence.* In section 6.3 we give a finite complete axiomatisation for $\asymp$. Finally, in section 6.4 we establish the relation between $\asymp$ and the bisimulations examined in previous chapters: it turns out that $\asymp$ is strictly included between our distributed bisimulation $\sim_d$ and Milner's strong bisimulation $\sim$.

6.1 Pomset bisimulation equivalence

In this section we set up our new calculus for processes. We adopt the same syntax Σ used in earlier chapters:

$$\Sigma \;=\; A \cup \{\,\mathrm{NIL}\,,\,+,\,|\,\}$$

As usual processes are terms of $\mathcal{P} = T_\Sigma$, denoted by p, q, r, etc. The elements of A are the *atoms* (or atomic actions) of our calculus. We let $a, b, \ldots$ range over A. General *actions* – the labels of transitions – will be terms over the restricted syntax $A \cup \{\,\mathrm{NIL}\,,\,|\,\}$. The set of actions, ranged over by $u, v, \ldots$, will be called Act.

We shall mostly abbreviate $a\!:\!\mathrm{NIL}$ to a (both for processes and for actions).

The behaviour of processes is specified by the following rules.

Rules $\mathcal{V}$

V1. $a\!:\!p \xrightarrow{a\,:\,\mathrm{NIL}} p$

V2. $p \xrightarrow{u} p'$ implies $a\!:\!p \xrightarrow{a\,:\,u} p'$

V3. $p \xrightarrow{u} p'$ implies $p + q \xrightarrow{u} p'$, $q + p \xrightarrow{u} p'$

V4. $p \xrightarrow{u} p'$ implies $p \mid q \xrightarrow{u} p' \mid q$, $q \mid p \xrightarrow{u} q \mid p'$

V5. $p \xrightarrow{u} p'$, $q \xrightarrow{v} q'$ imply $p \mid q \xrightarrow{u\,|\,v} p' \mid q'$

Rules V1, V3, V4 are essentially the same as for CCS transitions. Let us then focus on rules V2 and V5, which build compound actions above the arrows. Note that rule V5, used to build concurrent actions, was encountered already in the calculus MEIJE.

What is to be stressed here is that rules V2 and V5 merely *transfer* the constructors : and | from processes to actions. Thus V2 is substantially different from the rule:

(*) $\qquad p \xrightarrow{a} p'\,,\ p' \xrightarrow{u} p'' \qquad$ implies $\qquad p \xrightarrow{a:u} p''$

which was used for building the weak transitions $\stackrel{\mu}{\Longrightarrow}$ in CCS (as well as the compound weak transitions $\stackrel{\sigma}{\Longrightarrow}$ of chapter 2).

In fact V2 is our key rule for distinguishing causal sequences from purely computational ones. On the other hand rule V5 is necessary to detect the presence of concurrency.

Let us consider an example. The two processes $r = a|b$ and $s = a:b + b:a$ have the following transitions:

Example 2)

i) $r \xrightarrow{a}$ NIL $|\, b \xrightarrow{b}$ NIL

$\xrightarrow{b}\ a \,|$ NIL $\xrightarrow{a}$ NIL

$\xrightarrow{a|b}$ NIL | NIL

ii) $s \xrightarrow{a} b \xrightarrow{b}$ NIL

$\xrightarrow{b}\ a \xrightarrow{a}$ NIL

$\xrightarrow{a:b}$ NIL

$\xrightarrow{b:a}$ NIL

Here we have a whole set of transitions discriminating between r and s: the last transition of r, labelled by the concurrent action $a \mid b$, and the last two transitions of s, labelled by sequences of actions.

In fact, our semantics does not only distinguish between r and s. If we let $t = r + s$, rule V2 will allow us to differentiate the two processes:

$$r = a|b \quad \text{and} \quad t = a|b + a:b + b:a$$

whereas rule V5 will make the distinction between:

$$s = a:b + b:a \quad \text{and} \quad t = a|b + a:b + b:a$$

Consider now the set of transitions of the process $p = (a:b) \mid c$ examined in the introduction:

Example 3)

$$
\begin{aligned}
p = (a:b) \mid c \quad &\xrightarrow{a} \quad b \mid c \\
&\xrightarrow{c} \quad a:b \\
&\xrightarrow{a:b} \quad c \\
&\xrightarrow{a \mid c} \quad b \\
&\xrightarrow{(a:b) \mid c} \quad \mathrm{NIL}
\end{aligned}
$$

Note that p has a transition labelled with itself. In fact, since any computation is a possible action, every *deterministic process* may execute itself in one step:

Fact 6.1 *If p is a process over $\Sigma = A \cup \{\mathrm{NIL}, \mid\}$, then $p \xrightarrow{p} t$, where t is some (terminated) process over $\{\mathrm{NIL}, \mid\}$.*

We shall now introduce our new bisimulation equivalence. To this end, we first define an *equality* $\equiv$ on actions of Act. Let $\equiv$ be the congruence generated by the axioms:

$$
\begin{aligned}
&P1. \quad u \mid v = v \mid u \\
&P2. \quad u \mid (v \mid w) = (u \mid v) \mid w
\end{aligned}
$$

Definition 6.2 *A pomset bisimulation (p-bisimulation) is a relation $R \subseteq (\mathcal{P} \times \mathcal{P})$ satisfying, for any $(p,q) \in R$ and $u \in Act$, the following property :*

i) $p \xrightarrow{u} p' \implies q \xrightarrow{v} q'$, *with* $u \equiv v$ *and* $p' R q'$

ii) $q \xrightarrow{v} q' \implies p \xrightarrow{u} p'$, *with* $u \equiv v$ *and* $p' R q'$

If $C(R)$ is the set of pairs (p,q) satisfying clauses i) and ii), we may sum up the definition as: a relation $R \subseteq (\mathcal{P} \times \mathcal{P})$ is a p-bisimulation iff $R \subseteq C(R)$.

