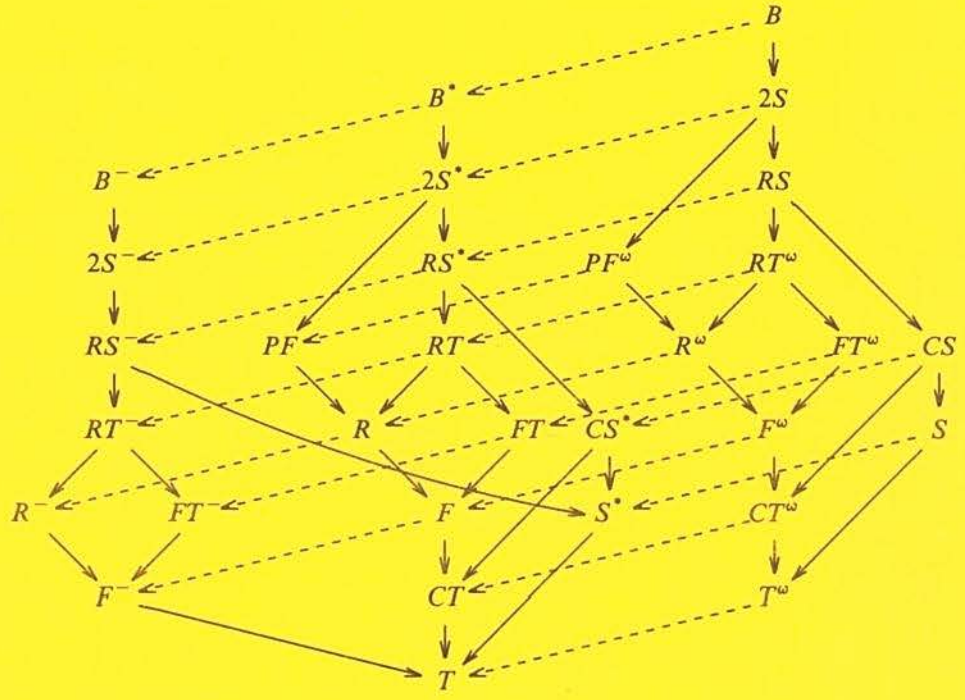
L4

Comparative Concurrency Semantics and Refinement of Actions



R.J. van Glabbeek

VRIJE UNIVERSITEIT TE AMSTERDAM

Comparative Concurrency Semantics and Refinement of Actions

ACADEMISCH PROEFSCHRIFT

ter verkrijging van de graad van doctor aan
de Vrije Universiteit te Amsterdam,
op gezag van de rector magnificus
dr. C. Datema,
hoogleraar aan de faculteit der letteren,
in het openbaar te verdedigen
ten overstaan van de promotiecommissie
van de faculteit der wiskunde en informatica
op woensdag 16 mei 1990 te 15.30 uur
in het hoofdgebouw van de universiteit, De Boelelaan 1105

door
Robert Jan van Glabbeek
geboren te Eindhoven

Centrum voor Wiskunde en Informatica
1990

Promotoren:	prof. dr. J.W. Klop
	prof. dr. J.A. Bergstra
Referent:	prof. dr. G. Winskel

Contents

Affiliations

Most of the work reported in this thesis was done when I was employed at the Centre for Mathematics and Computer Science, P.O. Box 4079, 1009 AB Amsterdam. The manuscript was finalized during my employment at the Technical University of Munich, Postfach 202420, D-8000 München 2. Chapter II is joint work with Frits Vaandrager, and Chapter III with Peter Weijland, both affiliated with the Centre for Mathematics and Computer Science. Chapters IV, V and VI are joint work with Ursula Goltz, Gesellschaft für Mathematik und Datenverarbeitung, Postfach 1240, D-5205 Sankt Augustin 1.

Acknowledgements

Jan Willem Klop has been an ideal promotor for me. He was always ready to discuss my work, but let me free to pursue my own goals in my own way. It has been a great pleasure to have him as a supervisor.

I also like to express my gratitude to Jan Bergstra, not only for being my second promotor, but also for initiating my employment at CWI and organizing a very interesting scientific environment. Moreover I appreciated the many discussions with him, in which he was always full of ideas.

I greatly benefited from the collaboration with Frits Vaandrager. With him I could discuss all my work in depth and he contributed numerous useful ideas.

I thank Ursula Goltz for our fruitful cooperation since 1987. Our joint work on action refinement forms an important constituent of this thesis. This work was initiated by a discussion with Albert Meyer and Ernst-Rüdiger Olderog at ICALP 87 in Karlsruhe, to whom I also express my gratitude.

I am grateful to my colleagues Jos Baeten and Jan Friso Groote for our pleasant collaboration and for their useful comments on earlier versions of parts of this thesis.

Furthermore I gratefully acknowledge the discussions and correspondence with Alex Rabinovich, Walter Vogler, Wolfgang Reisig, Vaughan Pratt, Tony Hoare, Ilaria Castellani, Rocco De Nicola, Pierpaolo Degano, Ugo Montanari and Gérard Boudol that contributed to the work reported in this thesis.

I thank the participants of the weekly Process Algebra Meetings (as far as not mentioned already): Wiet Bouma, Jeroen Bruijning, Nicolien Drost, Henk Goeman, Karst Koymans, Sjouke Mauw, Kees Middelburg, Hans Mulder, Alban Ponse, Gerard Renardel de Lavalette, Piet Rodenburg, Gert Veltink, Jos Vrancken, Fer-Jan de Vries, Peter Weijland, Freek Wiedijk and Han Zuidweg, for numerous lively discussions and comments.

More in general, my years at the CWI Department of Software Technology, headed by Jaco the Bakker, will remain a pleasant memory for its excellent working conditions and friendly atmosphere.

It is a pleasure to thank Glynn Winskel for his kind willingness to referee this thesis and to take part in the examination committee.

Thanks are also due to the printing department of CWI for its fast production of this work.

Finally, special thanks goes to Gertrud Jacobs for her careful preparation of Chapters IV, V and VI of the manuscript, and for being available whenever she was needed.

ERRATUM. All reference numbers in the introduction and in Chapters I - VI, starting from 57, should be decremented by 1.

Introduction

1. Comparative concurrency semantics. This thesis is about comparative concurrency semantics.

Concurrency is the study of concurrent systems. Often concurrency as area of scientific research is located in computer science. In that case the systems which are the subject of study are taken to be computers or computer programs. However, much theory in the field of concurrency applies equally well to other systems, like machines, elementary particles, protocols, networks of falling dominoes or human beings. *Concurrent* or *parallel* systems - as opposed to *sequential* systems - are systems capable of performing different activities at the same time.

Semantics is the study of the meaning of words. In concurrency, one often employs formal languages for the description of concurrent systems. These I call *system description languages*. Like all formal languages, system description languages are usually introduced to avoid the ambiguities of natural languages and to gain accuracy of expression. Therefore their semantics tends to be easier than the semantics of natural languages. Moreover the meaning of the words in a formal language should to some extent be given by the one who defines the language, rather than to be discovered by linguists.

Since system description languages tend to describe abstractions of systems rather than concrete systems, the meaning of an expression in a system description language is in general given by an *equivalence class* of systems (i.e. a class of systems which are considered to be equivalent on a chosen level of abstraction). Thus the meaning of the entire language is determined by a partition of a set of systems into equivalence classes and an allocation of one such equivalence class to each expression. For this reason it is convenient to divide the semantics of system description languages into two subfields, namely the study of equivalence relations on sets of concurrent systems, and the study of allocating equivalence classes to expressions in particular languages. The first field deals with the establishment of criteria, determining when two systems are sufficiently alike to be collected in the same equivalence class. It can be studied independently of a particular system description language. Therefore it can be simply referred to as *semantics of concurrent systems* or *concurrency semantics* for short.

In concurrency semantics a criterion, determining when two systems are sufficiently alike to be collected in the same equivalence class, is called *a semantics*, and the induced equivalence relation a *semantic equivalence*. In the literature on concurrency semantics many semantics have been proposed and most likely also a multitude of sensible semantics have never been proposed. The classification of these semantics is called *comparative concurrency semantics* and will be the primary subject of this thesis.

2. Design and verification. Much work in concurrency is motivated by an interest in design problems for concurrent systems. A fruitful method to design concurrent systems is by means of *stepwise refinement*. Here one starts with a description S_0 of the system one has in mind. This initial description is called a *specification* of the desired system. It abstracts from all the details of the desired system that are not essential in its behaviour and leaves open many design decisions that have to be taken later on. Then one starts refining the specification by adding step for step the details one needs to know when the system is going to be built. In this way one obtains a sequence

$$S_0 \rightarrow S_1 \rightarrow \cdots \rightarrow S_n$$

of system descriptions of which the last one says exactly how the system will look like. This final state in the design process describes the *implementation* of the desired system.

Roughly one can distinguish two different kinds of refinement steps in such a sequence of system descriptions. First of all there are steps in which information is added about *what* the system ought to do. These steps concern the goal of the entire exercise and can therefore not be proven correct in terms of this goal. Secondly there are steps that add information about *how* the system is going to do it. It is one of the tasks of concurrency theory to prove the correctness of such steps.

When considering only one step from a stepwise refinement sequence, the left-hand side of this step is called *specification* and the right-hand side *implementation*. Let $S \rightarrow I$ be a 'how'-step. The question is now to find criteria for determining whether or not this step is correct. Here at least two situations can be distinguished:

1. Although I describes much more activities of the desired system than S, all these extra activities can be considered as internal actions in which the user of the system is not interested. After abstraction from all these details, I and S are equivalent according to some suitable semantic equivalence.

2. Some choices about how the final system should behave, that were left open in S, are resolved in I. Therefore I and S cannot be equivalent. Here one needs a partial order between equivalence classes of concurrent systems, specifying when one system is a correct implementation of the other.

In order to tackle both cases one needs to define a suitable semantic equivalence and a partial order on the equivalence classes. Together these ingredients can be coded as a *preorder*, a reflexive and transitive relation, on system descriptions.

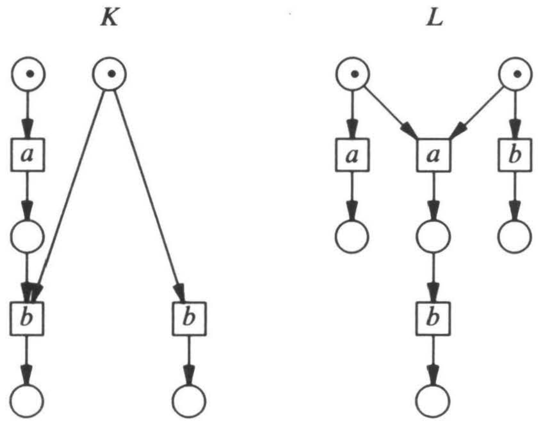
In this thesis, for reasons of convenience, attention is restricted to equivalences rather than arbitrary preorders. However, there exists a close correspondence between semantic equivalences and preorders. Most semantic equivalences are defined, or can be characterized, in terms of the properties that are shared by equivalent systems. For each system p, a set of properties $O(p)$ is defined, such that two systems p and q are equivalent iff $O(p) = O(q)$. Often $O(p)$ describes the observable behaviour of p according to some testing scenario. Now a corresponding preorder $\ll$ can be defined by $p \ll q$ iff $O(p) \subseteq O(q)$. Most preorders encountered in the literature on concurrency are of that form. I expect that using this insight, much work on classifying semantic equivalences can be generalized to preorders.

Above I argued that semantic equivalences (and preorders) can be relevant for the design of concurrent systems. However, in fact they are more often employed for *verification* purposes. In this case one is offered a specification and an implementation of a certain system and is asked to determine if the implementation is correct. In such applications the distance between the specification and the implementation tends to be larger than in one step in a design process. Therefore it is even more important to have solid criteria for deciding on the correctness of the implementation.

When semantic equivalences are used in the design of concurrent systems, or for verification purposes, they should be chosen in such a way that two system descriptions are considered equivalent only if the described behaviours share the properties that are essential in the context in which the system will be embedded. It depends on this context and on the interests of a particular user which properties are essential. Therefore it is not a task of concurrency semantics to find the 'true' semantic equivalence, but rather to determine which equivalence is suitable for which applications. It is the intention of this thesis to carry out a bit of this task. In particular it addresses the question which semantic equivalences are suitable for dealing with *action refinement*.

3. Refinement of actions. In this thesis concurrent systems are represented by expressions in a system description language or by elements of some mathematical model. The basic building block in the languages and models that occur in this thesis are the *actions* which may occur in a system. By an *action* here any activity is understood which is considered as a conceptual entity on a chosen level of abstraction. This allows design steps, in which actions are replaced by more complex system descriptions. Such a step in the design of a system is referred to as *refinement of actions*. Action refinement is a design step that adds information about what the system ought to do (a 'what'-step), at least if the refined actions are not considered to be internal. Therefore the 'correctness' of such refinement steps cannot be proven. However, the possibility of doing such steps puts some restrictions on the kind of equivalences that can be used for proving the correctness of 'how'-steps occurring in the same design process.

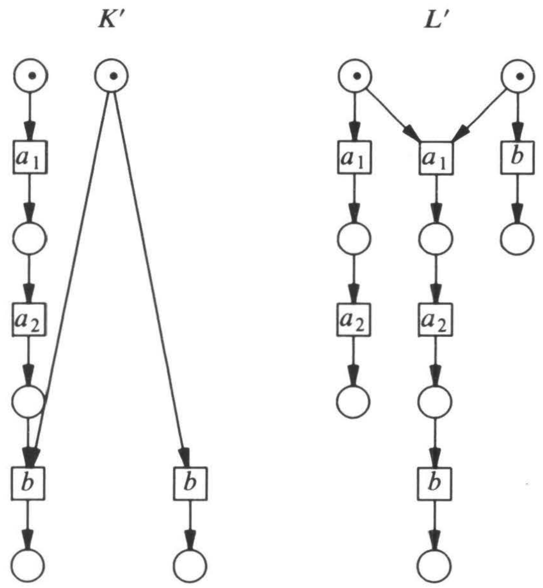
EXAMPLE: Consider the following specification of a concurrent system: 'The actions a and b should in principle be performed independently on different processors, but if one of the processors happens to be ready with a before the other starts with b, b may also be executed on this processor instead of the other one'. This system description is represented by the Petri net K below.



An introduction to Petri nets and the way they model concurrent systems can be found in REISIG [110].

Suppose that someone comes up with an implementation in which first it is determined whether the actions a and b will happen sequentially or independently, and subsequently one of these alternatives will take place, as represented by the Petri net L. Although this implementation does not seem very convincing, it will be considered 'correct' by many equivalences occurring in the literature.

Let the next step in the design process consist of refining the action a in the sequential composition of two actions a_1 and a_2. From L one thereby obtains the net L' on the right. If L' is going to be placed in an environment where a_2 becomes causally dependent on b - it may be the case that b is an output action, a_2 is an input action, and the environment needs data from b in order to compute the data that are requested by a_2 - then deadlock can occur. However, if the refinement step splitting a in a_1 and a_2 is carried out on K already, the resulting system K' is deadlock free in the environment sketched



above.

Thus the possibility of refining a somehow invalidates the correctness of the design step from K to L. □

A semantic equivalence is said to be *preserved under refinement of actions* if two equivalent processes remain equivalent after replacing all occurrences of an action a by a more complicated process $r(a)$. The example above indicates that for certain applications is may be fruitful to employ equivalences that are preserved under refinement of actions. It is one of the topics of this thesis to find out which equivalences have this property.

4. About the contents of this thesis. This thesis consists of seven chapters which are all based on separate papers and have their own introduction. This general introduction is only meant to give an indication of their contents and their role in the thesis.

In the first chapter several semantic equivalences for concrete sequential systems are presented, and motivated in terms of the observable behaviour of systems, according to some testing scenario. Here *concrete* means that no internal actions or internal choice is considered. These semantics are partially ordered by the relation 'makes strictly more identifications than', thus constituting a complete lattice. For ten of these semantic equivalences complete axiomatizations are provided. As in the rest of my thesis, stochastic and real-time aspects of concurrent systems are completely neglected. Furthermore the actions of which concurrent systems are considered to be composed, are left uninterpreted. Chapter I serves partly to give an overview of the literature on semantic equivalences for concrete sequential processes. The various notions that can be found elsewhere can easily be compared, since they are all presented in the same style, and using the same formalism. In order for the semantics of this chapter to be applicable for design and verification purposes, they have to be generalized to a setting with internal moves, and with parallelism. This can be done in many ways. In the last two chapters the two extreme points on the semantic lattice, trace semantics and bisimulation semantics, are generalized to a setting with parallelism and in Chapter III, bisimulation semantics is generalized to a setting with internal moves.

In the second chapter it is shown how semantic notions can be used in protocol verification and other applications. This chapter is entirely algebraic in style and employs axiom systems of which only classes of models are considered, rather than a particular model. It is based on the Algebra of Communicating Processes of BERGSTRA & KLOP [19,22]. In order to combine axiom systems representing semantic notions that are difficult to combine a new notion of 'proof' is developed.

The third chapter is devoted to the generalization of bisimulation equivalence to a setting with silent moves. It is argued that the solution of MILNER [92] (observation equivalence) does not respect the branching structure of processes and hence lacks an important feature of bisimulation semantics without internal moves. A finer equivalence is proposed which indeed respects branching structure. This new *branching bisimulation equivalence* turns out to have some practical advantages as well. In particular, we show that in a setting without parallelism it is preserved under refinement of actions, whereas observation equivalence is not.

In the fourth chapter an operator for refinement of actions is defined on four causality based models for concurrent systems, namely on three kinds of event structures and on Petri nets, and in the remaining three chapters it is investigated which of the 'linear time' and 'branching time' semantic equivalences proposed in the literature are preserved under refinement of actions and which are not. Chapter V can be regarded as an informal summary of the Chapters VI and VII. It uses Petri nets rather than event structures and contains no technicalities like definitions and proofs. Instead more attention has been paid to the examples.

All chapters in this thesis can be read independently, although for motivation it may be helpful to read the introduction to Chapter IV before Chapters V-VII, and depending on the taste of the reader it may be fruitful to consult Chapter V before or simultaneously with the last two chapters. Furthermore Chapter VI depends on Section 1 *or* 2 of Chapter IV. Conceptually Chapter VII follows Chapter VI, and it recalls its results.