As is now standard, we take our behavioural equivalence – which we call *pomset bisimulation equivalence* and denote by $\asymp$ – to be the union of all bisimulations:

Definition 6.3 *Let* $p, q \in \mathcal{P}$. Then $p \asymp q$ *iff* $\exists$ *p-bisimulation* R *s.t.* $p\,R\,q$.

It is easy to check that that $\asymp$ is preserved by the operators of Σ:

Property 6.4 $\sim_d$ *is a* Σ*-congruence.* □

We give next some examples of identifications modulo $\asymp$. Here again, it is convenient to define the *absorption* relation generating the equivalence, namely: $p \subsetneq q \iff (p+q) \asymp q$.

Example 5)

$$a:b + b:a \;\not\asymp\; a \mid b + a:b + b:a \;\not\asymp\; a \mid b \;\asymp\; a \mid b + a \mid b$$

Concerning the first three processes, we noted already in section 6.1 that their sets of transitions are pairwise distinct.

Example 6) $\quad a:(b \mid c) \quad \not\subseteq \quad (a:b) \mid c$

one distinguishing action being $(a:c)$ from the left member.

Example 7) $\quad a \mid b \quad \subsetneq \quad (a + c) \mid b \;+\; a \mid (b + d)$

Here the transitions a and b of the left member are matched respectively by the first and the second term in the right member, while the transition $a \mid b$ is matched by either summand in the right member.

Note that the above examples and counterexamples hold as well for our distributed bisimulation $\sim_d$ of chapter 4.

It is in fact conceivable, given their common attempt to distinguish causality from concurrency, that the *pomset bisimulation* $\asymp$ and the *distributed bisimulation* $\sim_d$ be just alternative formulations of the same notion of equivalence. We shall see in section 6.4 that this is not the case. It will turn out that $\asymp$ is less discriminating than $\sim_d$, and also – but this is no surprise – that $\asymp$ is more discriminating than Milner's strong bisimulation $\sim$.

6.2 A corresponding bisimulation equivalence for Labelled Event Structures

We show in this section that our pomset semantics for terms is compatible with a more abstract transition system semantics for *labelled event structures*, where transitions are labelled by *configurations* of events, as defined by Winskel.

Let us first recall, from chapter 3, the definition of (finite) LES.

Definition 6.5 *Let A be a non-empty set. A finite A-labelled event structure (A-LES) is a structure $S = (E, \leq, \#, \lambda)$ where*

i) *E is a finite set of events*

ii) *$\leq$ is a partial ordering on E, called the* causality *relation*

iii) *$\# \subseteq (E \times E) - (\leq \cup \geq)$ is a symmetric* conflict *relation, satisfying the property of* hereditariness: *$e \# e' \leq e''$ implies $e \# e''$*

iv) *$\lambda : E \longrightarrow A$ is the labelling function*

Two events in E are said to be *concurrent*, noted $\smile$, if they are neither comparable nor in conflict. That is:

$$\smile =_{def} (E \times E) - (\leq \cup \geq \cup \#)$$

Let $\mathcal{L}_A$ be the class of (finite) A-LES's. When the set A is fixed, A-LES's will be called simply LES's and we will write $\mathcal{L}$ in place of $\mathcal{L}_A$.

Again, we shall be mostly concerned with *isomorphism classes* of structures. We denote by $\rightleftharpoons$ the relation of isomorphism on LES's, which we assume to be clear to the reader.

As they stand, LES's are a static model: what is needed next is some mechanism of execution, which might serve as the basis for a definition of (possibly abstract) behaviour of a LES.

To this purpose, we introduce a notion of *computation* for LES's. This will enable us to describe LES's as labelled transition systems, whose transitions are labelled by computations.

Intuitively, a computation of a LES S is a deterministic beginning of S – since choices are resolved while a system computes.

Also, the actual computation does not keep trace of the structure of choices in the underlying LES (the detection of branching points being as usual deferred to the bisimulation equivalence).

As a matter of fact, our notion of computation is exactly Winskel's notion of *configuration* for a LES [Win 80] . The definition follows.

Definition 6.6 *Given an A-labelled event structure $S = (E, \leq, \#, \lambda)$ a computation of S is a substructure $U = S \restriction F$ such that:*

i) $F \subseteq E$,

ii) $S \restriction F$ *is conflict-free:* $e \in F \ \& \ e' \in F$ *implies* $\neg(e \# e')$

iii) $S \restriction F$ *is closed under causes:* $e \in F \ \& \ e' \leq e$ *implies* $e' \in F$

Note that the empty structure NIL is a computation of any LES.

Let *Comp* be the class of conflict–free LES's in $\mathcal{L}$. Then one may regard *Comp* as the class of possible computations of LES's.

Note that the isomorphism class of a computation is a partially ordered multiset of (atomic) labels, that is a pomset over A.