Chapter I

The Linear Time - Branching Time Spectrum

R.J. van Glabbeek

In this chapter various semantics in the linear time - branching time spectrum are presented in a uniform, model-independent way. Restricted to the domain of finitely branching, concrete, sequential processes, only twelve of them turn out to be different, and most semantics found in the literature that can be defined uniformly in terms of action relations coincide with one of these twelve. Several testing scenarios, motivating these semantics, are presented, phrased in terms of 'button pushing experiments' on generative and reactive machines. Finally ten of these semantics are applied to a simple language for finite, concrete, sequential, nondeterministic processes, and for each of them a complete axiomatization is provided.

Table of Contents

Introduction

Process theory. A *process* is the behaviour of a system. The system can be a machine, an elementary particle, a communication protocol, a network of falling dominoes, a chess player, or any other system. Process theory is the study of processes. Two main activities of process theory are *modelling* and *verification.* Modelling is the activity of representing processes, mostly as elements of a mathematical domain or as expressions in a system description language. Verification is the activity of proving statements about processes, for instance that the actual behaviour of a system is equal to its intended behaviour. Of course, this is only possible if a criterion has been defined, determining whether or not two processes are equal, i.e. two systems behave similarly. Such a criterion constitutes the *semantics* of a process theory. (To be precise, it constitutes the semantics of the equality concept employed in a process theory.) Which aspects of the behaviour of a system are of importance to a certain user depends on the environment in which the system will be running, and on the interests of the particular user. Therefore it is not a task of process theory to find the 'true' semantics of processes, but rather to determine which process semantics is suitable for which applications.

Comparative concurrency semantics. This thesis aims at the classification of process semantics.[1] The set of possible process semantics can be partially ordered by the relation 'makes strictly more identifications on processes than', thereby becoming a complete lattice[2]. Now the classification of some useful process semantics can be facilitated by drawing parts of this lattice and locating the positions of some interesting process semantics, found in the literature. Furthermore the ideas involved in the construction of these semantics can be unraveled and combined in new compositions, thereby creating an abundance of new process semantics. These semantics will, by their intermediate positions in the semantic lattice, shed light on the differences and similarities of the established ones. Sometimes they also turn out to be interesting in their own right. Finally the semantic lattice serves as a map on which it can be indicated which semantics satisfy certain desirable properties, and are suited for a particular class of applications.

Most semantic notions encountered in contemporary process theory can be classified along four different lines, corresponding with four different kinds of identifications. First there is the dichotomy of linear time versus branching time: to what extent should one identify processes differing only in the branching structure of their execution paths? Secondly there is the dichotomy of interleaving semantics versus partial order semantics: to what extent should one identify processes differing only in the causal dependencies between their actions (while agreeing on the possible orders of execution)? Thirdly one encounters different treatments of abstraction from internal actions in a process: to what extent should one identify processes differing only in their internal or silent actions? And fourthly there are different approaches to infinity: to what extent should one identify processes differing only in their infinite behaviour? These considerations give rise to a four dimensional representation of the proposed semantic lattice.

However, at least three more dimensions can be distinguished. In this thesis, stochastic and real-time aspects of processes are completely neglected. Furthermore it deals with *uniform concurrency*[3] only. This means that processes are studied, performing actions[4] $a,b,c,...$ which are not subject to further investigations. So it remains unspecified if these actions are in fact assignments to variables or the falling of dominoes or other actions. If also the options are considered of modelling (to a certain degree) the stochastic and real-time aspects of processes and the operational behaviour of the elementary actions, three more parameters in the classification emerge.

[1] This field of research is called *comparative concurrency semantics*, a terminology first used by MEYER in [90].

[2] The supremum of a set of process semantics is the semantics identifying two processes whenever they are identified by every semantics in this set.

[3] The term uniform concurrency is employed by DE BAKKER et al [14].

[4] Strictly speaking processes do not perform actions, but systems do. However, for reasons of convenience, this thesis sometimes uses the word process, when actually referring to a system of which the process is the behaviour.

Process domains. In order to be able to reason about processes in a mathematical way, it is common practice to represent processes as elements of a mathematical domain. Such a domain is called a *process domain.* The relation between the domain and the world of real processes is mostly stated informally. The semantics of a process theory can be modelled as an equivalence on a process domain, called a *semantic equivalence.* In the literature one finds among others:

- *graph domains*, in which a process is represented as a *process graph*, or *state transition diagram*,
- *net domains*, in which a process is represented as a (labelled) *Petri net*,
- *event structure domains*, in which a process is represented as a (labelled) *event structure*,
- *explicit domains*, in which a process is represented as a mathematically coded set of its properties,
- *projective limit domains*, which are obtained as projective limits of series of finite term domains,
- and *term domains*, in which a process is represented as a term in a system description language.

Action relations. Write $p \xrightarrow{a} q$ if the process p can evolve into the process q, while performing the action a. The binary predicates $\xrightarrow{a}$ are called *action relations.* The semantic equivalences which are treated in this chapter will be defined entirely in terms of action relations. Hence these definitions apply to any process domain on which action relations are defined. Furthermore they will be defined *uniformly* in terms of action relations, meaning that all actions are treated in the same way. For reasons of convenience, even the usual distinction between internal and external actions is dropped in this chapter.

Finitely branching, concrete, sequential processes. Being a first step, this chapter limits itself to a very simple class of processes. First of all only *sequential* processes are investigated: processes capable of performing at most one action at a time. Moreover the main interest is in *finitely branching* processes: processes having in each state only finitely many possible ways to proceed. Finally, instead of dropping the usual distinction between internal and external actions, one can equivalently maintain to study *concrete* processes in which no internal actions occur (and also no internal choices as in CSP [76]). For this simple class of processes, when considering only semantic equivalences that can be defined uniformly in terms of action relations, the announced semantic lattice collapses in six out of seven dimensions and covers only the *linear time - branching time* spectrum.

Literature. In the literature on uniform concurrency 11 semantics can be found, which are uniformly definable in terms of action relations and different on the domain of finitely branching, sequential processes (see Figure 1).

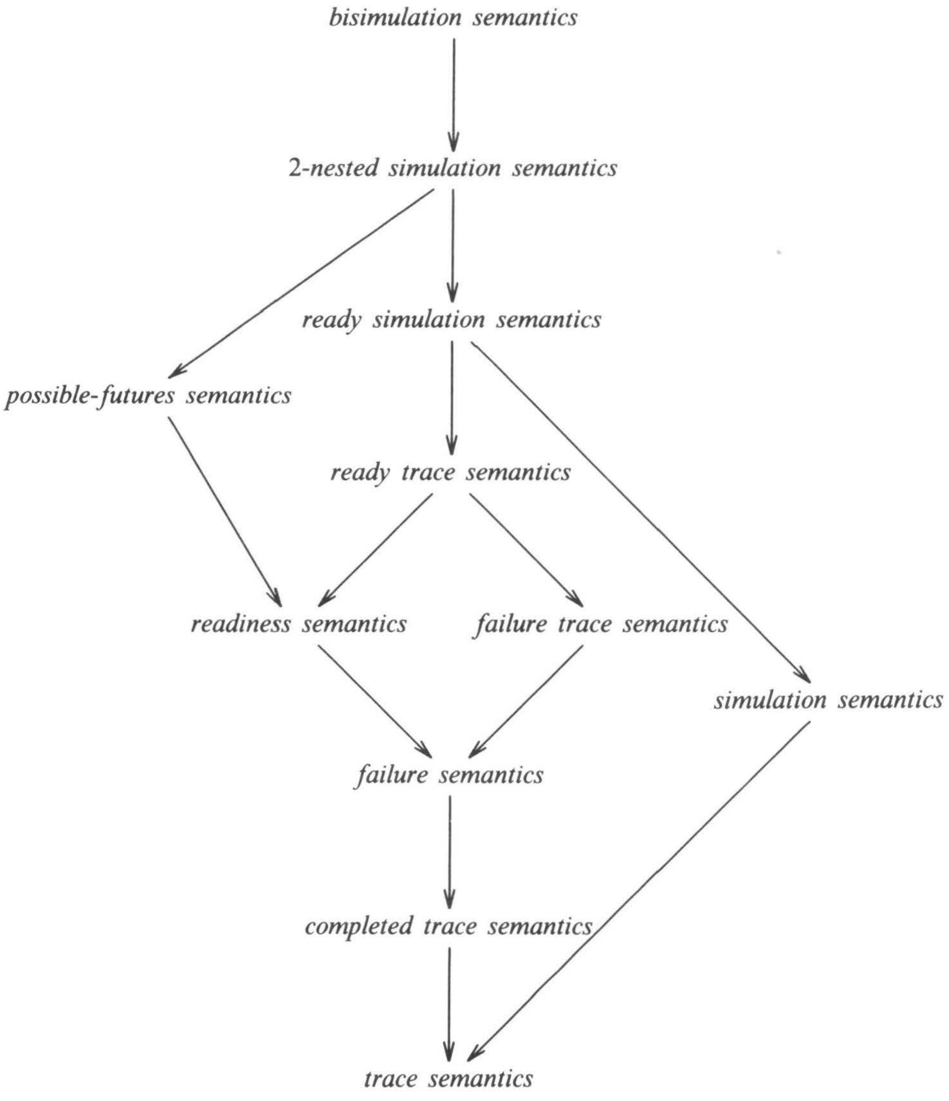


FIGURE 1. *The linear time - branching time spectrum*

The coarsest one (i.e. the semantics making the most identifications) is *trace semantics*, as presented in HOARE [75]. In trace semantics only *partial traces* are employed. The finest one (making less identifications than any of the others) is *bisimulation semantics*, as presented in MILNER [94]. Bisimulation semantics is the standard semantics for the system description language CCS (MILNER [92]). The notion of bisimulation was introduced in PARK [103]. Bisimulation equivalence is a refinement of *observational equivalence*, as introduced by HENNESSY & MILNER in [72]. On the domain of finitely branching, concrete, sequential processes, both equivalences coincide. Also the semantics of DE BAKKER & ZUCKER, presented in [15], coincides with bisimulation semantics on this domain. Then there are nine semantics in between. First of all a variant of trace semantics can be obtained by using *complete traces* besides (or instead of) partial ones. In this chapter it is called *completed trace semantics*. *Failure semantics* is introduced in BROOKES, HOARE & ROSCOE [33], and used in the construction of a model for the system description language CSP (HOARE [76]). It is finer than completed trace semantics. The semantics based on *testing equivalences*, as developed in DE NICOLA & HENNESSY [43], coincides with failure semantics on the domain of finitely branching, concrete, sequential processes, as do the semantics of KENNAWAY [79] and DARONDEAU [38]. This has been established in DE NICOLA [42]. In OLDEROG & HOARE [102] *readiness semantics* is presented, which is slightly finer than failure semantics. Between readiness and bisimulation semantics one finds *ready trace semantics*, as introduced independently in PNUELI [106] (there called *barbed semantics*), BAETEN, BERGSTRA & KLOP [10] and POMELLO [107] (under the name *exhibited behaviour semantics*). The natural completion of the square, suggested by failure, readiness and ready trace semantics yields *failure trace semantics*. For finitely branching processes this is the same as *refusal semantics*, introduced in PHILLIPS [105]. *Simulation equivalence*, based on the classical notion of *simulation* (see e.g. PARK [103]), is independent of the last five semantics. *Ready simulation semantics* was introduced in BLOOM, ISTRAIL & MEYER [28] under the name *GSOS trace congruence*. It is finer than ready trace as well as simulation equivalence. In LARSEN & SKOU [86] a more operational characterization of this equivalence was given under the name *⅔-bisimulation equivalence*. This characterization resembles the one used in this chapter. Finally *2-nested simulation equivalence*, introduced in GROOTE & VAANDRAGER [68], is located between ready simulation and bisimulation equivalence, and *possible-futures semantics*, as proposed in ROUNDS & BROOKES [112], can be positioned between 2-nested simulation and readiness semantics. Among the semantics which are not definable in terms of action relations and thus fall outside the scope of this chapter, one finds semantics that take stochastic properties of processes into account, as in VAN GLABBEEK, SMOLKA, STEFFEN & TOFTS [58] and semantics that make almost no identifications and are hardly used for system verification.

About the contents. In the first section of this chapter all semantics are defined, and motivated by several testing scenarios, which are phrased in terms of button pushing experiments. In Section 2 the semantics are partially ordered by the relation 'makes at least as many identifications as'. This yields the infinitary linear time - branching time spectrum. Counterexamples are provided, showing that on a graph domain this ordering cannot be further expanded. However, for deterministic processes the spectrum collapses, as was first observed by PARK [103]. Finally, in Section 3, nine of these semantics are applied to a simple language for finite, concrete, sequential, nondeterministic processes, and for each of them a complete axiomatization is provided.

1. SEMANTIC EQUIVALENCES ON LABELLED TRANSITION SYSTEMS

1.1. Labelled transition systems. In this thesis processes will be investigated, that are capable of performing actions from a given set Act. By an action any activity is understood that is considered as a conceptual entity on a chosen level of abstraction. Actions may be instantaneous or durational and are not required to terminate, but in a finite time only finitely many actions can be carried out. Any activity of an investigated process should be part of some action $a \in Act$ performed by the process. Different activities that are indistinguishable on the chosen level of abstraction are interpreted as occurrences of the same action $a \in Act$.

A process is *sequential* if it can perform at most one action at the same time. In this chapter only sequential processes will be considered. A domain of sequential processes can often be conveniently represented as a labelled transition system. This is a domain $\mathbf{A}$ on which infix written binary predicates $\xrightarrow{a}$ are defined for each action $a \in Act$. The elements of $\mathbf{A}$ represent processes, and $p \xrightarrow{a} q$ means that p can start performing the action a and after completion of this action reach a state where q is its remaining behaviour. In a labelled transition system it may happen that $p \xrightarrow{a} q$ and $p \xrightarrow{b} r$ for different actions a and b or different processes p and q. This phenomena is called *branching*. It need not be specified how the choice between the alternatives is made, or whether a probability distribution can be attached to it.

NOTATION: For any alphabet Σ, let Σ^* be the set of *strings* over Σ. Write ϵ for the empty string, $\sigma\rho$ for the concatenation of σ and $\rho \in \Sigma^*$, and a for the string, consisting of the single symbol $a \in \Sigma$.

DEFINITION: A *labelled transition system* is a pair $(\mathbf{A}, \rightarrow)$ with $\mathbf{A}$ a class and $\rightarrow \subseteq \mathbf{A} \times Act \times \mathbf{A}$, such that for $p \in \mathbf{A}$ and $a \in Act$ the class $\{q \in \mathbf{A} \mid p \xrightarrow{\sigma} q\}$ is a set.

Let for the remainder of this section $(\mathbf{A}, \rightarrow)$ be a labelled transition system, ranged over by $p,q,r,\ldots.$ Write $p \xrightarrow{a} q$ for $(p,a,q) \in \rightarrow$. The binary predicates $\xrightarrow{a}$ are called *action relations*.

DEFINITIONS (Remark that the following concepts are defined in terms of action relations only):

- The *generalized action relations* $\xrightarrow{\sigma}$ for $\sigma \in Act^*$ are defined inductively by:
  1. $p \xrightarrow{\epsilon} p$, for any process p.
  2. $(p,a,q) \in \rightarrow$ with $a \in Act$ implies $p \xrightarrow{a} q$ with $a \in Act^*$.
  3. $p \xrightarrow{\sigma} q \xrightarrow{\rho} r$ implies $p \xrightarrow{\sigma\rho} r$.

  In words: the generalized action relations $\xrightarrow{\sigma}$ are the reflexive and transitive closure of the ordinary action relations $\xrightarrow{a}$. $p \xrightarrow{\sigma} q$ means that p can evolve into q, while performing the sequence σ of actions. Remark that the overloading of the notion $p \xrightarrow{a} q$ is quite harmless.
- The set of *initial actions* of a process p is defined by: $I(p) = \{a \in Act \mid \exists q : p \xrightarrow{a} q\}$.
- A process $p \in \mathbf{A}$ is *finitely branching* if for each $q \in \mathbf{A}$ with $p \xrightarrow{\sigma} q$ for some $\sigma \in Act^*$, the set $\{(a,r) \mid q \xrightarrow{a} r,\ a \in Act,\ r \in \mathbf{A}\}$ is finite.

In the following, several semantic equivalences on $\mathbf{A}$ will be defined in terms of action relations. Most of these equivalences can be motivated by the observable behaviour of processes, according to some testing scenario. (Two processes are equivalent if they allow the same set of possible observations, possibly in response on certain experiments.) I will try to capture these motivations in terms of *button pushing experiments* (cf. MILNER [92], pp. 10-12).

1.2. Trace semantics. $\sigma \in Act^*$ is a *trace* of a process p, if there is a process q, such that $p \xrightarrow{\sigma} q$. Let $T(p)$ denote the set of traces of p. Two processes p and q are *trace equivalent* if $T(p) = T(q)$. In trace semantics two processes are identified iff they are trace equivalent.

Trace semantics is based on the idea that two processes are to be identified if they allow the same set of observations, where an observation simply consists of a sequence of actions performed by the process in succession.

1.3. Completed trace semantics. $\sigma \in Act^*$ is a *complete trace* of a process p, if there is a process q, such that $p \xrightarrow{\sigma} q$ and $I(q) = \varnothing$. Let $CT(p)$ denote the set of complete traces of p. Two processes p and q are *completed trace equivalent* if $T(p) = T(q)$ and $CT(p) = CT(q)$. In completed trace semantics two processes are identified iff they are completed trace equivalent.

Completed trace semantics can be explained with the following (rather trivial) *completed trace machine*. The process is modelled as a black box that contains as its interface to the outside world a display on which the name of the action is shown that is currently carried out by the process. The process autonomously chooses an execution path that is consistent with its position in the labelled transition system $(\mathbf{A}, \rightarrow)$. During this execution always an action

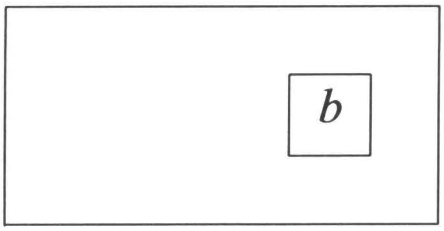


FIGURE 2. *The completed trace machine*

name is visible on the display. As soon as no further action can be carried out, the process reaches a state of deadlock and the display becomes empty. Now the existence of an observer is assumed that watches the display and records the sequence of actions displayed during a run of the process, possibly followed by deadlock. It is assumed that an observation takes only a finite amount of time and may be terminated before the process stagnates. Two processes are identified if they allow the same set of observations in this sense.

The *trace machine* can be regarded as a simpler version of the completed trace machine, were the last action name remains visible in the display if deadlock occurs (unless deadlock occurs in the beginning already). On this machine traces can be recorded, but stagnation can not be detected, since in case of deadlock the observer may think that the last action is still continuing.

1.4. Failure semantics. The *failure machine* contains as its interface to the outside world not only the display of the completed trace machine, but also a switch for each action $a \in Act$ (as in Figure 3).

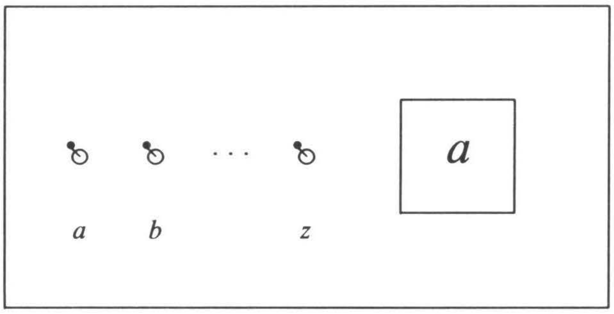


FIGURE 3. *The failure trace machine*

By means of these switches the observer may determine which actions are *free* and which are *blocked.* This situation may be changed any time during a run of the process. As before, the process autonomously chooses an execution path that fits with its position in $(\mathbf{A},\rightarrow)$, but this time the process may only start the execution of free actions. If the process reaches a state where all initial actions of its remaining behaviour are blocked, it can not proceed and the machine stagnates, which can be recognized from the empty display. In this case the observer may record that after a certain sequence of actions σ, the set X of free actions is refused by the process. X is therefore called a *refusal set* and $<\sigma,X>$ a *failure pair*. The set of all failure pairs of a process is called its *failure set*, and constitutes its observable behaviour.

DEFINITION: $<\sigma,X>\in Act^{*}\times\mathcal{P}(Act)$ is a *failure pair* of a process p, if there is a process q, such that $p \xrightarrow{\sigma} q$ and $I(q)\cap X=\varnothing$. Let $F(p)$ denote the set of failure pairs of p. Two processes p and q are *failure equivalent* if $F(p)=F(q)$. In failure semantics two processes are identified iff they are failure equivalent.

This version of failure semantics is taken from HOARE [76]. In BROOKES, HOARE & ROSCOE [33], where failure semantics was introduced, the refusal sets are required to be finite. It is not difficult to see that for finitely branching processes the two versions yield the same failure equivalence. In fact this follows immediately from the following proposition, that says that, for finitely branching processes, the failure pairs with infinite refusal set are completely determined by the ones with finite refusal set.

PROPOSITION 1.1: Let $p\in\mathbf{A}$ and $\sigma\in T(p)$. Put $Cont(\sigma)=\{a\in Act \mid \sigma a\in T(p)\}$.
i. Then, for $X\subseteq Act$, $<\sigma,X>\in F(p) \Leftrightarrow <\sigma,X\cap Cont(\sigma)>\in F(p)$.
ii. If p is finitely branching then $Cont(\sigma)$ is finite.
PROOF: Straightforward. □

In DE NICOLA [42] several equivalences, that were proposed in KENNAWAY [79], DARONDEAU [38] and DE NICOLA & HENNESSY [43], are shown to coincide with failure semantics on the domain of finitely branching transition systems without internal moves. For this purpose he uses the following alternative characterization of failure equivalence.

DEFINITION: Write p *after* σ *MUST* X if for each $q\in\mathbf{A}$ with $p \xrightarrow{\sigma} q$ there is an $r\in\mathbf{A}$ and $a\in X$ such that $q \xrightarrow{a} r$. Put $p\simeq q$ if for all $\sigma\in Act^{*}$ and $X\subseteq Act$: p *after* σ *MUST* $X \Leftrightarrow q$ *after* σ *MUST* X.

PROPOSITION 1.2: Let $p,q\in\mathbf{A}$. Then $p\simeq q \Leftrightarrow F(p)=F(q)$.
PROOF: p *after* σ *MUST* $X \Leftrightarrow (\sigma,X)\notin F(p)$ [42]. □

In HENNESSY [70], a model for nondeterministic behaviours is proposed in which a process is represented as an *acceptance tree*. An acceptance tree of a finitely branching process p without internal moves or internal nondeterminism can be represented as the set of all pairs $<\sigma,X>\in Act^{*}\times\mathcal{P}(Act)$ for which there is a process q, such that $p \xrightarrow{\sigma} q$ and $X\subseteq I(q)$. It follows that for such processes *acceptance tree equivalence* coincides with failure equivalence.

1.5. Failure trace semantics. The *failure trace machine* has the same layout as the failure machine, but is does not stagnate permanently if the process cannot proceed due to the circumstance that all actions it is prepared to continue with are blocked by the observer. Instead it idles - recognizable from the empty display - until the observer changes its mind and allows one of the actions the process is ready to perform. What can be observed are traces with idle periods in between, and for each such period the set of actions that are not blocked by the observer. Such observations can be coded as sequences of members and subsets of Act.

EXAMPLE: The sequence $\{a,b\}cdb\{b,c\}\{b,c,d\}a(Act)$ is the account of the following observation: At the beginning of the execution of the process p, only the actions a and b were allowed by the observer. Apparently, these actions were not on the menu of p, for p started with an idle period. Suddenly the observer canceled its veto on c, and this resulted in the execution of c, followed by d and b. Then again an idle period occurred, this time when b and c were the actions not being blocked by the observer. After a while the observer decided to allow d as well, but the process ignored this gesture and remained idle. Only when the observer gave the green light for the action a, it happened immediately. Finally, the process became idle once more, but this time not even one action was blocked. This made the observer realize that a state of eternal stagnation had been reached, and disappointed he terminated the observation.

A set $X \subseteq Act$, occurring in such a sequence, can be regarded as an offer from the environment, that is refused by the process. Therefore such a set is called a *refusal set*. The occurrence of a refusal set may be interpreted as a 'failure' of the environment to create a situation in which the process can proceed without being disturbed. Hence a sequence over $Act \cup \mathcal{P}(Act)$, resulting from an observation of a process p may be called a *failure trace* of p. The observable behaviour of a process, according to this testing scenario, is given by the set of its failure traces, its *failure trace set*. The semantics in which processes are identified iff their failure trace sets coincide, is called *failure trace semantics*.

DEFINITIONS:

- The *refusal relations* $\xrightarrow{X}$ for $X \subseteq Act$ are defined by: $p \xrightarrow{X} q$ iff $p = q$ and $I(p) \cap X = \varnothing$.

  $p \xrightarrow{X} q$ means that p can evolve into q, while being idle during a period in which X is the set of actions allowed by the environment.
- The *failure trace relations* $\xrightarrow{\sigma}$ for $\sigma \in (Act \cup \mathcal{P}(Act))^*$ are defined as the reflexive and transitive closure of both the action and the refusal relations. Again the overloading of notation is harmless.
- $\sigma \in (Act \cup \mathcal{P}(Act))^*$ is a *failure trace* of a process p, if there is a process q, such that $p \xrightarrow{\sigma} q$. Let $FT(p)$ denote the set of failure traces of p. Two

processes p and q are *failure trace equivalent* if $FT(p)=FT(q)$.

EXERCISES:

1. Explain why $a\{a,b\}a$ can never be a failure trace of a process $p\in\mathbf{A}$.
2. Can $\{a\}b$ and $\{b\}a$ be two failure traces of such a process? And $a\{a\}b$ and $a\{b\}a$?
3. $\{a,b\}cc$, $\{a\}c\{b\}c$, $\{b\}c\{a\}c$, $c\{a,b\}c$, $c\{a\}\{b\}c$ and c are failure traces of a process $p\in\mathbf{A}$. Which selections from this series provide the same information about p?

1.6. Ready trace semantics. The *Ready trace machine* is a variant of the failure trace machine that is equipped with a lamp for each action $a\in Act$.

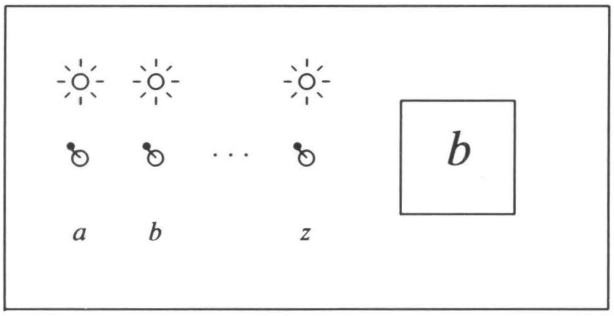


FIGURE 4. *The ready trace machine*

Each time the process idles, the lamps of all actions the process is ready to engage in are lit. Of course all these actions are blocked by the observer, otherwise the process wouldn't idle. Now the observer can see which actions could be released in order to let the process proceed. During the execution of an action no lamps are lit. An observation now consists of a sequence of members and subsets of $\mathbf{A}$, the actions representing information obtained from the display, and the sets of actions representing information obtained from the lights. Such a sequence is called a *ready trace* of the process, and the subsets occurring in a ready trace are referred to as *menus*. The information about the free and blocked actions is now redundant. The set of all ready traces of a process is called its *ready trace set*, and constitutes its observable behaviour.

DEFINITIONS:

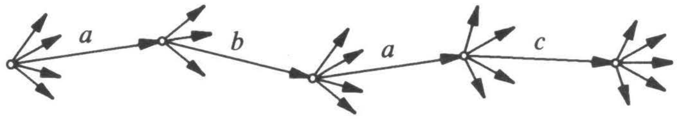
- The *ready trace relations* $\overset{\sigma}{*\!\!\!\rightarrow}$ for $\sigma\in(Act\cup\mathcal{P}(Act))^*$ are defined inductively by:
  1. $p \overset{\epsilon}{*\!\!\!\rightarrow} p$, for any process p.
  2. $p \xrightarrow{a} q$ implies $p \overset{a}{*\!\!\!\rightarrow} q$.

3. $p \overset{X}{\ast\!\!\!\rightarrow} q$ with $X \subseteq Act$ whenever $p = q$ and $I(p) = X$.
4. $p \overset{\sigma}{\ast\!\!\!\rightarrow} q \overset{\rho}{\ast\!\!\!\rightarrow} r$ implies $p \overset{\sigma\rho}{\ast\!\!\!\rightarrow} r$.

The special arrow $\overset{\sigma}{\ast\!\!\!\rightarrow}$ had to be used, since further overloading of $\xrightarrow{\sigma}$ would cause confusion with the failure trace relations.

- $\sigma \in (Act \cup \mathcal{P}(Act))^*$ is a *ready trace* of a process p, if there is a process q, such that $p \overset{\sigma}{\ast\!\!\!\rightarrow} q$. Let $RT(p)$ denote the set of ready traces of p. Two processes p and q are *ready trace equivalent* if $RT(p) = RT(q)$. In ready trace semantics two processes are identified iff they are ready trace equivalent.

In BAETEN, BERGSTRA & KLOP [10], PNUELI [106] and POMELLO [107] ready trace semantics was defined slightly differently. By the proposition below, their definition yields the same equivalence as mine.



DEFINITION: $X_0 a_1 X_1 a_2 \cdots a_n X_n \in \mathcal{P}(Act) \times (Act \times \mathcal{P}(Act))^*$ is a *normal ready trace* of a process p, if there are processes $p_1, \cdots, p_n$ such that $p \xrightarrow{a_1} p_1 \xrightarrow{a_2} \cdots \xrightarrow{a_n} p_n$ and $I(p_i) = X_i$ for $i = 1, \cdots, n$. Let $RT_N(p)$ denote the set of normal ready traces of p. Two processes p and q are ready trace equivalent in the sense of [10, 106, 107] if $RT_N(p) = RT_N(q)$.

PROPOSITION 1.3: Let $p, q \in \mathbb{A}$. Then $RT_N(p) = RT_N(q) \Leftrightarrow RT(p) = RT(q)$.
PROOF: The normal ready traces of a process are just the ready traces which are an alternating sequence of sets and actions, and vice versa the set of all ready traces can be constructed form the set of normal ready traces by means of doubling and leaving out menus. □

1.7. Readiness semantics. The *readiness machine* has the same layout as the ready trace machine, but, like the failure machine, can not recover from an idle period. By means of the lights the menu of initial actions of the remaining behaviour of an idle process can be recorded, but this happens at most once during an observation of a process, namely at the end. An observation either results in a trace of the process, or in a pair of a trace and a menu of actions by which the observation could have been extended if the observer wouldn't have blocked them. Such a pair is called a *ready pair* of the process, and the set of all ready pairs of a process is its *ready set*.

DEFINITION: $<\sigma, X> \in Act^* \times \mathcal{P}(Act)$ is a *ready pair* of a process p, if there is a process q, such that $p \xrightarrow{\sigma} q$ and $I(q) = X$. Let $R(p)$ denote the set of ready pairs of p. Two processes p and q are *ready equivalent* if $R(p) = R(q)$. In readiness semantics two processes are identified iff they are ready equivalent.

Two preliminary versions of readiness semantics were proposed in ROUNDS & BROOKES [112]. In *possible-futures semantics* the menu consists of the entire trace set of remaining behaviour of an idle process, instead of only the set of its initial actions; in *acceptance-refusal semantics* a menu may be any finite subset of initial actions, while also the finite refusal sets of Subsection 1.4 are observable.

DEFINITION: $<\sigma, X> \in Act^* \times \mathcal{P}(Act^*)$ is a *possible-future* of a process p, if there is a process q, such that $p \xrightarrow{\sigma} q$ and $T(q)=X$. Let $PF(p)$ denote the set of possible futures of p. Two processes p and q are *possible-futures equivalent* if $PF(p)=PF(q)$.

DEFINITION: $<\sigma, X, Y> \in Act^* \times \mathcal{P}(Act) \times \mathcal{P}(Act)$ is a *acceptance-refusal triple* of a process p, if X and Y are finite and there is a process q, such that $p \xrightarrow{\sigma} q$, $X \subseteq I(q)$ and $Y \cap I(q) = \varnothing$. Let $AR(p)$ denote the set of acceptance-refusal triples of p. Two processes p and q are *acceptance-refusal equivalent* if $AR(p)=AR(q)$.

It is not difficult to see that for finitely branching processes acceptance-refusal equivalence coincides with readiness equivalence: $<\sigma, X>$ is a ready pair of a process p iff p has an acceptance-refusal triple $<\sigma, X, Y>$ with $X \cup Y = Cont(\sigma)$ (as defined in the proof of Proposition 1.1).

1.8. Infinite observations. All testing scenarios up till now assumed that an observation takes only a finite amount of time. However, they can be easily adapted in order to take infinite behaviours into account.

DEFINITION:

- For any alphabet Σ, let Σ^ω be the set of infinite sequences over Σ.
- $a_1 a_2 \cdots \in Act^\omega$ is an *infinite trace* of a process $p \in \mathbf{A}$, if there are processes $p_1, p_2, \cdots$ such that $p \xrightarrow{a_1} p_1 \xrightarrow{a_2} \cdots$. Let $T^\omega(p)$ denote the set of infinite traces of p.
- Two processes p and q are *infinitary trace equivalent* if $T(p)=T(q)$ and $T^\omega(p)=T^\omega(q)$.
- p and q are *infinitary completed trace equivalent* if $CT(p)=CT(q)$ and $T^\omega(p)=T^\omega(q)$. Note that in this case also $T(p)=T(q)$.
- p and q are *infinitary failure equivalent* if $F(p)=F(q)$ and $T^\omega(p)=T^\omega(q)$.
- p and q are *infinitary ready equivalent* if $R(p)=R(q)$ and $T^\omega(p)=T^\omega(q)$.
- Infinitary failure traces and infinitary ready traces $\sigma \in (Act \cup \mathcal{P}(Act))^\omega$ and the corresponding sets $FT^\omega(p)$ and $RT^\omega(p)$ are defined in the obvious way. Two processes p and q are *infinitary failure trace equivalent* if $FT^\omega(p)=FT^\omega(q)$, and likewise for infinitary ready trace equivalence.

With Königs lemma one easily proves that for finitely branching processes all infinitary equivalences coincide with the corresponding finitary ones.

1.9. Simulation semantics. The testing scenario for finitary simulation semantics resembles that for trace semantics, but in addition the observer is, at any time during a run of the investigated process, capable of making arbitrary (but finitely) many copies of the process in its present state and observe them independently. Thus an observation yields a tree rather than a sequence of actions. Such a tree can be coded as an expression in a simple modal language.

DEFINITIONS:

- The set $\mathcal{L}_S$ of *simulation formulas* over Act is defined inductively by:
  1. $T \in \mathcal{L}_S$.
  2. If $\phi, \psi \in \mathcal{L}_S$ then $\phi \wedge \psi \in \mathcal{L}_S$.
  3. If $\phi \in \mathcal{L}_S$ and $a \in Act$ then $a\phi \in \mathcal{L}_S$.
- The *satisfaction relation* $\models \subseteq \mathbf{A} \times \mathcal{L}_S$ is defined inductively by:
  1. $p \models T$ for all $p \in \mathbf{A}$.
  2. $p \models \phi \wedge \psi$ if $p \models \phi$ and $p \models \psi$.
  3. $p \models a\phi$ if for some $q \in \mathbf{A}$: $p \xrightarrow{a} q$ and $q \models \phi$.
- Let $S(p)$ denote the set of all simulation formula that are satisfied by the process p:
  $S(p) = \{\phi \in \mathcal{L}_S \mid p \models \phi\}$. Two processes p and q are *finitary simulation equivalent* if $S(p) = S(q)$.

The following concept of *simulation*, occurs frequently in the literature (see e.g. PARK [103]). The derived notion of *simulation equivalence* coincides with finitary simulation equivalence for finitely branching processes.

DEFINITION: A *simulation* is a binary relation R on processes, satisfying, for $a \in Act$:

- if pRq and $p \xrightarrow{a} p'$, then $\exists q'$: $q \xrightarrow{a} q'$ and $p'Rq'$.