As announced earlier, our intention is to describe LES's operationally as performing a sequence of computations in *Comp*. To do this, we still need to define what remains of a LES after a computation, i.e. the *residual* of a LES by a computation. For $S = (E, \leq, \#, \lambda)$, $F \subseteq E$, define first:

$$\#(F) = \{e \in E \mid \exists\, e' \in F \ s.t. \ e \# e'\}$$

Definition 6.7 *Let $S = (E, \leq, \#, \lambda)$ be an A-labelled event structure and $U = S \restriction F$ a computation of S. The residual of S by U, denoted S/U, is the substructure:*

$$S \restriction [E - (F \cup \#(F))]$$

We may now define transitions of the form: $S \xrightarrow{U} S'$ on LES's, where U is some computation of S and $S' = S/U$. We thus obtain a description of LES's as labelled transition systems, where labels are elements of *Comp*.

Using these transitions, we shall now define a notion of bisimulation on LES's.

Definition 6.8 *A LES-bisimulation is a relation* $R \subseteq (\mathcal{L} \times \mathcal{L})$ *satisfying, for any* $(S,T) \in R$ *and* $U,V \in Comp$, *the following property :*

i) $S \xrightarrow{U} S' \implies T \xrightarrow{V} T'$, *with* $U \rightleftharpoons V$ *and* $S' \; R \; T'$

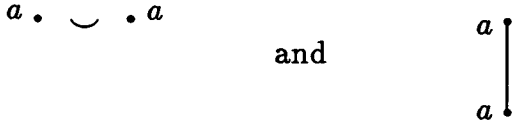
ii) $T \xrightarrow{V} T' \implies S \xrightarrow{U} S'$, *with* $U \rightleftharpoons V$ *and* $S' \; R \; T'$

Let us denote by $\asymp_{\mathcal{L}}$ the union of all LES–bisimulations.

It is easy to see that any conflict–free LES's S is a computation of itself, namely $S \xrightarrow{S}$ NIL.

Hence, when $S,T \in Comp$, we have $S \asymp_{\mathcal{L}} T \iff S \rightleftharpoons T$, since the two maximal computations $S \xrightarrow{S}$ NIL and $T \xrightarrow{T}$ NIL are necessarily in correspondence.

Note on the other hand that we could not use R in place of $\rightleftharpoons$ *within* the definition of LES–bisimulation, since this would imply, for example, the identification of the two LES's:



Our next concern will be to relate our semantics for LES's with the semantics for terms we presented in the previous section.

We gave in chapter 3 an interpretation of terms of T_Σ as LES's. We refer the reader there for the definition of the operators of Σ on LES's. We shall use here $L(p)$ to denote the LES interpretation of the process p.

We show next that our semantics for LES's is compatible with the pomset semantics for processes. This is not surprising, if we note that computations reflect exactly the relations of causality and concurrency in the underlying LES.

We start by showing that there is an exact correspondence between transitions on terms and transitions on LES's.

Theorem 6.9 *Let* $p, p' \in \mathcal{P}$, $P' \in \mathcal{L}$, $U \in Comp - \{\text{NIL}\}$. *Then:*

i) $p \xrightarrow{u} p' \implies L(p) \xrightarrow{U} P'$, where $U \rightleftharpoons L(u)$ and $P' \rightleftharpoons L(p')$.

ii) $L(p) \xrightarrow{U} P' \implies p \xrightarrow{u} p'$, where $U \rightleftharpoons L(u)$ and $P' \rightleftharpoons L(p')$.

Outline of proof: We only sketch the proof, which is somewhat tedious but presents no difficulty.

The statement *i*) is proved by using induction on the structure of p. It essentially amounts to proving that LES's (as elements of a Σ–algebra) satisfy the behavioural rules V1–V5.

As regards point *ii*), we have to show that any (nonempty) computation of $L(p)$ is the interpretation of some action of p, and similarly for its residual. Again, this may be shown by structural induction on p.

□

This result gives more consistency to our "structural" pomset semantics for terms. It shows that our notion of *composite action* for terms has a natural model–theoretic counterpart.

As a consequence, we may deduce that the corresponding bisimulation equivalences coincide:

Corollary 6.10 (*Compatibility of* $\asymp$ *and* $\asymp_{\mathcal{L}}$)

$$\textit{For any } p, q \in \mathcal{P} : \quad p \asymp q \iff L(p) \asymp_{\mathcal{L}} L(q)$$

Proof: *Only if*: by induction on the sizes of p, q. We use point *i*) of theorem 6.9 and induction to show that: $\{(L(p), L(q))\}$ is a LES–bisimulation.

If: similarly, using point *ii*) of theorem 6.9.

□

We conclude here our discussion about the semantics of LES's. In the next section we shall return to our "syntactic" equivalence $\asymp$ and examine its algebraic properties.

6.3 Algebraic characterisation

The purpose of this section is to axiomatise the pomset equivalence $\asymp$ on T_Σ.

Since terms are interpreted as ordinary transition systems over a set of actions, we may adopt here the general method established by Hennessy and Milner in [HM 85].

Essentially, this method consists in transforming terms into *sumforms* $\sum_{i\in I} a_i \bullet p_i$, where the a_i's are actions of a given set of actions A. Such sumforms are just a *notation* for finite (acyclic) transition systems labelled by actions of A: the a_i's are the initial actions of the system, and the p_i's the corresponding residuals.

Now it is a standard result (see again [HM 85]) that bisimulation equivalence on sumforms reduces to equality modulo axioms A1–A4.