Process p *can be simulated by* q, notation $s \underset{\rightarrow}{\subset} t$, if there is a simulation R with pRq.
p and q are *similar*, notation $p \leftrightarrows q$, if $p \underset{\rightarrow}{\subset} q$ and $q \underset{\rightarrow}{\subset} p$.

PROPOSITION 1.4: *Similarity is an equivalence on the domain of processes.*
PROOF: It has to be checked that $p \underset{\rightarrow}{\subset} p$, and $p \underset{\rightarrow}{\subset} q$ & $q \underset{\rightarrow}{\subset} r \Rightarrow p \underset{\rightarrow}{\subset} q$.

- The identity relation is a simulation with pRp.
- If R is a simulation with pRq and S is a simulation with qSr, then the relation $R \circ S$, defined by $x(R \circ S)z$ iff $\exists y$: xRy & ySz, is a simulation with $p(R \circ S)r$. □

Hence the relation will be called *simulation equivalence*.

PROPOSITION 1.5: Let $p,q \in \mathbf{A}$ be finitely branching processes. Then $p \leftrightarrows q \iff S(p)=S(q)$.
PROOF: See HENNESSY & MILNER [73]. □

The testing scenario for simulation semantics differs from that for finitary simulation semantics, in that both the duration of observations and the amount of copies that can be made each time are not required to be finite.

1.10. Ready simulation semantics. Of course one can also combine the copying facility with any of the other testing scenarios. The observer can then plan experiments on one of the generative machines from the Subsections 1.3 to 1.7 together with a *duplicator*, an ingenious device by which one can duplicate the machine whenever and as often as one wants. In order to represent observations, the modal language from the previous subsection needs to be slightly extended.

DEFINITIONS:

- The *completed simulation formulas* and the corresponding satisfaction relation are defined by means of the extra clauses:
  4. $0 \in \mathcal{L}_{CS}$.
  4. $p \models 0$ if $I(p)=\varnothing$.
- For the *failure simulation formulas* one needs:
  4. If $X \subseteq Act$ then $X \in \mathcal{L}_{FS}$.
  4. $p \models X$ if $I(p) \cap X = \varnothing$.
- For the *ready simulation formulas*:
  4. If $X \subseteq Act$ then $X \in \mathcal{L}_{RS}$.
  4. $p \models X$ if $I(p)=X$.
- For the *failure trace simulation formulas*:
  4. If $\phi \in \mathcal{L}_{FTS}$ and $X \subseteq Act$ then $X\phi \in \mathcal{L}_{FTS}$.
  4. $p \models X\phi$ if $I(p) \cap X = \varnothing$ and $p \models \phi$.
- And for the *ready trace simulation formulas*:
  4. If $\phi \in \mathcal{L}_{RTS}$ and $X \subseteq Act$ then $X\phi \in \mathcal{L}_{RTS}$.
  4. $p \models X\phi$ if $I(p)=X$ and $p \models \phi$.

Note that traces, complete traces, failure pairs, etc. can be obtained as the corresponding kind of simulation formulas without the operator $\wedge$.

By means of the formulas defined above one can define the finitary versions of *completed simulation equivalence*, *ready simulation equivalence*, etc. It is obvious that failure trace simulation equivalence coincides with failure simulation equivalence and ready trace simulation equivalence with ready simulation equivalence ($p \models X\phi \Leftrightarrow p \models X \wedge \phi$). Also it is not difficult to see that failure simulation equivalence and ready simulation equivalence coincide. So two different equivalences remain. For finitely branching processes the finitary versions of these two equivalences coincide with the following infinitary versions.

DEFINITION: A *complete simulation* is a binary relation R on processes, satisfying, for $a \in Act$:

- if pRq and $p \xrightarrow{a} p'$, then $\exists q'$: $q \xrightarrow{a} q'$ and $p'Rq'$;
- if pRq then $I(p) = \varnothing \Leftrightarrow I(q) = \varnothing$.

Two processes p and q are *completed simulation equivalent* if there exists a complete simulation R with pRq and a complete simulation S with qSp.

DEFINITION: A *ready simulation* is a binary relation R on processes, satisfying, for $a \in Act$:

- if pRq and $p \xrightarrow{a} p'$, then $\exists q'$: $q \xrightarrow{a} q'$ and $p'Rq'$;
- if pRq then $I(p) = I(q)$.

Two processes p and q are *ready simulation equivalent* if there exists a ready simulation R with pRq and a ready simulation S with qSp.

An alternative and maybe more natural testing scenario for finitary ready simulation semantics (or simulation semantics) can be obtained by exchanging the duplicator for an *undo*-button on the (ready) trace machine (Figure 5).

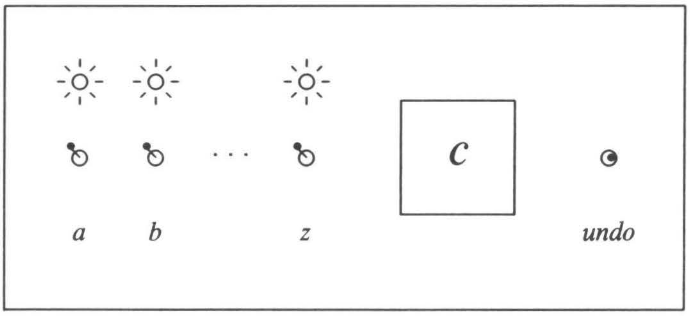


FIGURE 5. *The ready simulation machine*

It is assumed that all intermediate states that are past through during a run of a process are stored in a memory inside the black box. Now pressing the *undo*-button causes the machine to shift one state backwards. In case the button is pressed during the execution of an action, this execution will be interrupted and the process assumes the state just before this action began. In the initial state pressing the button has no effect. An observation now consists of a (ready) trace, enriched with *undo*-actions. Such observations can easily be translated in (ready) simulation formulas.

1.11. Refusal (simulation) semantics. In the testing scenarios presented so far, a process is considered to perform actions and make choices autonomously. The investigated behaviours can therefore be classified as *generative processes*. The observer merely restricts the spontaneous behaviour of the generative machine by cutting off some possible courses of action. An alternative view of the investigated processes can be obtained by considering them to react on stimuli from the environment and be passive otherwise. *Reactive machines* can be obtained out of the generative machines presented so far by replacing the switches by buttons and the display by a green light.

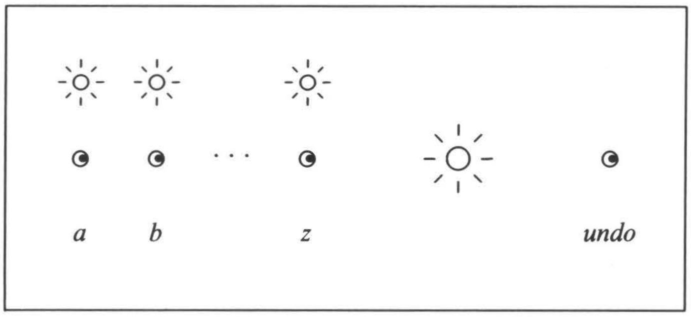


FIGURE 6. *The reactive ready simulation machine*

Initially the process waits patiently until the observer tries to press one of the buttons. If the observer tries to press an a-button, the machine can react in two different ways: if the process can not start with an a-action the button will not go down and the observer may try another one; if the process can start with an a-action it will do so and the button goes down. Furthermore the green light switches on. During the execution of a no buttons can be pressed. As soon as the execution of a is completed the light switches off, so that the observer knows that the process is ready for a new trial. Reactive machines as described above originate from MILNER [92, 93].

Next I will discuss the equivalences that originate from the various reactive machines. First consider the reactive machine that resembles the failure trace machine, thus without menu-lights and *undo*-button. An observation on such a machine consists of a sequence of accepted and refused actions. Such a sequence can be modelled as a failure trace where all refusal sets are singletons. For finitely branching processes the resulting equivalence is exactly the equivalence that originates from PHILLIPS notion of *refusal testing* [105]. There it is called *refusal equivalence*. The following proposition shows that for finitely branching processes refusal equivalence coincides with failure equivalence.

PROPOSITION 1.6: Let $p \in \mathbf{A}$, $\sigma \in FT(p)$ and $Cont(\sigma) = \{a \in Act \mid \sigma a \in FT(p)\}$.
i. Then, for $X \subseteq Act$, $\sigma X \rho \in FT(p) \Leftrightarrow \sigma(X \cap Cont(\sigma))\rho \in FT(p)$.
ii. If p is finitely branching then $Cont(\sigma)$ is finite.
iii. $\sigma(X \cup Y)\rho \in FT(p) \Leftrightarrow \sigma XY\rho \in FT(p)$.
PROOF: Straightforward. □

If the menu-lights are added to the reactive failure trace machine considered above one can observe ready trace sets, and the green light is redundant. If the green light (as well as the menu-lights) are removed one can only test trace equivalence, since any refusal may be caused by the last action not being ready yet. Reactive machines seem to be unsuited for testing completed trace and failure equivalence. If the menu-lights and the *undo*-button are added to the reactive failure trace machine one gets ready simulation again and if only the *undo*-button is added one obtains an equivalence that may be called *refusal simulation equivalence* and coincides with ready simulation equivalence on the domain of finitely branching processes. The following *refusal simulation formulas* originate from BLOOM, ISTRAIL & MEYER [28].

DEFINITION: The *refusal simulation formulas* and the corresponding satisfaction relation are defined by adding to the definitions of Subsection 1.9 the following extra clauses:
4. If $a \in Act$ then $\neg a \in \mathcal{L}_{CS}$.
4. $p \models \neg a$ if $a \notin I(p)$.

1.12. 2-nested simulation semantics. *2-nested simulation equivalence* popped up naturally in GROOTE & VAANDRAGER [68] as the coarsest congruence with respect to a large and general class of operators that is finer than completed trace equivalence. In order to obtain a testing scenario for this equivalence one has to introduce the rather unnatural notion of a *lookahead* [68]: The *2-nested simulation machine* is a variant of the ready trace machine with duplicator, where in an idle state the machine not only tells which actions are on the menu, but even which simulation formulas are satisfied in the current state.

DEFINITION: A *2-nested simulation* is a simulation contained in simulation equivalence ($\leftrightarrows$). p and q are *2-nested simulation equivalent* if there exists a 2-nested simulation R with pRq and a 2-nested simulation S with qSp.

1.13. Bisimulation semantics. The testing scenario for bisimulation semantics, as presented in MILNER [92] is the oldest and most powerful testing scenario, from which most others have been derived by omitting some of its features. It was based on a reactive failure trace machine with duplicator, but additionally the observer is equipped with the capacity of *global testing*. Global testing is described in ABRAMSKY [1] as: "the ability to enumerate all (of finitely many) possible 'operating environments' at each stage of the test, so as to guarantee that all nondeterministic branches will be pursued by various copies of the subject process". MILNER [92] implemented global testing by assuming that

(i) It is the *weather* which determines in each state which a-move will occur in response of pressing the a-button (if the process under investigation is capable of doing an a-move at all);
(ii) The weather has only finitely many states - at least as far as choice-resolution is concerned;
(iii) We can control the weather.

Now it can be ensured that all possible moves a process can perform in reaction on an a-experiment will be investigated by simply performing the experiment in all possible weather conditions. Unfortunately, as remarked in MILNER [93], the second assumption implies that the amount of different a-moves an investigated process can perform is bounded by the number of possible weather conditions; so for general application this condition has to be relaxed.

A different implementation of global testing is given in LARSEN & SKOU [86]. They assumed that every transition in a transition system has a certain positive probability of being taken. Therefore an observer can with an arbitrary high degree of confidence assume that all transitions have been examined, simply by repeating an experiment many times.

As argued among others in BLOOM, ISTRAIL & MEYER [28], global testing in the above sense is a rather unrealistic testing ability. Once you assume that the observer is really as powerful as in the described scenarios, in fact more can be tested then only bisimulation equivalence: in the testing scenario of Milner also the correlation between weather conditions and transitions being taken by the investigated process can be recovered, and in that of Larsen & Skou one can determine the relative probabilities of the various transitions.

An observation in the global testing scenario can be represented as a formula in *Hennessy-Milner logic* [72] (*HML*). An HML formula is a simulation formula in which it is possible to indicate that certain branches are not present.

DEFINITION: The *HML-formulas* and the corresponding satisfaction relation are defined by adding to the definitions in Subsection 1.9 the following extra clauses:

4. If $\phi \in \mathcal{L}$ then $\neg\phi \in \mathcal{L}$.
4. $p \models \neg\phi$ if $p \not\models \phi$.

Let $HML(p)$ denote the set of all HML-formula that are satisfied by the process p: $HML(p) = \{\phi \in \mathcal{L} \mid p \models \phi\}$. Two processes p and q are *HML-equivalent* if $HML(p) = HML(q)$.

For finitely branching processes HENNESSY & MILNER [72] provided the following characterization of this equivalence.

DEFINITION: Let $p, q \in \mathbf{A}$ be finitely branching processes. Then:

- $p \sim_0 q$ is always true.
- $p \sim_{n+1} q$ if for all $a \in Act$:
    - $p \xrightarrow{a} p'$ implies $\exists q'$: $q \xrightarrow{a} q'$ and $p' \sim_n q'$;

- $q \xrightarrow{a} q'$ implies $\exists p'$: $p \xrightarrow{a} p'$ and $p' \sim_n q'$.
- p and q are *observational equivalent*, notation $p \sim q$, if $p \sim_n q$ for every $n \in \mathbb{N}$.

PROPOSITION 1.7: Let $p,q \in \mathbf{A}$ be finitely branching processes. Then $p \sim q \Leftrightarrow HML(p) = HML(q)$.
PROOF: In HENNESSY & MILNER [73]. □

As observed by PARK [103], for finitely branching processes observation equivalence can be reformulated as bisimulation equivalence.

DEFINITION: A *bisimulation* is a binary relation R on processes, satisfying, for $a \in Act$:

- if pRq and $p \xrightarrow{a} p'$, then $\exists q'$: $q \xrightarrow{a} q'$ and $p'Rq'$;
- if pRq and $q \xrightarrow{a} q'$, then $\exists p'$: $p \xrightarrow{a} p'$ and $p'Rq'$.

Two processes p and q are *bisimilar*, notation $p \leftrightarrows q$, if there exists a bisimulation R with pRq.

The relation $\leftrightarrows$ is again a bisimulation. As for similarity, one easily checks that bisimilarity is an equivalence on $\mathbf{A}$. Hence the relation will be called *bisimulation equivalence*. Finally note that the concept of bisimulation does not change if in the definition above the action relations $\xrightarrow{a}$ were replaced by generalized action relations $\xrightarrow{\sigma}$.

PROPOSITION 1.8: Let $p,q \in \mathbf{A}$ be finitely branching processes. Then $p \leftrightarrows q \Leftrightarrow p \sim q$.
PROOF: "$\Rightarrow$": Straightforward with induction. "$\Leftarrow$" follows from Theorem 5.6 in MILNER [92]. □

For infinitely branching processes $\sim$ is coarser then $\leftrightarrows$ and will be called *finitary bisimulation equivalence*.

Another characterization of bisimulation semantics can be given by means of ACZEL's universe $\mathcal{V}$ of non-well-founded sets [4]. This universe is an extension of the Von Neumann universe of well-founded sets, where the axiom of foundation (every chain $x_0 \ni x_1 \ni \cdots$ terminates) is replaced by an *anti-foundation axiom*.

DEFINITION: Let B denote the unique function $\mathcal{B}:\mathbf{A} \rightarrow \mathcal{V}$ satisfying $\mathcal{B}(p) = \{<a, \mathcal{B}(q)> | p \xrightarrow{a} q\}$ for all $p \in \mathbf{A}$. Two processes p and q are *branching equivalent* if $B(p) = B(q)$.

It follows from Aczel's anti-foundation axiom that such a solution exists. In fact the axiom amounts to saying that systems of equations like the one above have unique solutions. In [4] there is also a section on communicating systems. There two processes are identified iff they are branching equivalent.

A similar idea underlies the semantics of DE BAKKER & ZUCKER [15], but there the domain of processes is a complete metric space and the definition of B above only works for finitely branching processes, and only if $=$ is interpreted as *isometry*, rather then equality, in order to stay in well-founded set theory. For finitely branching processes the semantics of De Bakker and Zucker coincides with the one of Aczel and also with bisimulation semantics. This is observed in VAN GLABBEEK & RUTTEN [57], where also a proof can be found of the next proposition, saying that bisimulation equivalence coincides with branching equivalence.

PROPOSITION 1.9: Let $p,q \in \mathbf{A}$. Then $p \leftrightarrows q \Leftrightarrow B(p)=B(q)$.
PROOF: "$\Leftarrow$". Let B be the relation, defined by pBq iff $B(p)=B(q)$, then it suffices to prove that B is a bisimulation. Suppose pBq and $p \xrightarrow{a} p'$. Then $<a,B(p')> \in B(p)=B(q)$. So by the definition of $B(q)$ there must be a process q' with $B(p')=B(q')$ and $q \xrightarrow{a} q'$. Hence $p'Bq'$, which had to be proved. The second requirement for B being a bisimulation follows by symmetry.
"$\Rightarrow$". Let B^* denote the unique solution of

$$\mathcal{B}^*(p)=\{<a,\ \mathcal{B}^*(r')> \mid \exists r: r \leftrightarrows p \ \&\ r \xrightarrow{a} r'\}.$$

As for B it follows from the anti-foundation axiom that such a unique solution exists. From the symmetry and transitivity of $\leftrightarrows$ it follows that

$$p \leftrightarrows q \ \Rightarrow\ B^*(p)=B^*(q). \qquad (*)$$

Hence it remains to be proven that $B^*=B$. This can be done by showing that B^* satisfies the equations $\mathcal{B}(p)=\{<a,\ \mathcal{B}(q)> \mid p \xrightarrow{a} q\}$, which have B as unique solution. So it has to be established that $B^*(p)=\{<a,\ B^*(q)> \mid p \xrightarrow{a} q\}$. The direction "$\supseteq$" follows directly from the reflexivity of $\leftrightarrows$. For "$\subseteq$", suppose $<a,X> \in B^*(p)$. Then $\exists r: r \leftrightarrows p$, $r \xrightarrow{a} r'$ and $X=B^*(r')$. Since $\leftrightarrows$ is a bisimulation, $\exists p': p \xrightarrow{a} p'$ and $r' \leftrightarrows p'$. Now from (*) it follows that $X=B^*(r')=B^*(p')$. Therefore $<a,X> \in \{<a,\ B^*(q)> \mid p \xrightarrow{a} q\}$, which had to be established. □

2. THE SEMANTIC LATTICE

2.1. Ordering the equivalences for finitely branching processes. In Section 1 twelve semantics were defined that are different for finitely branching processes. These will be abbreviated by *T, CT, F, R, FT, RT, S, CS, RS, PF, 2S* and *B*. Write $\mathbb{S} \preccurlyeq \mathbb{T}$ if semantics $\mathbb{S}$ makes at least as much identifications as semantics $\mathbb{T}$. This is the case if the equivalence corresponding with $\mathbb{S}$ is equal to or coarser than the one corresponding with $\mathbb{T}$.