Thus the required axiom system will consist of laws A1–A4, plus a set of laws effecting the translation of terms into sumforms (*normalisation*). In fact the main difficulty is in finding the normalisation laws.

Let us now formalise this method in our setting. We first enrich our syntax by allowing terms of the form $u \bullet p$, for $u \in Act$, with behaviour given by:

V6. $\quad u \bullet p \xrightarrow{u} p$

Let $\Sigma_\bullet$ denote the new syntax. We now define sumforms as follows:

Definition 6.11 *A sumform (sf) is a term of $T_{\Sigma_\bullet}$ - defined modulo axioms A1, A2, A3 - of the form:*

$$\hat{p} \;=\; \sum_{i\in I} u_i \bullet p_i$$

where for each $i \in I$, $u_i \in Act$ and p_i is a sumform. By convention $\hat{p} = \mathrm{NIL}$ if $I = \emptyset$.

Note that, due to the composite nature of our actions, sumforms are not quite sufficient to equate terms syntactically. Since the bisimulation $\asymp$ actually uses *equivalence classes* of actions (up to axioms P1, P2), we would need here, besides sumforms for terms, also some kind of equational theory for actions.

Roughly speaking, to carry out syntactical comparisons on terms, we must proceed as follows:

(i) Expand terms into sumforms, by means of the normalisation axioms.

(ii) Rearrange the actions in sumforms as required by phase (iii), using axioms P1, P2.

(iii) Compare the sumforms thus obtained, using the laws A1–A4.

In fact, to be completely rigourous, we should generalise our algebra of terms to a heterogeneous algebra, including a new carrier of actions. Then our axiom system would be made out of laws A1–A4, the normalisation axioms, and the additional law:

$$u \equiv v \implies u \bullet p = v \bullet p$$

where $\equiv$ is the congruence generated by P1, P2 on Act (see page 148).

However we do not really want to bother about such details here. We shall therefore present a unique set of axioms $\mathcal{H}$ including A1–A4, P1, P2, where it is intended that the only axioms that apply to actions are P1, P2. In fact these two axioms, although obviously valid for general processes, are only necessary to equate actions.

In order to normalise terms we need, besides the laws P3 and A1–A3 encountered already, two specific axioms H1 and H2. The new axioms are given in figure 6.1, together with the rest of the laws $\mathcal{H}$.

We have now the following:

Lemma 6.12 *Normalisation lemma:*

For any $p \in T_\Sigma$ *there exists a sumform* $\hat{p}$ *such that* $p =_{\mathcal{H}} \hat{p}$.

Proof: The proof of this lemma, by induction on the size of p, makes use of all axioms in $\mathcal{E}$ except for the axioms P1, P2 and the idempotence law A4. Axioms A1, A2, A3 are used implicitly throughout. As usual, we proceed by case analysis.

i) NIL is a normal form.

ii) $p = a:q$. If $\hat{q} = \mathrm{NIL}$, we let $\hat{p} = (a:\mathrm{NIL}) \bullet \mathrm{NIL}$. Then, using the laws H1, A3 and induction, we get:

$$p =_{\mathcal{H}} a:\hat{q} =_{\mathcal{H}} (a:\mathrm{NIL}) \bullet \mathrm{NIL} + \mathrm{NIL} =_{\mathcal{H}} \hat{p}$$

If $\hat{q} = \sum_{i\in I} u_i \bullet q_i \neq \mathrm{NIL}$, let: $\hat{p} = (a:\mathrm{NIL}) \bullet \hat{q} + \sum_{i\in J} (a:u_i) \bullet q_i$. We then obtain, using axiom H1 and induction:

$$p =_{\mathcal{H}} a:\hat{q} = a:(\sum_{i\in I} u_i \bullet q_i) =_{\mathcal{H}} (a:\mathrm{NIL}) \bullet \hat{q} + \sum_{i\in I} (a:u_i) \bullet q_i = \hat{p}$$

iii) $p = q \mid r$. If $\hat{r} = \mathrm{NIL}$, we set $\hat{p} = \hat{q}$. Then we may use induction, as well as axioms H2, A3 and P3, to get:

$$p =_{\mathcal{H}} \hat{q} \mid \hat{r} = (\sum_{i\in I} u_i \bullet q_i) \mid \mathrm{NIL} =_{\mathcal{H}} \sum_{i\in I} u_i \bullet (q_i \mid \mathrm{NIL}) + \mathrm{NIL} + \mathrm{NIL} =_{\mathcal{H}} \hat{p}$$

Otherwise, if both $\hat{q} = \sum_{i\in I} u_i \bullet q_i \neq \mathrm{NIL}$ and $\hat{r} = \sum_{j\in J} v_j \bullet r_j \neq \mathrm{NIL}$, let: $\hat{p} = \sum_{i\in I} u_i \bullet (q_i \mid \hat{r}) + \sum_{i\in I, j\in J} (u_i \mid v_j) \bullet (q_i \mid r_j) + \sum_{j\in J} v_j \bullet (\hat{q} \mid r_j)$. We may then deduce:

$$p =_{\mathcal{H}} \hat{q} \mid \hat{r} = (\sum_{i\in I} u_i \bullet q_i) \mid (\sum_{j\in J} v_i \bullet r_j) =_{\mathcal{H}} \hat{p}$$

using induction and axiom H2.

iv) $p = q + r$. If $\hat{q} = \mathrm{NIL}$, set $\hat{p} = \hat{r}$ and use A3. Similarly if $\hat{r} = \mathrm{NIL}$. Otherwise define $\hat{p} = \hat{q} + \hat{r}$. We have then $p =_{\mathcal{H}} \hat{p}$ by simple induction.

This concludes the proof of our lemma. □

We may now state our characterisation result:

Theorem 6.13 *The equivalence $\asymp$ is the $\Sigma_\bullet$-congruence $=_{\mathcal{H}}$ generated by the axioms $\mathcal{H}$.*

Proof. The soundness of the axioms is easy to show. The proof of completeness is at this point completely standard. □

Axioms $\mathcal{H}$

$$(A1) \quad x + (y + z) = (x + y) + z$$

$$(A2) \quad x + y = y + x$$

$$(A3) \quad x + \mathrm{NIL} = x$$

$$(A4) \quad x + x = x$$

$$(P1) \quad x \mid (y \mid z) = (x \mid y) \mid z$$

$$(P2) \quad x \mid y = y \mid x$$

$$(P3) \quad x \mid \mathrm{NIL} = x$$

$$(H1) \quad a : (\textstyle\sum_{i \in I} u_i \bullet p_i) = (a : \mathrm{NIL}) \bullet (\sum_{i \in I} u_i \bullet p_i) + \sum_{i \in I} (a : u_i) \bullet p_i$$

$$(H2) \quad (\textstyle\sum_{i \in I} u_i \bullet p_i) \mid (\sum_{j \in J} v_i \bullet q_j) = \sum_{i \in I} u_i \bullet (p_i \mid \sum_{j \in J} v_j \bullet q_j) + \sum_{i \in I, j \in J} (u_i \mid v_j) \bullet (p_i \mid q_j) + \sum_{j \in J} v_j \bullet (\sum_{i \in I} u_i \bullet p_i \mid q_j)$$

Figure 6.1: Axiomatisation of $\asymp$.

6.4 Comparison with other equivalences

We want now to relate our new equivalence $\asymp$ to Milner's strong bisimulation $\sim$ and to the distributed bisimulation $\sim_d$ of chapter 4.

Let us start with the easiest task, the comparison with $\sim$. Note that all CCS transitions, whose label is an atomic action, are also transitions in our calculus. Moreover, since the equality $\equiv$ on *Act* reduces to identity when actions are atomic, the bisimulation $\asymp$ uses atomic transitions exactly in the same way as does $\sim$.

We thus have the implication: $p \asymp q \implies p \sim q$. On the other hand we saw examples in the preceding section, and notably the interleaving one, where $p \sim q$ but $p \not\asymp q$. We conclude that $\asymp \subset \sim$.

The relation between $\asymp$ and the distributed equivalence $\sim_d$ is less evident to work out. To settle the question raised at the end of the previous section, we immediately give a counterexample to $\asymp \subseteq \sim_d$.

Example 8) $\quad a \mid (b + c) \quad \subsetneq \quad a{:}\,(b + c) \;+\; (a \mid b) \;+\; (a \mid c)$

One may easily check that any transition of the left member is matched by some transition in the right member.

Now the same absorption is not valid for $\sim_d$, since the left member has a distributed transition $\xrightarrow{a} <\mathrm{NIL}, b + c>$ that the right member cannot produce.

We will show on the other hand that $\sim_d \subseteq \asymp$ (and thus $\sim_d \subset \asymp$). To do this, we prove that $\asymp$ satisfies the axioms $\mathcal{D}$ (given in figure 4.1 of chapter 4) which characterise $\sim_d$.

This involves introducing the new operator $\between$ in our calculus, and extending correspondingly our equivalence $\asymp$.

Let then Σ_1 denote the enlarged syntax:

$$\Sigma_1 = A \cup \{\mathrm{NIL}, +, \mid, \between\}$$

and $\mathcal{P}_1$ be the set of processes on Σ_1.

The operator $\between$ is specified by the two rules:

V7. $p \xrightarrow{u} p'$ implies $p \between q \xrightarrow{u} p' \mid q$

V8. $p \xrightarrow{u} p'$, $q \xrightarrow{v} q'$ imply $p \between q \xrightarrow{u|v} p' \mid q'$

We prove next that the axioms $\mathcal{D}$ are *sound* for $\asymp$ (extended to the new calculus).

Theorem 6.14 *If* $p, q \in \mathcal{P}_1$, *then* $p =_{\mathcal{D}} q$ *implies* $p \asymp q$.

Proof: Since $\asymp$ is a congruence, it is enough to show that $p \asymp q$ for any instance $p = q$ of an axiom of $\mathcal{D}$.

Now A1–A4 are already axioms of $\mathcal{H}$. Thus the only laws we have to consider are CPE and LP1–LP4. For each of these laws, we will produce a c–bisimulation that contains the pair (p, q) (if $p = q$ is an instance of the law).