THEOREM 2.1: $T \preccurlyeq CT \preccurlyeq F \preccurlyeq R \preccurlyeq RT$, $F \preccurlyeq FT \preccurlyeq RT \preccurlyeq RS \preccurlyeq 2S \preccurlyeq B$, $T \preccurlyeq S \preccurlyeq CS \preccurlyeq RS$, $CT \preccurlyeq CS$ *and* $R \preccurlyeq PF \preccurlyeq 2S$.

PROOF: The first statement is trivial. For the next five statements it suffices to show that $CT(p)$ can be expressed in terms of $F(p)$, $F(p)$ in terms of $R(p)$, $R(p)$ in terms of $RT(p)$, $F(p)$ in terms of $FT(p)$ and $FT(p)$ in terms of $RT(p)$.

- $CT(p) = \{\sigma \in A^* \mid <\sigma, A> \in F(p)\}$.
- $<\sigma, X> \in F(p) \Leftrightarrow \exists Y \subseteq A: <\sigma, Y> \in R(p) \;\&\; X \cap Y = \varnothing$.
- $<\sigma, X> \in R(p) \Leftrightarrow \sigma X \in RT(p)$.
- $<\sigma, X> \in F(p) \Leftrightarrow \sigma X \in FT(p)$.
- $\sigma = \sigma_1 \sigma_2 \cdots \sigma_n \in FT(p)\ (\sigma_i \in A \cup \mathcal{P}(A)) \Leftrightarrow$ $\exists \rho = \rho_1 \rho_2 \cdots \rho_n \in RT(p)\ (\rho_i \in A \cup \mathcal{P}(A))$ such that for $i = 1, \ldots, n$ either $\sigma_i = \rho_i \in A$ or $\sigma_i, \rho_i \subseteq A$ and $\sigma_i \cap \rho_i = \varnothing$.

The remaining statements are (also) trivial. □

Theorem 1 is illustrated in Figure 1. There, however, completed trace semantics is missing, since it did not occur in the literature.

2.2. Ordering the equivalences for infinitely branching processes. When the restriction to finitely branching processes is dropped, there exists a finitary and an infinitary variant of each of these semantics, depending on whether or not infinite observations are taken into account. These versions will be notationally distinguished by means of superscripts '*' and 'ω' respectively; the unsubscripted abbreviation will, for historical reasons, refer to the infinitary versions in case of 'simulation'-like semantics and to the finitary versions otherwise. For the semantics that are based on refusal sets, there exists even a third version, namely when also the refusal sets are required to be finite. These will be denoted by means of a superscript '$-$'. So F^- denotes failure semantics as defined in [33] (see Subsection 1.4), R^- denotes acceptance-refusal semantics [112] (Subsection 1.7), FT^- denotes refusal semantics (Subsection 1.11), RS^- denotes refusal simulation semantics (also Subsection 1.11) and B^- denotes HML-semantics (Subsection 1.13). Now the $\preccurlyeq$-relation is represented by arrows in Figure 7.

THEOREM 2.2: Let $\mathbb{S}, \mathfrak{T}$ be any two of the semantics mentioned above. Then $\mathbb{S} \preccurlyeq \mathfrak{T}$ whenever this is indicated in Figure 7.

Again the proof is straightforward. If the labelled transition system **A** on which these semantic equivalences are defined is large enough, then they are all different and $\mathbb{S} \preccurlyeq \mathfrak{T}$ holds only if this follows from Theorem 2.2 (and the fact that $\preccurlyeq$ is a partial order), as will be shown in Subsection 2.8. However, for certain labelled transition systems much more identifications can be made. Is has been remarked already that for finitely branching processes all semantics that are connected by dashed arrows in Figure 7 coincide. This result will be slightly strengthened in the next subsection. In the subsequent subsection a class of processes will be defined on which all the semantics coincide.

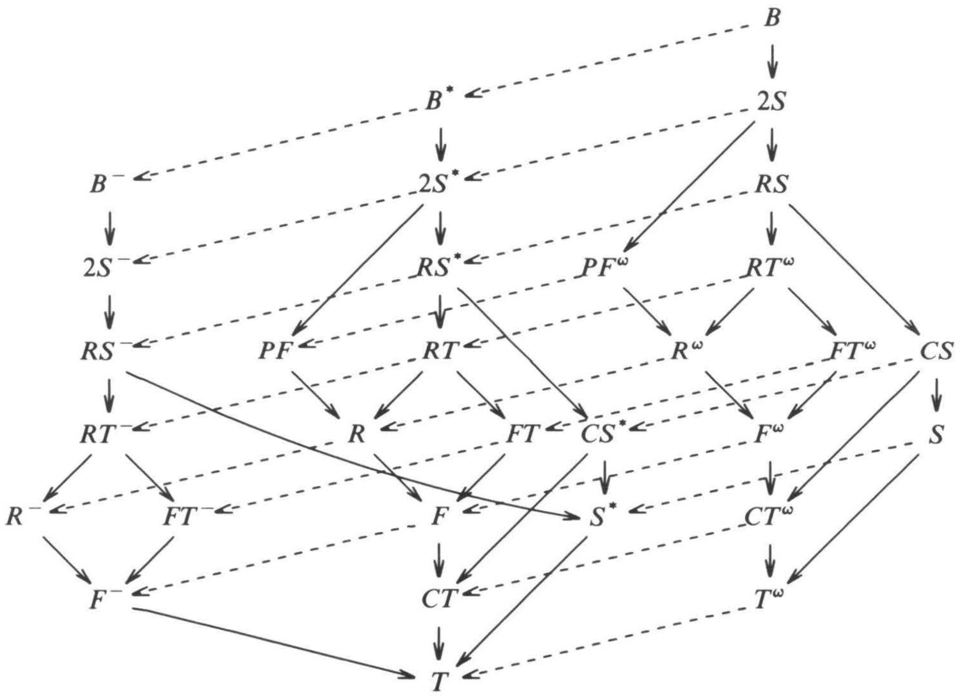


FIGURE 7. *The infinitary linear time - branching time spectrum*

2.3. Image finite processes.

DEFINITION: A process $p \in \mathbf{A}$ is *image finite* if for each $\sigma \in Act^*$ the set $\{q \in \mathbf{A} \mid p \xrightarrow{\sigma} q\}$ is finite.

Note that finitely branching processes are image finite, but the reverse does not hold.

THEOREM 2.3: On a domain of image finite processes, semantics that are connected with a dashed arrow in Figure 7 coincide.

PROOF: For the upper two arrows, connecting HML-semantics with finitary bisimulation semantics and finitary bisimulation semantics with bisimulation semantics, the proof has been given in HENNESSY & MILNER [73]. For the other simulation-like semantics the proof goes likewise. For the trace-like semantics the correspondence between the finitary and infinitary versions (the arrows on the right) follows directly from Königs lemma. Here I only prove the correspondence between F^- and F; the remaining cases can be proved likewise.

It has to be established that, for image finite processes p and $q \in \mathbf{A}$, $F^-(p) = F^-(q) \Rightarrow F(p) = F(q)$, where $F^-(p)$ denotes the set of failure pairs $<\sigma, X>$ of p with finite refusal set X. The reverse implication is trivial. For finitely branching processes $F(p)$ is completely determined by $F^-(p)$ (Proposition 1.1), from which the implication follows. For arbitrary image finite processes this is no longer the case, but the implication still holds.

Let p and $q \in \mathbf{A}$ be two image finite processes with $F(p) \neq F(q)$. Say there is a failure pair $<\sigma, X> \in F(p) - F(q)$. By image finiteness of q there are only finitely many processes r_i with $q \xrightarrow{\sigma} r_i$, and for each of those there is an action $a_i \in I(r_i) \cap X$ (otherwise $<\sigma, X>$ would be a failure pair of q). Let Y be the set of all those a_i's. then Y is a finite subset of X, so $<\sigma, Y> \in F^-(p)$. On the other hand $a_i \in I(r_i) \cap Y$ for all r_i, so $<\sigma, Y> \notin F^-(q)$. □

2.4. Deterministic processes.

DEFINITION: A process p is *deterministic* if $p \xrightarrow{\sigma} q \;\&\; p \xrightarrow{\sigma} r \;\Rightarrow\; q = r$.

REMARK: *If p is deterministic and $p \xrightarrow{\sigma} p'$ then also p' is deterministic.* Hence any domain of processes on which action relations are defined, has a subdomain of deterministic processes with the inherited action relations. (A similar remark can be made for image finite processes.)

PROOF: Suppose $p' \xrightarrow{\rho} q$ and $p' \xrightarrow{\rho} r$. Then $p \xrightarrow{\sigma\rho} q$ and $p \xrightarrow{\sigma\rho} r$, so $q = r$. □

THEOREM 2.4 (PARK [103]): *On a domain of deterministic processes all semantics on the infinitary linear time - branching time spectrum coincide.*

PROOF: Because of Theorem 2.2 it suffices to show that $BS \preccurlyeq TS$. This is the case if $T(p) = T(q) \Rightarrow p \leftrightarrows q$ for any two deterministic processes p and q. Let R be the relation, defined by pRq iff $T(p) = T(q)$, then it suffices to prove that R is a bisimulation. Suppose pRq and $p \xrightarrow{\sigma} p'$. Then $\sigma \in T(p) = T(q)$. So there is a process q' with $q \xrightarrow{\sigma} q'$. Now let $\rho \in T(p')$. Then $\exists r: p' \xrightarrow{\rho} r$. Hence $p \xrightarrow{\sigma\rho} r$ and $\sigma\rho \in T(p) = T(q)$. So there must be a process s with $q \xrightarrow{\sigma\rho} s$. By the definition of the generalized action relations $\exists t: q \xrightarrow{\sigma} t \xrightarrow{\rho} s$, and since q is deterministic, $t = q'$. Thus $\rho \in T(q')$, and from this it follows that $T(p') \subseteq T(q')$. Since also p is deterministic the converse can be established in the same way, and together this yields $T(p') = T(q')$, or $p'Rq'$. This finishes the proof. □

2.5. Process graphs. In process theory it is common practice to represent processes as elements in a mathematical domain. The semantics of a process theory can then be modelled as an equivalence on such a domain. In Section 1 several semantic equivalences were defined on any domain of sequential processes which is provided with action relations. Such a domain was called a labelled transition system. In Section 3 a term domain $\mathbb{P}$ with action relations will be presented for which these definitions apply. The present subsection introduces one of the most popular labelled transition systems: the domain $\mathbb{G}$ of *process graphs* or *state transition diagrams*.

DEFINITION: A *process graph* over a given alphabet *Act* is a rooted, directed graph whose edges are labelled by elements of *Act*. Formally, a process graph g is a triple $(\text{NODES}(g),\text{EDGES}(g),\text{ROOT}(g))$, where

- $\text{NODES}(g)$ is a set, of which the elements are called the *nodes* or *states* of g,
- $\text{ROOT}(g)\in\text{NODES}(g)$ is a special node: the *root* or *initial state* of g,
- and $\text{EDGES}(g)\subseteq\text{NODES}(g)\times Act\times\text{NODES}(g)$ is a set of triples (s,a,t) with $s,t\in\text{NODES}(g)$ and $a\in Act$: the *edges* or *transitions* of g.

If $e=(s,a,t)\in\text{EDGES}(g)$, one says that *e goes from s to t.* A (finite) *path* π in a process graph is an alternating sequence of nodes and edges, starting and ending with a node, such that each edge goes from the node before it to the node after it. If $\pi=s_0(s_0,a_1,s_1)s_1(s_1,a_2,s_2)\cdots(s_{n-1},a_n,s_n)s_n$, also denoted as $\pi\colon s_0\xrightarrow{a_1}s_1\xrightarrow{a_2}\cdots\xrightarrow{a_n}s_n$, one says that π *goes from* s_0 *to* s_n; it *starts* in s_0 and *ends* in $end(\pi)=s_n$. Let $\text{PATHS}(g)$ be the set of paths in g starting from the root. If s and t are nodes in a process graph then *t can be reached from s* if there is a path going from s to t. A process graph is said to be *connected* if all its nodes can be reached from the root; it is a *tree* if each node can be reached from the root by exactly one path. Let $\mathbb{G}$ be the domain of connected process graphs over a given alphabet *Act*.

DEFINITION: For $g\in\mathbb{G}$ and $s\in\text{NODES}(g)$, let g_s be the process graph defined by

- $\text{NODES}(g_s)=\{t\in\text{NODES}(g)\,|\,\textit{there is a path going from s to t}\}$,
- $\text{ROOT}(g_s)=s\in\text{NODES}(g_s)$,
- and $(t,a,u)\in\text{EDGES}(g_s)$ iff $t,u\in\text{NODES}(g_s)$ and $(t,a,u)\in\text{EDGES}(g)$.

Of course $g_s\in\mathbb{G}$. Remark that $g_{\text{ROOT}(g)}=g$. Now on $\mathbb{G}$ action relations $\xrightarrow{a}$ for $a\in Act$ are defined by $g\xrightarrow{a}h$ iff $(\text{ROOT}(g),a,s)\in\text{EDGES}(g)$ and $h=g_s$. This makes $\mathbb{G}$ into a labelled transition system. Hence all semantic equivalences of Section 1 are well-defined on $\mathbb{G}$. Below the sets of observations $O(g)$ for $O\in\{T,\ CT,\ R,\ F,\ RT,\ FT\}$i= and $g\in\mathbb{G}$, are characterized in terms of the paths of g, rather than the generalized action relations between graphs.

DEFINITION: Let $g \in \mathbf{G}$ and let π: $s_0 \xrightarrow{a_1} s_1 \xrightarrow{a_2} \cdots \xrightarrow{a_n} s_n \in \text{PATHS}(g)$. Consider the following notions:

- the *trace* associated to π: $T(\pi)=a_1 a_2 \cdots a_n \in Act^*$;
- the *menu* of a node $s \in \text{NODES}(g)$: $I(s)=\{a \in Act \mid \exists t: (s,a,t) \in \text{EDGES}(g)\}$;
- the *ready pair* associated to π: $R(\pi)=<T(\pi), I(s_n)>$;
- the *failure set* of π: $F(\pi)=\{<T(\pi),X> \mid I(s_n) \cap X = \varnothing\}$;
- the *ready trace set* of π: $RT(\pi)$ is the smallest subset of $(Act \cup \mathcal{P}(Act))^*$ satisfying
  - $I(s_0)a_1 I(s_1)a_2 \cdots a_n I(s_n) \in RT(\pi)$,
  - $\sigma X \rho \in RT(\pi) \;\Rightarrow\; \sigma\rho \in RT(\pi)$,
  - $\sigma X \rho \in RT(\pi) \;\Rightarrow\; \sigma X X \rho \in RT(\pi)$;
- the *failure trace set* of π: $FT(\pi)$ is the smallest subset of $(Act \cup \mathcal{P}(Act))^*$ satisfying
  - $(A - I(s_0))a_1(A - I(s_1))a_2 \cdots a_n(A - I(s_n)) \in FT(\pi)$,
  - $\sigma X \rho \in FT(\pi) \;\Rightarrow\; \sigma\rho \in FT(\pi)$,
  - $\sigma X \rho \in FT(\pi) \;\Rightarrow\; \sigma X X \rho \in FT(\pi)$,
  - $\sigma X \rho \in FT(\pi) \wedge Y \subseteq X \;\Rightarrow\; \sigma Y \rho \in FT(\pi)$;

PROPOSITION 2.5:

$$T(g)=\{T(\pi) \mid \pi \in \text{PATHS}(g)\}$$

$$CT(g)=\{T(\pi) \mid \pi \in \text{PATHS}(g) \wedge I(end(\pi))=\varnothing\}$$

$$R(g)=\{R(\pi) \mid \pi \in \text{PATHS}(g)\}$$

$$F(g)=\bigcup_{\pi \in \text{PATHS}(g)} F(\pi)$$

$$RT(g)=\bigcup_{\pi \in \text{PATHS}(g)} RT(\pi)$$

$$FT(g)=\bigcup_{\pi \in \text{PATHS}(g)} FT(\pi)$$

PROOF: Straightforward. □

Analogously, the simulation-like equivalences can be characterized by means of simulation relations between the nodes of two process graphs, rather than between process graphs themselves. Below this is done for bisimulation equivalence.

DEFINITION: Let $g,h \in \mathbf{G}$. A *bisimulation* between g and h is a binary relation $R \subseteq \text{NODES}(g) \times \text{NODES}(h)$, satisfying:

1. $\text{ROOT}(g) R \,\text{ROOT}(h)$.
2. If sRt and $(s,a,s') \in \text{EDGES}(g)$, then there is an edge $(t,a,t') \in \text{EDGES}(h)$ such that $s'Rt'$.
3. If sRt and $(t,a,t') \in \text{EDGES}(h)$, then there is an edge $(s,a,s') \in \text{EDGES}(g)$ such that $s'Rt'$.

This definition is illustrated in Figure 8. Now it follows easily that two graphs

g and h are bisimilar iff there exists a bisimulation between them.

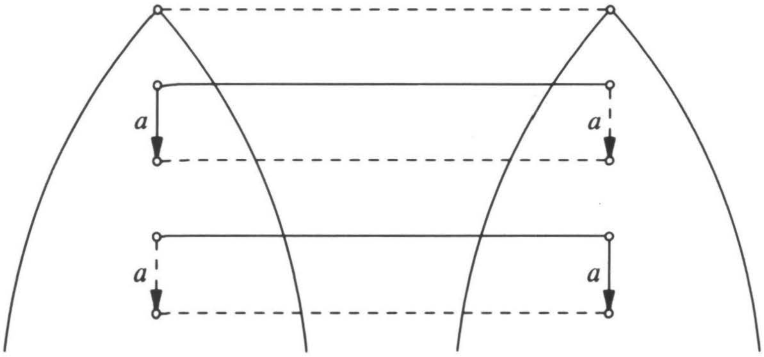


FIGURE 8

Proposition 2.5 yields a technique for deciding that two process graphs are ready trace equivalent, c.q. failure trace equivalent, without calculating their entire ready trace or failure trace set.