Let $p = q$ be an instance of CPE. Then the required c–bisimulation is:

$$R_0 = \{(r \mid s,\ r \between s + s \between r),\ (r' \mid s',\ s' \mid r') \mid r, s, r', s' \in \mathcal{P}_1\} \cup Id$$

Let now $p = q$ be an instance of LPi, for $i \in \{1, 2, 3, 4\}$. Then it is easy to check that the relation R_i, where:

$$R_1 = \{((r + s) \between t,\ r \between t + s \between t) \mid r, s, t \in \mathcal{P}_1\} \cup Id$$

$$R_2 = \{((r \between s) \between t,\ r \between (s \mid t)) \mid r, s, t \in \mathcal{P}_1\} \cup Id$$

$$R_3 = \{(r \between \mathrm{NIL},\ r),\ (r \mid \mathrm{NIL},\ r) \mid r \in \mathcal{P}_1\}$$

$$R_4 = \{(\mathrm{NIL} \between r,\ r) \mid r \in \mathcal{P}_1\}$$

is a c-bisimulation containing (p, q). □

We may thus conclude that our new equivalence is weaker than the distributed equivalence of chapter 4. Intuitively – looking back at example 8) – this is because distributed equivalence captures the dependence between actions and *choices* (the action a and the choice $(b + c)$ in the example), whereas pomset equivalence does not have this capacity.

Chapter 7

Conclusions

This work has been concerned with behavioural aspects of concurrent systems. I hope it might provide some contribution to the search for execution models that account for the causal and parallel structure of processes. In recent years there has been a convergence of interest upon non interleaved models for concurrency [AD 86, BC 86, DDM 85, GR 83, Gra 81, NT 84, Rei 85, GV 87, Win 82, Win 87] .

Our different semantics have been applied, throughout the thesis, to an elementary subset of pure CCS. We discuss here some possible extensions of our approaches to a more comprehensive language (or to different formalisms). We point to a few technical problems and suggest connections with some recent work in the field, which might be worth investigating.

We proceed in a rather schematic way, dwelling on the relevant chapters.

Chapter 3. We did not develop the study of the abstraction equivalence $\sim_{abs}$ for LES's as far as we did for NDP's in chapter 2, mainly because we found a mismatch with the operational description (given by the distributed transitions of chapter 4). We could have adjusted our definition of abstraction homomorphism so as to obtain a single–valued d–bisimulation. However the only way we could devise for doing this was to introduce a recursive clause in the definition of homomorphism (to replace clause iv)), which would have taken off much of the attractiveness of the morphism itself.

Chapter 4. The next step would be to extend the distributed bisimulation $\sim_d$ to finitary CCS with restriction and renaming. However we encounter an obvious problem here, since our distributed transitions (at least in the original

formulation of our semantics) split the potentially communicating components of a process. Thus for example we cannot adopt the rule:

$$p \xrightarrow{b} <p', p''> \implies p\backslash a \xrightarrow{b} <p'\backslash a,\ p''\backslash a>$$

since this would preclude any communication on a between $p'\backslash a$ and $p''\backslash a$.

A similar problem arises with renaming. As a matter of fact, the second formulation of our semantics – where transitions yield a local and a global residual – would be probably better suited for defining these operators. However, before adopting this formulation for further study, we would need to understand it better in our present calculus (with communication), where we still lack an algebraic characterisation for the corresponding equivalence ($\cong_d$).

In fact it may be that operators like CCS *restriction* are tailored towards a sequential interpretation of parallelism. Although some kind of encapsulation operator is essential in a language, to allow for different levels of description, it might be that a completely new operator is needed to deal with true concurrency.

Concerning the relation of our distributed semantics to other semantics, we noted already its similarity with Montanari & Degano semantics by *histories* (at least in its later formulation given in [DDM 85]). However a study of the exact relation between the two semantics still needs to be done.

We would like to suggest also a possible correlation with the *generalised pomset bisimulation* proposed in [GV 87] for Petri nets (which may be easily transferred to labelled event structures). Essentially, [GV 87] require that for any equivalent computations of two LES's, also their decompositions in smaller computations be equivalent. This allows them to express the dependency between actions and *choices*, which seems to be also a distinguishing capacity of our semantics. When considered on the characteristic examples of chapters 4 and 6, their equivalence seems to coincide, on event structures, with our distributed equivalence on the corresponding terms. Whether the two semantics actually agree is certainly an interesting question to investigate.

Chapter 5. It would be nice here to have a generalisation of our simplification result to the calculus with communication. This would enable us to continue the study of $\approx_d$ using the second formulation of our semantics, when this is preferable.

Chapter 6. Our pomset semantics is applied to a very simple language, with no communication and no τ–transitions. In fact this semantics was originally elaborated (in [CB 86]) for a more general language, where prefixing is replaced by general sequentialisation. Moreover, we give in [CB 86] a characterisation of the class of event structures needed to model the language, thus generalising a result of Gischer [Gi 84] .

An extension of pomset semantics to the entire pure CCS is delineated in [CB 87b] . There it is shown that both distributed and pomset transitions may be derived from a more intensional semantics, where transitions are labelled with *proof* terms, and sequences of transitions are considered up to a permutation of concurrent transitions.

As is to be expected, the introduction of communication (in the standard pattern of CCS) complicates considerably the interpretation of terms as event structures. Event structures with an ordering relation of causality are not sufficient here. We therefore adopt a variant of one of Winskel's later definitions for LES's, where the causality relation is replaced by an enabling relation.

We do not have an axiomatisation in this general setting, (nor does it seem easy to derive one). Also, we lack here a characterisation of the class of event structures that model the language.

Our pomset semantics for LES's has been extended to Petri nets by van Glabbeek & Vaandrager in [GV 87] . It is suggested by U. Goltz that for some class of Petri nets the definition of this semantics could be simplified by considering computations of bounded length.

Having a description of concurrent processes as pomset transition systems (or even step transition systems), one may use it as a basis to define a more refined version of different notions of equivalence. In [ADF 86] pomset transitions are used to define *testing* equivalence on event structures, while [TV 87] presents a *failure* semantics for CSP processes interpreted as step transition systems.

CCS semantics on the syntax: $\Sigma' = A_\tau \cup \{\text{NIL}, +, |\}$.

i) $a : t \xrightarrow{a} t$

ii) $t \xrightarrow{a} t'$ implies $(t+s) \xrightarrow{a} t'$, $(s+t) \xrightarrow{a} t'$

iii) $t \xrightarrow{a} t'$ implies $(t\,|\,s) \xrightarrow{a} (t'\,|\,s)$, $(s\,|\,t) \xrightarrow{a} (s\,|\,t')$

iv) $t \xrightarrow{a} t'$, $s \xrightarrow{\bar{a}} s'$ imply $(t\,|\,s) \xrightarrow{\tau} (t'\,|\,s')$

Axioms

	$A1.$	$x + x' = x' + x$
sum − laws	$A2.$	$x + (x' + x'') = (x + x') + x''$
	$A3.$	$x + \text{NIL} = x$
absorption law	$A4.$	$x + x = x$
	$P1.$	$x\,|\,x' = x'\,|\,x$
par − laws	$P2.$	$x\,|\,(x'\,|\,x'') = (x\,|\,x')\,|\,x''$
	$P3.$	$x\,|\,\text{NIL} = x$
interleaving	IN.	If $x = \sum_{i\in I} a_i x_i$, $y = \sum_{j\in J} b_j y_j$, *then*

$$x\,|\,y = \sum_{i\in I} a_i\,(x_i\,|\,y) + \sum_{j\in J} b_j\,(x\,|\,y_j) + \sum_{a_i = \bar{b_j}} \tau\,(x_i\,|\,y_j)$$

Bibliography

[AB 84] D. AUSTRY, G. BOUDOL. *Algèbre de processus et synchronisation*, J. Theoretical Computer Science 30, 1984.

[ADF 86] L. ACETO, R. DE NICOLA, A. FANTECHI. *Testing Equivalences for Event Structures*, Nota interna B4–63, Istituto di Elaborazione dell'Informazione, CNR, Pisa, 1986.

[BK 84] J. BERGSTRA, J. KLOP. *Algebra of Communicating Processes*, Proc. CWI Symposium on Mathematics & Comp. Sci., Centrum voor Wiskunde en Informatica, Amsterdam 1984.

[BK 85] J. BERGSTRA, J. KLOP. *Algebra of Communicating Processes with abstraction*, J. Theoretical Computer Science 37(1), 1985.

[Bou 85] G. BOUDOL. *Notes on Algebraic Calculi of Processes*, in Logics and Models of Concurrent Systems, NATO ASI Series F13, Springer Verlag, 1985.

[BC 86] G. BOUDOL, I. CASTELLANI *On the semantics of concurrency: partial orders and transition systems*, Proc. TAPSOFT 87 (Pisa), LNCS 249, Springer Verlag 1987.

[BC 87a] G. BOUDOL, I. CASTELLANI. *Concurrency and Atomicity*, full version of the previous paper, selected for a special issue of the J. Theoretical Computer Science.

[BC 87b] G. BOUDOL, I. CASTELLANI. *Permutation of transitions: An Event Structure Semantics for CCS and SCCS*, to appear as an INRIA Research Report.

[BRS 85] G. BOUDOL, G. ROUCAIROL, R. DE SIMONE. *Petri Nets and Algebraic Calculi of Processes*, in Proc Advances in Petri Nets, 1985, LNCS 222, 1986.

[BR 83] S. BROOKES, C. ROUNDS. *Behavioural Equivalence Relations induced by Program Logics*, in Proc. ICALP '83, LNCS 154, 1983.

[Ca 85] I. CASTELLANI. *Bisimulations and Abstraction Homomorphisms*, Proc. TAPSOFT Conference, Berlin 1985, LNCS 185, 1985.
J. Computer and System Sciences 34, 1987.

[CFM 82] I. CASTELLANI, P. FRANCESCHI, U. MONTANARI. *Labelled Event Structures: A Model for Observable Concurrency*, in Proc. IFIP TC2 Working Conference on Formal Description of Programming Concepts II, Garmisch, 1982.

[CH 85] I. CASTELLANI, M. HENNESSY. *Distributed Bisimulations*, unpublished research note, Edinburgh, 1985.

[Dij 68] E. W. DIJKSTRA. *Cooperating Sequential Processes*, in Programming Languages (F. Genuys, Ed.), 1968.

[DM 85] P. DEGANO, U. MONTANARI. *Distributed Systems, Partial Orderings of Events and Event Structures*, in Control Flow and Data Flow: Concepts of Distributed Programming (M. Broy, Ed.), NATO ASI Series F14, 1985.

[DDM 85] P. DEGANO, R. DE NICOLA, U. MONTANARI. *Partial Ordering Derivations for CCS*, in Proc. FCT 85, LNCS 199, 1985.

[DDM 87] P. DEGANO, R. DE NICOLA, U. MONTANARI. *A Partial Ordering Semantics for CCS*, draft.

[DNi 84] R. DE NICOLA. *Extensional Equivalences for Transition Systems*, Acta Informatica 24, 1987.

[DH 84] R. DE NICOLA, M. HENNESSY. *Testing equivalences for processes*, J. Theoretical Computer Science 34, 1984.