Let $g,h \in \mathbf{G}$, $\pi: s_0 \xrightarrow{a_1} s_1 \xrightarrow{a_2} \cdots \xrightarrow{a_n} s_n \in \mathrm{PATHS}$ and $\pi': t_0 \xrightarrow{a_1} t_1 \xrightarrow{a_2} \cdots \xrightarrow{a_n} t_m \in \mathrm{PATHS}$. Path π' is a *failure trace augmentation* of π, notation $\pi \leqslant_{FT} \pi'$, if $FT(\pi) \subseteq FT(\pi')$. This is the case exactly when $n = m$ and $I(t_i) \subseteq I(s_i)$ for $i = 1,...,n$. Write $\pi =_{FT} \pi'$ for $\pi \leqslant_{FT} \pi' \wedge \pi' \leqslant_{FT} \pi$. It follows that $\pi =_{FT} \pi' \Leftrightarrow FT(\pi) = FT(\pi') \Leftrightarrow RT(\pi) = RT(\pi')$. From this the following can be concluded.

COROLLARY 2.5: *Two process graphs $g,h \in \mathbf{G}$ are ready trace equivalent iff*

- *for any path $\pi \in \mathrm{PATHS}(g)$ in g there is a $\pi' \in \mathrm{PATHS}(h)$ such that $\pi =_{FT} \pi'$*
- *and for any path $\pi \in \mathrm{PATHS}(g)$ in h there is a $\pi' \in \mathrm{PATHS}(g)$ such that $\pi =_{FT} \pi'$.*

They are failure trace equivalent iff

- *for any path $\pi \in \mathrm{PATHS}(g)$ in g there is a $\pi' \in \mathrm{PATHS}(h)$ such that $\pi \leqslant_{FT} \pi'$*
- *and for any path $\pi \in \mathrm{PATHS}(g)$ in h there is a $\pi' \in \mathrm{PATHS}(g)$ such that $\pi \leqslant_{FT} \pi'$.*

If g and h are moreover without infinite paths, then it suffices to check the requirements above for maximal paths.

2.6. Drawing process graphs.

DEFINITION: Let $g,h \in \mathbf{G}$. A *graph isomorphism* between g and h is a bijective function $f : \mathrm{NODES}(g) \rightarrow \mathrm{NODES}(h)$ satisfying

- $f(\mathrm{ROOT}(g)) = \mathrm{ROOT}(g)$ and
- $(s,a,t) \in \mathrm{EDGES}(g) \Leftrightarrow (f(s),a,f(t)) \in \mathrm{EDGES}(h)$.

Graphs g and h are *isomorphic*, notation $g \cong h$, if there exists a graph isomorphism between them.

In this case g and h differ only in the identity of their nodes. Remark that graph isomorphism is an equivalence on $\mathbb{G}$.

PROPOSITION 2.6: *For $g,h \in \mathbb{G}$, $g \cong h$ iff there exists a bisimulation R between g and h, satisfying*
4. If sRt and uRv then $s=u \Leftrightarrow t=v$.

PROOF: Suppose $g \cong h$. Let f :NODES $(g) \rightarrow$ NODES (h) be a graph isomorphism. Define $R \subseteq$ NODES $(g) \times$ NODES (h) by sRt iff $f(s)=t$. Then it is routine to check that R satisfies clauses 1, 2, 3 and 4. Now suppose R is a bisimulation between g and h, satisfying 4. Define f :NODES $(g) \rightarrow$ NODES (h) by $f(s)=t$ iff sRt. Since g is connected it follows from the definition of a bisimulation that for each s such a t can be found. Furthermore direction "$\Rightarrow$" of clause 4 implies that $f(s)$ is uniquely determined. Hence f is well-defined. Now direction "$\Leftarrow$" of clause 4 implies that f is injective. From the connectedness of h if follows that f is also surjective, and hence a bijection. Finally clauses 1, 2 and 3 imply that f is a graph isomorphism. □

COROLLARY: *If $g \cong h$ then g and h are equivalent according to all semantic equivalences of Section 1.*

Finitely branching connected process graphs can be pictured by using open dots (∘) to denote nodes, and labelled arrows to denote edges, as can be seen in Subsection 2.8. There is no need to mark the root of such a process graph if it can be recognized as the unique node without incoming edges, as is the case in all my examples. These pictures determine process graphs only up to graph isomorphism, but usually this suffices since it is virtually never needed to distinguish between isomorphic graphs.

2.7. Embedding labelled transition systems in $\mathbb{G}$. Let $\mathbf{A}$ be an arbitrary labelled transition system and let $p \in \mathbf{A}$. The *canonical graph* $G(p)$ of p is defined as follows:

- NODES $(G(p)) = \{q \in \mathbf{A} \mid \exists \sigma \in A^*: p \xrightarrow{\sigma} q\}$,
- ROOT $(G(p)) = p \in$ NODES $(G(p))$,
- and $(q,a,r) \in$ EDGES $(G(p))$ iff $q,r \in$ NODES $(G(p))$ and $q \xrightarrow{a} r$.

Of course $G(p) \in \mathbb{G}$. This means G is a function from $\mathbf{A}$ to $\mathbb{G}$.

PROPOSITION 2.7: *$G: \mathbf{A} \rightarrow \mathbb{G}$ is an injective function, satisfying, for $a \in Act$: $G(p) \xrightarrow{a} G(q) \Leftrightarrow p \xrightarrow{a} q$.*
PROOF: Trivial. □
COROLLARY: *For $p \in \mathbf{A}$ and $O \in \{T, CT, F, R, FT, RT, S, CS, RS, PF, 2S, B\}$, $O(G(p)) = O(p)$.*

Proposition 2.7 says that G is an *embedding* of $\mathbb{A}$ in $\mathbb{G}$. It implies that any labelled transition system over *Act* can be represented as a subclass $G(\mathbb{A})=\{G(p)\in\mathbb{G} \mid p\in\mathbb{A}\}$ of $\mathbb{G}$.

Since $\mathbb{G}$ is also a labelled transition system, G can be applied to $\mathbb{G}$ itself. The following proposition says that the function $G:\mathbb{G}\rightarrow\mathbb{G}$ leaves its arguments intact up to graph isomorphism.

PROPOSITION 2.8: *For* $g\in\mathbb{G}$, $G(g)\cong g$.
PROOF: Remark that $\text{NODES}(G(g))=\{g_s \mid s\in\text{NODES}(g)\}$. Now the function $f:\text{NODES}(G(g))\rightarrow\text{NODES}(g)$ defined by $f(g_s)=s$ is a graph isomorphism. □

2.8. Counterexamples. In this subsection a number of examples will be presented, showing that on $\mathbb{G}$ all semantic notions mentioned in Theorem 2.2 are different and $\mathbb{S} \preccurlyeq \mathfrak{T}$ holds only if this follows from that theorem. Moreover, apart from the examples needed to show the difference between semantics that are connected by a dashed arrow in Figure 7, all examples will use finite processes only. Thus it follows that neither the ordering of Theorem 2.1 nor the ordering of Theorem 2.2 can be further expanded. Let $\mathbb{H}$ be the set of finite connected process graphs. Here a process graph g is *finite* if $\text{PATHS}(g)$ is finite. Finite graphs are acyclic and have only finitely many nodes and edges. They represent finite processes.

THEOREM 2.9: *Let* $\mathbb{S}$ *and* $\mathfrak{T}$ *be semantics on* $\mathbb{H}$ *from the series T, CT, F, R, FT, RT, S, CS, RS, PF, 2S, B. Then* $\mathbb{S}\preccurlyeq\mathfrak{T}$ *only if this follows from Theorem 2.1. (and the fact that* $\preccurlyeq$ *is a partial order).*

PROOF: The following counterexamples provide for any statement $\mathbb{S}\preccurlyeq\mathfrak{T}$, not following from Theorem 2.1 and the fact that $\preccurlyeq$ is a partial order, two finite connected process graphs that are identified in $\mathfrak{T}$, but distinguished in $\mathbb{S}$.

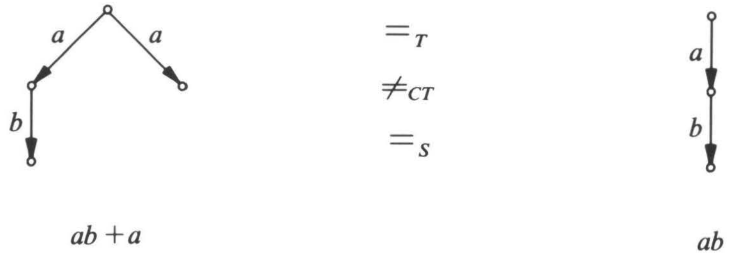


FIGURE 9

1. *T*$\not\preccurlyeq$*CT*. For the graphs of Figure 9, $T(\mathit{left})=T(\mathit{right})=\{\epsilon, a, ab\}$, whereas $CT(\mathit{left})\neq CT(\mathit{right})$ (since $a\in CT(\mathit{left})-CT(\mathit{right})$). Hence they are identified in trace semantics but distinguished in completed trace semantics.

Furthermore the two graphs are simulation equivalent (the construction of the two simulations is left to the reader). Since $\preccurlyeq$ is a partial order, the same example shows that $\mathcal{S} \not\preccurlyeq \mathcal{T}$ for $\mathcal{S} \in \{CT, CS, F, R, FT, RT, RS, PF, 2S, B\}$ and $\mathcal{T} \in \{T, S\}$.

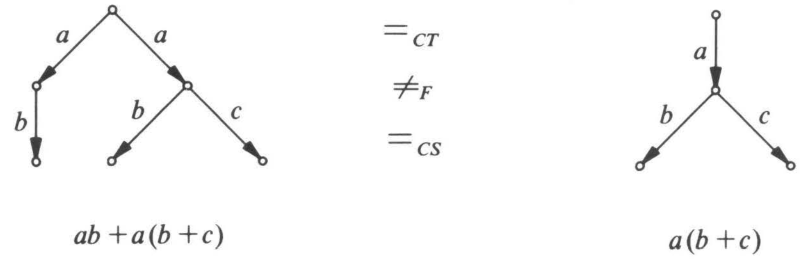
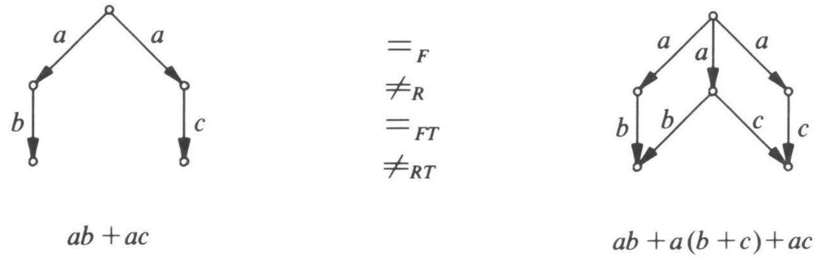


FIGURE 10

2. $CT \not\succcurlyeq F$. For the graphs of Figure 10, $CT(left) = CT(right) = \{ab, ac\}$, whereas $F(left) \neq F(right)$ (since $<a, \{b\}> \in F(left) - F(right)$). Hence they are identified in completed trace semantics but distinguished in failure semantics. Furthermore the two graphs are completed simulation equivalent (the construction of the two completed simulations is again left to the reader). Since $\preccurlyeq$ is a partial order, the same example shows that $\mathcal{S} \not\preccurlyeq \mathcal{T}$ for $\mathcal{S} \in \{F, R, FT, RT, RS, PF, 2S, B\}$ and $\mathcal{T} \in \{CT, CS\}$.



fi

FIGURE 11

3. $FT \not\succcurlyeq R$. For the graphs of Figure 11, $FT(left) = FT(right)$, whereas $R(left) \neq R(right)$. The first statement follows from Corollary 2.5, since the new maximal paths at the right-hand side are both failure trace augmented by the two maximal paths both sides have in common. The second one follows since $<a, \{b,c\}> \in R(right) - R(left)$. Hence these processes are identified in failure trace semantics but distinguished in readiness semantics. Since $\preccurlyeq$ is a

partial order, the same example shows that $\mathcal{S}\not\succcurlyeq\mathcal{T}$ for any $\mathcal{S}\preccurlyeq FT$ and $\mathcal{T}\succcurlyeq R$, so in particular $F\not\succcurlyeq R$ and $FT\not\succcurlyeq RT$.

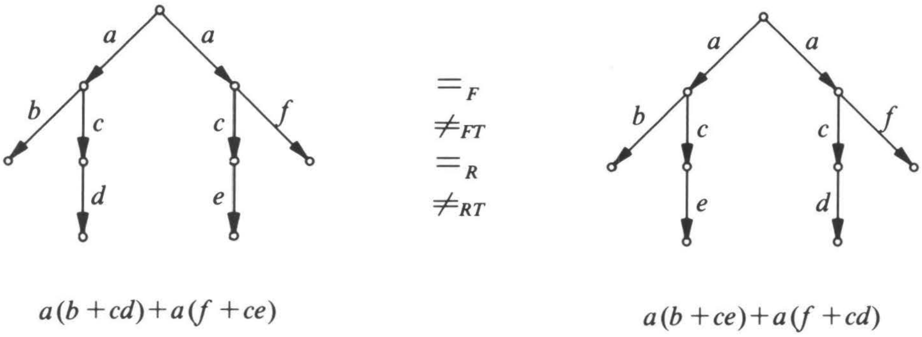


FIGURE 12

4. $R\not\succcurlyeq FT$. For the graphs of Figure 12, $R(left)=R(right)$, whereas $FT(left)\neq FT(right)$. The first statement follows since in the second graph only 4 ready pairs swopped places. The second one follows since $a\{b\}ce\in FT(left)-FT(right)$. Hence these processes are identified in readiness semantics but distinguished in failure trace semantics. Since $\preccurlyeq$ is a partial order, the same example shows that $\mathcal{S}\not\succcurlyeq\mathcal{T}$ for any $\mathcal{S}\preccurlyeq R$ and $\mathcal{T}\succcurlyeq FT$, so in particular $F\not\succcurlyeq FT$ and $R\not\succcurlyeq RT$. Since $PF(left)\neq PF(right)$ this example does not show that $PF\not\succcurlyeq FT$. It it left as an exercise to the reader to adapt the example so that also that is established.

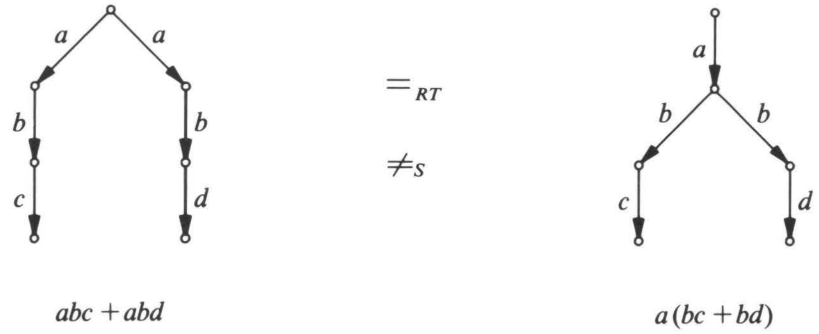


FIGURE 13

5. $RT\not\succcurlyeq S$. For the graphs of Figure 13, $RT(left)=RT(right)$, whereas $S(left)\neq S(right)$. The first statement follows immediately from Corollary 2.5.

The second one follows since $a(bcT \wedge bdT) \in S(right) - S(left)$. Hence these processes are identified in ready trace semantics but distinguished in simulation semantics. Since $\preccurlyeq$ is a partial order, the same example shows that $\mathfrak{S} \not\succcurlyeq \mathfrak{T}$ for any $\mathfrak{S} \preccurlyeq RT$ and $\mathfrak{T} \succcurlyeq S$, so in particular $T \not\succcurlyeq S$, $CT \not\succcurlyeq CS$ and $RT \not\succcurlyeq RS$.

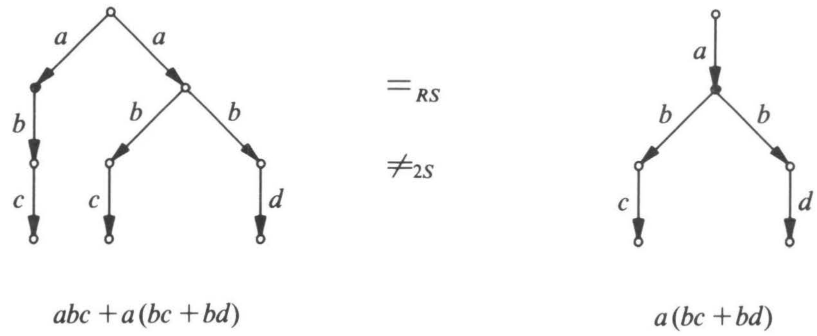


FIGURE 14

6. $RS \not\succcurlyeq 2S$. The graphs of Figure 14 are ready simulation equivalent, but not 2-nested simulation equivalent. There exists exactly one simulation from *right* by *left*, namely the one mapping *right* on the right-hand side of *left*, and this simulation is a ready simulation as well as a 2-nested simulation. There also exists exactly one simulation from *left* by *right*, which maps the black node on the *left* on the black node on the *right*. This simulation is a ready simulation (related nodes have the same menu of initial actions) but not a 2-nested simulation (the two subgraphs originating from the two black nodes are not simulation equivalent). Hence $RS \not\succcurlyeq 2S$. Furthermore $PF(left) \neq PF(right)$, since $<a, \{\epsilon, b, bc\}> \in PF(left) - PF(right)$. Hence $\mathfrak{S} \not\succcurlyeq PF$ for any $\mathfrak{S} \preccurlyeq RS$.