[DSi 85] R. DE SIMONE. *Higher level syncronizing devices in MEIJE-SCCS*, J. Theoretical Computer Science 37, 1985.

[Gis 84] J. L. GISCHER. *Partial Orders and the Axiomatic Theory of Shuffle*, Ph. D. Thesis, Stanford University, 1984.

[GLT 80] H. J. GENRICH, K. LAUTENBACH, P. S. THIAGARAJAN. *Elements of General Net Theory*, in Net Theory and Applications (W. Brauer, Ed.) LNCS 84, 1980.

[Gol 86] U. GOLTZ. *Building Structured Petri Nets*, GMD Research report 223, 1986.

[GM 84] U. GOLTZ, A. MYCROFT. *On the relationship of CCS and Petri Nets*, in Proc. ICALP 84, LNCS 179, 1984.

[GR 83] U. GOLTZ, W. REISIG. *The Non-sequential Behaviour of Petri Nets*, Information and Control 57, 1983.

[Gra 81] J. GRABOWSKI. *On Partial Languages*, Fundamenta Informaticae IV.2, 1981.

[GV 87] R. VAN GLABBEEK, F. VAANDRAGER. *Petri Net Models for Algebraic Theories of Concurrency*, Proc. PARLE Conference, Eindhoven, 1987, to appear in LNCS.

[Hen 80] M. HENNESSY. *Time and Interleaving*, 1980. To be published.

[HM 85] M. HENNESSY, R. MILNER. *Algebraic laws for Nondeterminism and Concurrency* , J. ACM 32, 1985.

[HP 80] M. HENNESSY, G. PLOTKIN. *A Term Model for CCS*, LNCS 88, 1980.

[Kel 76] R. KELLER. *Formal verification of Parallel Programs*, Communications of the ACM n. 19, Vol. 7, 1976.

[LMV 87] V. LECOMPTE, E. MADELAINE, D. VERGAMINI *Un système de vérification de processus parallèles et communicants*, INRIA research report 83, 1987.

[Maz 84] A. MAZURKIEWICZ. *Traces, Histories, Graphs: Instances of a Process Monoid*, MFCS 84, LNCS 176, 1984.

[Mil 79] R. MILNER. *Flowgraphs and Flowalgebras*, J. ACM 26, 1979.

[Mil 80] R. MILNER. *A Calculus of Communicating Systems*, LNCS 92, 1980.

[Mil 81] R. MILNER. *On relating Synchrony and Asynchrony*, CSR 75-80, Edinburgh, 1981.

[Mil 83] R. MILNER. *Calculi for Synchrony and Asynchrony*, J. Theoretical Computer Science, Vol. 25, 1983.

[Mil 84] R. MILNER. *Lectures on a Calculus for Communicating Systems*, in Proc. Marktoberdorf Summerschool, 1984. NATO ASI Series F Vol. 14, Springer Verlag, 1985.

[MM 79] G. MILNE, R. MILNER. *Concurrent processes and their syntax*, J. ACM 26(2), 1979.

[NPW 81] M. NIELSEN, G. PLOTKIN, G. WINSKEL. *Petri Nets, Event Structures and Domains* (Part 1) J. Theoretical Computer Science, Vol. 13, 1981.

[NT 84] M. NIELSEN, P.S. THIAGARAJAN. *Degrees of Nondeterminism and Concurrency: a Petri Net view*, in LNCS 181, 1984.

[Parikh 81] R. PARIKH. *Propositional Dynamic Logics of Programs: a Survey*, in "Logics of Programs", LNCS 125, 1981.

[Park 81] D. PARK. *Concurrency and Automata on Infinite Sequences*, in LNCS 104, 1981.

[Petri 62] C. A. PETRI. *Kommunication mit Automaten*, Schriften des Institutes für Instrumentelle Mathematik, Bonn, 1962.

[Petri 77] C. A. PETRI. *Nonsequential processes*, GMD Research report 77-05, 1977.

[Plo 81] G. PLOTKIN. *A Structured Approach to Operational Semantics*, DAIMI FN-19, Computer Science Dept, Aarhus University.

[Pnu 85] A. PNUELI. *Linear and Branching Structures in the Semantics and Logics of Reactive Systems*, in Proc ICALP 85, LNCS 194, 1985.

[Pra 82] V. R. PRATT. *On the Composition of Processes*, in Proc. 9th POPL, 1982.

[Pra 85] V. R. PRATT. *The Pomset Model of Parallel Processes: Unifying the Temporal and the Spatial*, Seminar on Concurrency, LNCS 197, 1985.

[Rei 85] W. REISIG. *On the Semantics of Petri Nets*, in Formal Models in Programming, North Holland, 1985.

[Shi 82] M. SHIELDS. Non–sequential behaviours (1 and 2). Technical Reports CSR 119-82 and CSR 144-83, University of Edinburgh, 1982–83.

[Sti 85] C. STIRLING. *A proof theoretic characterisation of observational equivalence*, J. Theoret. Comput. Sci. 39, 1985.

[Wi 80] J. WINKOWSKI. *Behaviours of Concurrent Systems*, J. Theoret. Comput. Sci. 12 , 1980.

[Win 80] G. WINSKEL. *Events in Computation*, Ph. D. Thesis, Edinburgh University, 1980.

[Win 82] G. WINSKEL. *Event Structure Semantics for CCS and Related Languages*, in Proc. ICALP 82, LNCS 140, 1982.

[Win 87] G. WINSKEL. *Event Structures*, LNCS 255, 1987.