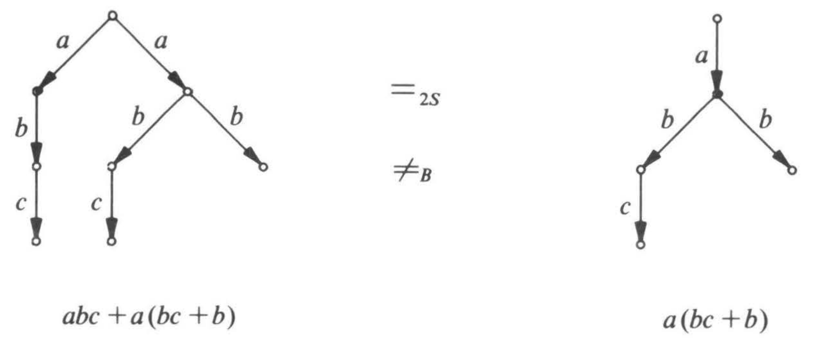


FIGURE 15

7. $2S \not\succcurlyeq B$. The graphs of Figure 15 are 2-nested simulation equivalent, but not bisimulation equivalent. There now exists 2-nested simulations in both directions since the two subgraphs originating from the two black nodes are simulation equivalent. However, $a\neg b\neg cT \in HML(\mathit{left}) - HML(\mathit{right})$. □

THEOREM 2.10: *Let $\mathcal{S}$ and $\mathcal{T}$ be semantics on* **G** *mentioned in Subsection 2.2. Then $\mathcal{S} \preccurlyeq \mathcal{T}$ only if this follows from Theorem 2.2. (and the fact that $\preccurlyeq$ is a partial order).*

PROOF: The following counterexamples provide for any statement $\mathcal{S} \preccurlyeq \mathcal{T}$, not following from Theorem 2.2 and the fact that $\preccurlyeq$ is a partial order, two connected process graphs that are identified in $\mathcal{T}$, but distinguished in $\mathcal{S}$.

8. $B^* \not\succcurlyeq T^\omega$. The graphs of Figure 4 in Chapter 3 are finitary bisimulation equivalent (as follows straightforward with induction) but not infinitary trace equivalent (since only the graph at the right has an infinite trace). Since $\preccurlyeq$ is a partial order it follows that $\mathcal{S} \not\succcurlyeq \mathcal{T}$ for $\mathcal{S} \preccurlyeq B^*$ and $\mathcal{T} \succcurlyeq T^\omega$.

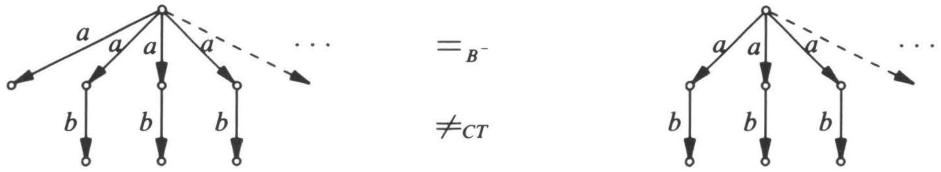


FIGURE 16

9. $B^- \not\succcurlyeq CT$. For the graphs of Figure 16, $HML(\mathit{left}) = HML(\mathit{right})$, whereas $CT(\mathit{left}) \neq CT(\mathit{right})$. The first statement follows since by means of HML-formulas one can only say that a *finite* set of actions can not take place in a certain state. The second one follows since $a \in CT(\mathit{left}) - CT(\mathit{right})$. Since $\preccurlyeq$ is a partial order it follows that $\mathcal{S} \not\succcurlyeq \mathcal{T}$ for $\mathcal{S} \preccurlyeq B^-$ and $\mathcal{T} \succcurlyeq CT$. □

One could say that a semantics $\mathcal{S}$ *respects deadlock behaviour* iff $\mathcal{S} \succcurlyeq CT$. The example above then shows that non of the semantics on the left in Figure 7 respects deadlock behaviour; only the left-hand process of Figure 16 can deadlock after an a-move.

3. Complete axiomatizations

3.1. A language for finite, concrete, sequential processes. Consider the following basic CCS- and CSP-like language BCCSP for finite, concrete, sequential processes over a finite alphabet Act:

inaction: 0 (called *nil* or *stop*) is a constant, representing a process that refuses to do any action.

action: a is a unary operator for any action $a \in Act$. The expression ap represents a process, starting with an a-action and proceeding with p.

choice: $+$ is a binary operator. $p+q$ represents a process, first being involved in a choice between its summands p and q, and then proceeding as the chosen process.

The set $\mathbb{P}$ of (closed) *process expressions* or terms over this language is defined as usual:

- $0 \in \mathbb{P}$,
- $ap \in \mathbb{P}$ for any $a \in Act$ and $p \in \mathbb{P}$,
- $p+q \in \mathbb{P}$ for any $p,q \in \mathbb{P}$.

Subterms $a0$ may be abbreviated by a.

On $\mathbb{P}$ action relations $\xrightarrow{a}$ for $a \in Act$ are defined as the predicates on $\mathbb{P}$ generated by the *action rules* of Table 1. Here a ranges over Act and p and q over $\mathbb{P}$.

$$\boxed{ap \xrightarrow{a} p \qquad \frac{p \xrightarrow{a} p'}{p+q \xrightarrow{a} p'} \qquad \frac{q \xrightarrow{a} q'}{p+q \xrightarrow{a} q'}}$$

Table 1

Now all semantic equivalences of Section 1 are well-defined on $\mathbb{P}$, and for each of the semantics it is determined when two process expressions denote the same process.

3.2. Axioms. In Table 2, complete axiomatizations can be found for ten of the twelve semantics of this chapter that differ on BSSCP. Axioms for 2-nested simulation and possible-futures semantics are more cumbersome, and the corresponding testing notions are less plausible. Therefore they have been omitted. In order to formulate the axioms, variables have to be added to the language as usual. In the axioms they are supposed to be universally quantified. Most of the axioms are axiom schemes, in the sense that there is one axiom for each substitution of actions from *Act* for the parameters a,b,c. Some of the axioms are conditional equations, using an auxiliary operator I. Thus provability is defined according to the standards of either first-order logic with equality or conditional equational logic. I is a unary operator that calculates the set of initial actions of a process expression, coded as a process expression again.

THEOREM 3.1: *For each of the semantics $O \in \{T, S, CT, CS, F, R, FT, RT, RS, B\}$ two process expressions $p,q \in \mathbb{P}$ are O-equivalent iff they can be proved equal from the axioms marked with '+' in the column for O in Table 2. The axioms marked with 'v' are valid in O-semantics but not needed for the proof.*

	B	RS	RT	FT	R	F	CS	CT	S	T
$x+y = y+x$	+	+	+	+	+	+	+	+	+	+
$(x+y)+z = x+(y+z)$	+	+	+	+	+	+	+	+	+	+
$x+x = x$	+	+	+	+	+	+	+	+	+	+
$x+0 = x$	+	+	+	+	+	+	+	+	+	+
$I(x) = I(y) \Rightarrow a(x+y) = ax+a(x+y)$		+	v	v	v	v	v	v	v	v
$I(x) = I(y) \Rightarrow ax+ay = a(x+y)$			+	+	v	v		v		v
$ax+ay = ax+ay+a(x+y)$				+		v		v		v
$a(bx+u)+a(by+v) = a(bx+by+u)+a(bx+by+v)$					+	+		v		v
$ax+a(y+z) = ax+a(x+y)+a(y+z)$						+		v		v
$a(bx+u+y) = a(bx+u)+a(bx+u+y)$							+	v	v	v
$a(bx+u)+a(cy+v) = a(bx+cy+u+v)$								+		v
$a(x+y) = ax+a(x+y)$									+	v
$ax+ay = a(x+y)$										+
$I(0) = 0$	+	+	+	+	+	+	+	+	+	+
$I(ax) = a0$	+	+	+	+	+	+	+	+	+	+
$I(x+y) = I(x)+I(y)$	+	+	+	+	+	+	+	+	+	+

TABLE 2

PROOF: For *F, R* and *B* the proof is given in BERGSTRA, KLOP & OLDEROG [24] by means of *graph transformations*. A similar proof for RT can be found in BAETEN, BERGSTRA & KLOP [10]. For the remaining semantics a proof can be given along the same lines. □

Concluding remarks

In this chapter various semantic equivalences for concrete sequential processes are defined, motivated, compared and axiomatized. Of course many more equivalences can be given then the ones presented here. The reason for selecting just these, is that they can be motivated rather nicely and/or play a role in the literature on semantic equivalences. In Abramsky & Vickers [2] the observations which underly many of the semantics in this chapter are placed in a uniform algebraic framework, and some general completeness criteria are stated and proved.

It is left for a future occasion to give (and apply) criteria for selecting between these equivalences for particular applications (such as the complexity of deciding if two finite-state processes are equivalent, or the range of useful operators for which they are congruences). The work in this direction reported so far, includes [28] and [68].

An interesting topic is the generalization of this work to a setting with silent moves and/or with parallelism. In Chapter III the generalization of bisimulation semantics to a setting with silent steps is considered; in Chapters IV-VII bisimulation and trace semantics will be considered in a setting with parallelism. In both cases there turn out to be many interesting variations. Generalizing the entire spectrum to a setting with both silent actions and parallelism remains as of yet to be done. However, in many papers parts of a classification can be found already (see for instance [107]).

A generalization to preorders, instead of equivalences, can be obtained by replacing conditions like $O(p)=O(q)$ by $O(p)\subseteq O(q)$. Since preorders are often useful for verification purposes, it seems to be worthwhile to have to classify them as well.

Furthermore it would be interesting to give explicit representations of the equivalences, by representing processes as sets of observations instead of equivalence classes of process graphs, and defining operators like action prefixing and choice directly on these representations, as has been done for failure semantics in [33] and for readiness semantics in [102].

Chapter II

Modular Specifications in Process Algebra

With Curious Queues

Rob van Glabbeek and Frits Vaandrager

In recent years a wide variety of process algebras has been proposed in the literature. Often these process algebras are closely related: they can be viewed as homomorphic images, submodels or restrictions of each other. The aim of this chapter is to show how the semantical reality, consisting of a large number of closely related process algebras, can be reflected, and even used, on the level of algebraic specifications and in process verifications. This is done by means of the notion of a module. The simplest modules are building blocks of operators and axioms, each block describing a feature of concurrency in a certain semantical setting. These modules can then be combined by means of a union operator $+$, an export operator $\Box$, allowing to forget some operators in a module, an operator H, changing semantics by taking homomorphic images, and an operator S which takes subalgebras. These operators enable us to combine modules in a subtle way, when the direct combination would be inconsistent. We show how auxiliary process algebra operators can be hidden when this is needed. Moreover it is demonstrated how new process combinators can be defined in terms of the more elementary ones in a clean way. As an illustration of our approach, a methodology is presented that can be used to specify FIFO-queues, and that facilitates verification of concurrent systems containing these queues.

Notes. This chapter appeared as Report CS-R8821, Centrum voor Wiskunde en Informatica, Amsterdam 1988, an extended abstract of which has been published in Algebraic Methods: Theory, Tools and Applications, LNCS 394, Springer-Verlag, pp. 465-506. Apart from Sections 4 and 5 it moreover appeared in the Ph.D. Thesis of Frits Vaandrager [122].

The research of the authors was supported by ESPRIT project no. 432, An Integrated Formal Approach to Industrial Software Development (METEOR). The research of the second author was also supported by RACE project no. 1046, Specification and Programming Environment for Communication Software (SPECS).

TABLE OF CONTENTS

INTRODUCTION

During the last decade, a lot of research has been done on *process a* branch of theoretical computer science concerned with the modelling current systems as elements of an algebra. Besides the Calculus of Comm cating Systems (CCS) of MILNER [92,95], several related formalisms have bee developed, such as the theory of Communicating Sequential Processes (CSP) of HOARE [76], the MEIJE calculus of AUSTRY & BOUDOL [6] and the Algebra of Communicating Processes (ACP) of BERGSTRA & KLOP [19,20,22].

When work on process algebra started, many people hoped that it would be possible to come up, eventually, with the 'ultimate' process algebra, leading to a 'Church thesis' for concurrent computation. This process algebra, one imagined, should contain only a few fundamental operators and it should be suited to model all concurrent computational processes. Moreover there should be a calculus for this model making it possible to prove the identity of processes algebraically, thus proving correctness of implementations with respect to specifications. As far as we know, the ultimate process algebra has not yet been found, but we will not exclude that it will be discovered in the near future.

Two things however, have become clear in the meantime: (1) it is doubtful whether algebraic system verification, as envisaged in [92], will be possible in this model, and (2) even if the ultimate process algebra exists, this certainly does not mean that all other process algebras are no longer interesting. We elaborate on this below.

A central idea in process algebra is that two processes which cannot be distinguished by observation should preferably be identified: the process semantics should be fully abstract with respect to some notion of testing (see [43,92] and the first chapter of this thesis). This means that the choice of a suitable process algebra may depend on the tools an environment has to distinguish between certain processes. In different applications the tools of the environment may be different, and therefore different applications may require different process algebras. A large number of process semantics are not fully abstract with respect to any (reasonable) notion of testing (bisimulation semantics and partial order semantics, for instance). Still these semantics can be very interesting because they have simple definitions or correspond to some strong operational intuition. Our hypothetical ultimate process algebra will make very few identifications, because it should be resistant against all forms of testing. Therefore not many algebraic laws will be valid in this model and algebraic system verification will presumably not be possible (specification and implementation correspond to different processes in the model).

Another factor which plays a role has to do with the operators of process algebras. For theoretical purposes it is in general desirable to work with a single, small set of fundamental operators. We doubt however that a unique optimal and minimal collection exists. What is optimal depends on the type of results one likes to prove. This becomes even more clear if we look towards practical applications. Some operators in process algebra can be used for a wide range of applications, but we agree with JIFENG & HOARE [77] that we may have to accept that each application will require derivation of specialised laws (and operators) to control its complexity.

Many people are embarrassed by the multitude of process algebras occurring in the literature. They should be aware of the fact that there are close relationships between the various process algebras: often one process algebra can be viewed as a homomorphic image, subalgebra or restriction of another one. The aim of this chapter is to show how the semantical reality, consisting of a large number of closely related process algebras, can be reflected, and even used, on the level of algebraic specifications and in process verifications.

This chapter is about process algebras, their mutual relationships, and strategies to prove that a formula is valid in a process algebra. Still, we do not present any particular process algebra here. In the other chapters of this thesis several process algebras are discussed. However we neither define all the operations we use in this chapter nor all the semantical notions that will be considered here. In this chapter we only define classes of models of process modules. One reason for doing this is that a detailed description of all particular process algebras we use would make this thesis too long. Another reason is that there is often no clear argument for selecting a particular process algebra. In such situations we are interested in assertions saying that a formula is valid in all algebras satisfying a certain theory. A number of times we need results stating that some formulas *cannot* be proven from a certain module. A standard way to prove this is to give a model of the module where the formulas are not true. For this reason we will often refer to particular process algebras which have been described elsewhere in the literature.

The discussion of this chapter takes place in the setting of ACP. We think however that the results can be carried over to CCS, CSP, MEIJE, or any other process algebra formalism.

Modularisation.

The creation of an algebraic framework suitable to deal with realistic applications, gives rise to the construction of building blocks, or modules, of operators and axioms, each block describing a feature of concurrency in a certain semantical setting. These modules can then be combined by means of a module combinator $+$. We give some examples:

i) A kernel module, that expresses some basic features of concurrent processes, is the module ACP. For a lot of applications however, ACP does not provide enough operators. Often the use of *renaming operators* makes specifications shorter and more comprehensible. These renaming operators can be defined in a separate module RN. Now the module ACP+RN combines the specification and verification power of modules ACP and RN.

ii) The axioms of module ACP correspond to the semantical notion of bisimulation. For some applications bisimulation semantics does not make enough identifications. In these cases one would like to deal with processes on the level of, for example, failure semantics. Now one can define a module F, corresponding to the identifications made in failure semantics on top of the identifications of bisimulation semantics. The module ACP+F then corresponds to the failure model.

Once a number of modules have been defined, they can be combined in a lot of ways. Some combinations are interesting (for example the module ACP+RN+F), for other combinations no interesting applications exist (the module RN+F). Didactical aspects aside, a major advantage of the modular approach is that results which have been proved from a module M, can also be proved from a module M+N. This means that process verifications become *reusable*.

It turns out that certain pairs of modules are incompatible in a very strong sense: with the combination of two modules strange and counter-intuitive identities can be derived. In BAETEN, BERGSTRA & KLOP [10], for example, it is shown that the combination of failure semantics and the priority operator is inconsistent in the sense that an identity can be derived which says that a particular process that can do a *b*-action after it has done an *a*-action, equals a process that cannot do this. Another example can be found in BERGSTRA, KLOP & OLDEROG [23], where it is pointed out that the combination of failure semantics and Koomen's Fair Abstraction Rule (KFAR) is inconsistent.

In the first section of this chapter we present, besides the combinator +, some other operators on modules. We discuss an export operator □, allowing to forget some operators in a module, an operator *H*, changing semantics by taking homomorphic images, and an operator *S* which takes subalgebras. These operators enable us to combine modules in a subtle way, when the direct combination would be inconsistent. In Section 2 we describe a large number of process modules which play a role in the ACP framework. Section 3 contains two examples of applications of the new module operators in process algebra:

1. The axiom system ACP contains auxiliary operators $\lfloor\!\lfloor$ and $|$ (left-merge and communication-merge) which drastically simplify computations and have some desirable 'metamathematical' consequences (finite axiomatisability[1]; greater suitability for term rewriting analysis). These auxiliary operators can be defined in a large class of process algebras. However, it turns out that in a setting with the silent step τ the left-merge cannot be added consistently to all algebras (for instance not to the usual variants of failure semantics). Now one may think that this result means that someone who is doing failure semantics with τ's cannot profit from the nice properties of the left-merge. However, we will show in this chapter that use of the module approach makes it possible to do failure semantics with τ's but still benefit from the left-merge in verifications. The idea is that verifications take place on two levels: the level of bisimulation semantics where the left-merge can be used, and a level of for instance failure semantics, where no left-merge is present. The failure model can be

[1] Recently, FARON MOLLER [97] from Edinburgh showed that in bisimulation semantics the merge operator cannot be finitely axiomatised without auxiliary operators.

obtained from the bisimulation model by removing the auxiliary operators and taking a homomorphic image. Now we use the observation that certain formulas (the 'positive' ones without auxiliary operators) are preserved under this procedure. A consequence of this application is that even if bisimulation semantics is not considered to be an appropriate process semantics (since it is not fully abstract with respect to any reasonable notion of testing), it still can be useful as an expedient for proving formulas in failure semantics.

2. As already pointed out above, one would like to have, from a theoretical point of view, as few operators or combinators as possible. On the other hand, when dealing with applications, it is often very rewarding to introduce new operators. This paradox can be resolved if the new operators are definable in terms of the more elementary ones. In that case the new operators can be considered as notations which are useful, but do not complicate the underlying theory. A problem with defining operators in terms of other operators is that often auxiliary atomic actions are needed in the definition. These auxiliary actions can then not be used in any other place, because that would disturb the intended semantics of the operator. In the laws that can be derived for the defined operator, the auxiliary actions occur prominently. These 'side effects' are often quite unpleasant. One may think that side effects are unavoidable and that someone who really does not like them should define new operators directly in the algebras (even though this is in conflict with the desire to have as few operators as possible). However, we will show that the module approach can be used to solve also this problem: with the restriction operator we remove the auxiliary actions from the signature and then we apply the subalgebra operator in order to 'move' to algebras where the auxiliary actions are not present at all.

The concept of hiding auxiliary operators in a module in some formal way is quite familiar in the literature (see BERGSTRA, HEERING & KLINT [17] for example), but the use of module operators H and S, and their application in combining modules that would be incompatible otherwise, is, as far as we know, new. The H and S operations are in spirit related to the **abstract** operation of SANNELLA & WIRSING [114] and SANNELLA & TARLECKI [113], which also extends the model class of a module.

In previous papers on ACP, the underlying logic used in process verifications was not made explicit. The reason for this was that a long definition of the logic would distract the reader's attention from the more essential parts of the paper. It was felt that filling in the details of the logic would not be too difficult and that moreover different options were equivalent. In this chapter we generalise the classical notion of a formal proof of a formula from a theory to the notion of a formal proof of a formula from a module. The definition of this last notion is parametrised by the underlying logic. What is provable from a module really depends on the logic that is used, and this makes it necessary to consider in more detail the issue of logics. In an appendix we present three alternatives: (1) Equational logic. This logic is suited for dealing with finite processes, but not strong enough for handling infinite processes; (2) Infinitary conditional equational logic. This is the logic used in most process verifications in the ACP framework until now; (3) First order logic with equality.

Our investigations into the precise nature of the calculi used in process algebra, led us to alternative formulations of some of the proof principles in ACP which fit better in our formal setup. We present a reformulation of the Recursive Specification Principle (RSP) and also an alphabet operator which returns a process instead of a set of actions.

Queues.
As an illustration of the techniques developed in Sections 1 to 3, we present in Section 4 an algebraic treatment of FIFO-queues. FIFO-queues play an important role in the description of languages with asynchronous message passing, the modelling of communication channels occurring in computer networks and the implementation of languages with synchronous communication. We show how the chaining operator can be used to give short specifications of various (faulty) queues and simple proofs of numerous identities, for example of the fact that the chaining of a queue with unbounded capacity and a one datum buffer is again a queue.

We give an example of an identity that holds intuitively (there is no experiment that distinguishes between the two processes) but is not valid in bisimulation semantics. We use the machinery developed in Section 1-3 to extend the axiom system in a neat way (avoiding inconsistencies) so that we can prove the processes identical.

A protocol verification.
The usefulness of the proof technique for queues is illustrated in Section 5, where a modular verification is presented of a concurrent alternating bit protocol. This verification takes 4 pages (or 5 if the proof of the standard facts about the queues is included) and is thereby considerably shorter than the proof of similar protocols in papers by KOYMANS & MULDER [81] and LARSEN & MILNER [85] (15 and 11 pages respectively). The verification shows that the protocol is correct if the channels behave as faulty FIFO-queues with unbounded capacity. However, a minor change in the proof is enough to show that the protocol also works if the channels behave as *n*-buffers, faulty *n*-buffers, etc. In our view the basic merit of our way of dealing with queues is that it becomes possible to use inductive arguments when dealing with the length of queues in protocol systems.

1. Module logic

In this chapter, as in many other papers about process algebra, we use formal calculi to prove statements about concurrent systems. In this section we answer the following questions:

- Which kind of calculi do we use?
- What do we understand by a proof?

In the next sections we will apply this general setup to the setting of concurrent systems.

1.1. Statements about concurrent systems. In many theories of concurrency it is common practice to represent processes - the behaviours of concurrent systems - as elements in an *algebra*. This is a mathematical domain, on which some operators and predicates are defined. Algebras, which are suitable for the representation of processes are called *process algebras*. Thus a statement about the behaviour of concurrent systems can be regarded as a statement about the elements of a certain process algebra. Such a statement can be represented by a formula in a suitable language which is interpreted in this process algebra. Sometimes we consider several process algebras at the same time and want to formulate a statement about concurrent processes without choosing one of these algebras. In this case we represent the statement by a formula in a suitable language which has an interpretation in all these process algebras. Hence we are interested in assertions of the form: 'Formula ϕ holds in the process algebra $\mathcal{A}$', notation $\mathcal{A} \models \phi$, or 'Formula ϕ holds in the class of process algebras $\mathcal{C}$', notation $\mathcal{C} \models \phi$. Now we can formulate the goal that is pursued in the present section: to propose a method for proving assertions $\mathcal{A} \models \phi$, or $\mathcal{C} \models \phi$.

1.2. Proving formulas from theories. Classical logic gave us the notion of a formal proof of a formula ϕ from a theory T. Here a theory is a set of formulas. We write $T \vdash \phi$ if such a proof exists. The use of this notion is revealed by the following soundness theorem: *If $T \vdash \phi$ then ϕ holds in all algebras satisfying T.* Here an algebra $\mathcal{A}$ satisfies T, notation $\mathcal{A} \models T$, if all formulas of T hold in this algebra. Thus if we want to prove $\mathcal{A} \models \phi$ it suffices to prove $T \vdash \phi$ and $\mathcal{A} \models T$ for a suitable theory T. Likewise, if we want to prove $\mathcal{C} \models \phi$, with $\mathcal{C}$ a class of algebras, it suffices to prove $T \vdash \phi$ and $\mathcal{C} \models T$.

At first sight the method of proving $\mathcal{A} \models \phi$ by means of a formal proof of ϕ out of T seems very inefficient. Instead of verifying $\mathcal{A} \models \phi$, one has to verify $\mathcal{A} \models \psi$ for all $\psi \in T$, and moreover the formal proof has to be constructed. However, there are two circumstances in which this method *is* efficient, and in most applications both of them apply. First of all it might be the case that ϕ is more complicated than the formulas of T and that a direct verification of $\mathcal{A} \models \phi$ is much more work than the formal proof and all verifications $\mathcal{A} \models \psi$ together. Secondly, it might occur that a single theory T with $\mathcal{A} \models T$ is used to prove many formulas ϕ, so that many verifications $\mathcal{A} \models \phi$ are balanced against many formal proofs of ϕ out of T and a single set of verifications $\mathcal{A} \models \psi$. Especially when constructing formal proofs is considered easier then making verifications $\mathcal{A} \models \phi$, this reusability argument is very powerful. It also indicates that for a given algebra $\mathcal{A}$ we want to find a theory T from which most interesting formulas ϕ with $\mathcal{A} \models \phi$ can be proved.

Often there are reasons for representing processes in an algebra that satisfies a particular theory T, but there is no clear argument for selecting one of these algebras. In this situation we are interested in assertions $\mathcal{C} \models \phi$ with $\mathcal{C}$ the class of all algebras satisfying T. Of course assertions of this type can be conveniently proved by means of a formal proof of ϕ from T.

1.3. Proving formulas from modules. In process algebra we often want to modify the process algebra currently used to represent processes. Such a modification might be as simple as the addition of another operator, needed for the proper modelling of yet another feature of concurrency, but it can also be a more involved modification, such as factoring out a congruence, in order to identify processes that should not be distinguished in a certain application. It is our explicit concern to organise proofs of statements about concurrent systems in such a way that, whenever possible, our results carry over to modifications of the process algebra for which they were proved.

Now suppose $\mathcal{A}$ is a process algebra satisfying the theory T and a statement $\mathcal{A} \models \phi$ has been proved by means of a formal proof of ϕ out of T. Furthermore suppose that $\mathcal{B}$ is obtained from $\mathcal{A}$ by factoring out a congruence relation on $\mathcal{A}$ (so $\mathcal{B}$ is a *homomorphic image* of $\mathcal{A}$) and for a certain application $\mathcal{B}$ is considered to be a more suitable model of concurrency than $\mathcal{A}$. Then in general $\mathcal{B} \models \phi$ cannot be concluded, but if ϕ belongs to a certain class of formulas (the *positive* ones) it can. So if ϕ is positive we can use the following theorem: 'If $\mathcal{A} \models T$, $T \vdash \phi$, ϕ is positive, and $\mathcal{B}$ is a homomorphic image of $\mathcal{A}$, then $\mathcal{B} \models \phi$'. This saves us the trouble of finding another theory U, verifying that $\mathcal{B} \models U$ and proving $U \vdash \phi$ for many formulas ϕ that have been proved from T already. Another way of formulating the same idea is to introduce a module $H(T)$. We postulate that one may derive '$H(T) \vdash \phi$' from '$T \vdash \phi$' and 'ϕ is positive', and $H(T) \vdash \phi$ implies that ϕ holds in all homomorphic images of algebras satisfying T.

Thus we propose a generalisation of the notion of a formal proof. Instead of theories we use the more general notion of *modules*. Like a theory a module characterises a class $\mathcal{C}$ of algebras, but besides the class of all algebras satisfying a given set of formulas, $\mathcal{C}$ can for instance also be the class of homomorphic images or subalgebras of a class of algebras specified earlier. Now a proof in the framework of module algebra is a sequence or tree of assertions $M \vdash \phi$ such that in each step either the formula ϕ is manipulated, as in classical proofs, or the module M is manipulated. Of course we will establish a soundness theorem as before, and then an assertion $\mathcal{A} \models \phi$ can be proved by means of a module M with $\mathcal{A} \models M$ and a formal proof of ϕ out of M. We will now turn to the formal definitions.

1.4. Signatures. Let NAMES be a given set of names.
A *sort declaration* is an expression $\mathbb{S}:S$ with $S\in$ NAMES.
A *function declaration* is an expression $\mathbb{F}:f:S_1\times\cdots\times S_n\to S$ with $f,S_1,\ldots,S_n,S\in$ NAMES.
A *predicate declaration* is an expression $\mathbb{R}:p\subseteq S_1\times\cdots\times S_n$ with $p,S_1,\ldots,S_n\in$ NAMES.
A *signature* σ is a set of sort, function and predicate declarations, satisfying:

$$(\mathbb{F}:f:S_1\times\cdots\times S_n\to S)\in\sigma \;\Rightarrow\; (\mathbb{S}:S_i)\in\sigma\ (i=1,\ldots,n)\wedge(\mathbb{S}:S)\in\sigma$$

$$(\mathbb{R}:p\subseteq S_1\times\cdots\times S_n)\in\sigma \;\Rightarrow\; (\mathbb{S}:S_i)\in\sigma\ (i=1,\ldots,n)$$

A function declaration $\mathbb{F}:f:\to S$ of arity 0 is sometimes called a *constant declaration* and written as $\mathbb{F}:f\in S$.

1.5. σ-Algebras. Let σ be a signature. A *σ-algebra* $\mathcal{A}$ is a function on σ that maps

$(\mathbb{S}:S)\in\sigma$ to a set $S^{\mathcal{A}}$,

$(\mathbb{F}:f:S_1\times\cdots\times S_n\to S)\in\sigma$ to a function $f^{\mathcal{A}}_{S_1\times\cdots\times S_n\to S}:S_1^{\mathcal{A}}\times\cdots\times S_n^{\mathcal{A}}\to S^{\mathcal{A}}$,

$(\mathbb{R}:p\subseteq S_1\times\cdots\times S_n)\in\sigma$ to a predicate $p^{\mathcal{A}}_{S_1\times\cdots\times S_n}\subseteq S_1^{\mathcal{A}}\times\cdots\times S_n^{\mathcal{A}}$.

Let $\mathcal{A}$ and $\mathcal{B}$ be σ-algebras. $\mathcal{B}$ is a *subalgebra* of $\mathcal{A}$ if $S^{\mathcal{B}}\subseteq S^{\mathcal{A}}$ for all $(\mathbb{S}:S)\in\sigma$, if moreover $f^{\mathcal{A}}_{S_1\times\cdots\times S_n\to S}$ restricted to $S_1^{\mathcal{B}}\times\cdots\times S_n^{\mathcal{B}}\to S^{\mathcal{B}}$ is just $f^{\mathcal{B}}_{S_1\times\cdots\times S_n\to S}$ for all $\mathbb{F}:f:S_1\times\cdots\times S_n\to S$ in σ, and if $p^{\mathcal{A}}_{S_1\times\cdots\times S_n}$ restricted to $S_1^{\mathcal{B}}\times\cdots\times S_n^{\mathcal{B}}$ is just $p^{\mathcal{B}}_{S_1\times\cdots\times S_n}$ for all $\mathbb{R}:p\subseteq S_1\times\cdots\times S_n$ in σ.

A *homomorphism* $h:\mathcal{A}\to\mathcal{B}$ consists of mappings $h_S:S^{\mathcal{A}}\to S^{\mathcal{B}}$ for all $\mathbb{S}:S$ in σ, such that

$$h_S(f^{\mathcal{A}}_{S_1\times\cdots\times S_n\to S}(x_1,\ldots,x_n)) = f^{\mathcal{B}}_{S_1\times\cdots\times S_n\to S}(h_{S_1}(x_1),\ldots,h_{S_n}(x_n))$$

$$\text{for all } (\mathbb{F}:f:S_1\times\cdots\times S_n\to S)\in\sigma \text{ and all } x_i\in S_i^{\mathcal{A}} (i=1,\ldots,n)$$

$$p^{\mathcal{A}}_{S_1\times\cdots\times S_n}(x_1,\ldots,x_n) \Leftrightarrow p^{\mathcal{B}}_{S_1\times\cdots\times S_n}(h_{S_1}(x_1),\ldots,h_{S_n}(x_n))$$

$$\text{for all } (\mathbb{R}:p\subseteq S_1\times\cdots\times S_n)\in\sigma \text{ and all } x_i\in S_i^{\mathcal{A}} (i=1,\ldots,n)$$

$\mathcal{B}$ is a *homomorphic image* of $\mathcal{A}$ if there exists a surjective homomorphism $h:\mathcal{A}\to\mathcal{B}$.

Let $\mathcal{A}$ be a σ-algebra. The *restriction* $\rho\Box\mathcal{A}$ of $\mathcal{A}$ to the signature ρ is the $\rho\cap\sigma$-algebra $\mathcal{B}$, defined by

$S^{\mathcal{B}} = S^{\mathcal{A}}$ for all $(\mathbb{S}:S)\in\rho\cap\sigma$

$f^{\mathcal{B}}_{S_1\times\cdots\times S_n\to S} = f^{\mathcal{A}}_{S_1\times\cdots\times S_n\to S}$ for all $(\mathbb{F}:f:S_1\times\cdots\times S_n\to S)\in\rho\cap\sigma$

$p^{\mathcal{B}}_{S_1\times\cdots\times S_n} = p^{\mathcal{A}}_{S_1\times\cdots\times S_n}$ for all $(\mathbb{R}:p\subseteq S_1\times\cdots\times S_n)\in\rho\cap\sigma$

1.6. Logics. A *logic* $\mathcal{L}$ is a complex of prescriptions, defining for any signature σ

- a set $F_\sigma^\mathcal{L}$ of *formulas* over σ such that $F_\sigma^\mathcal{L} \cap F_\rho^\mathcal{L} = F_{\sigma\cap\rho}^\mathcal{L}$,
- a binary relation $\models_\sigma^\mathcal{L}$ on σ-algebras$\times F_\sigma^\mathcal{L}$ such that for all ρ-algebras $\mathcal{A}$ and $\phi \in F_{\sigma\cap\rho}^\mathcal{L}$: $\sigma\Box\mathcal{A} \models_{\sigma\cap\rho}^\mathcal{L} \phi \Leftrightarrow \mathcal{A} \models_\rho^\mathcal{L} \phi$
- and a set $I_\sigma^\mathcal{L}$ of *inference rules* $\dfrac{H}{\phi}$ with $H \subseteq F_\sigma^\mathcal{L}$ and $\phi \in F_\sigma^\mathcal{L}$.

If $\mathcal{A} \models_\sigma^\mathcal{L} \phi$ we say that the σ-algebra $\mathcal{A}$ *satisfies* the formula ϕ, or that ϕ *holds* in $\mathcal{A}$. A *theory* over σ is a set of formulas over σ. If T is a theory over σ and $\mathcal{A} \models_\sigma^\mathcal{L} \phi$ for all $\phi \in T$ we say that $\mathcal{A}$ satisfies T, notation $\mathcal{A} \models_\sigma^\mathcal{L} T$. We also say that $\mathcal{A}$ is a *model* of T.

A logic $\mathcal{L}$ is *sound* if $\dfrac{H}{\phi} \in I_\sigma^\mathcal{L}$ implies $\mathcal{A} \models_\sigma^\mathcal{L} H \;\Rightarrow\; \mathcal{A} \models_\sigma^\mathcal{L} \phi$ for any σ-algebra $\mathcal{A}$.

A formula $\phi \in F_\sigma^\mathcal{L}$ is *preserved under subalgebras* if $\mathcal{A} \models_\sigma^\mathcal{L} \phi$ implies $\mathcal{B} \models_\sigma^\mathcal{L} \phi$, for any subalgebra $\mathcal{B}$ of $\mathcal{A}$.
A formula $\phi \in F_\sigma^\mathcal{L}$ is *preserved under homomorphisms* if $\mathcal{A} \models_\sigma^\mathcal{L} \phi$ implies $\mathcal{B} \models_\sigma^\mathcal{L} \phi$, for any homomorphic image $\mathcal{B}$ of $\mathcal{A}$.

Without doubt, the definition of a 'logic' as presented above is too general for most applications. However, it is suited for our purposes and anyone can substitute his/her favourite (and more restricted) definition whenever he/she likes.

In the process algebra verifications of this chapter we will use infinitary conditional equational logic. The definition of this logic can be found in the appendix. For comparison, the definitions of equational logic and first order logic with equality are included too.

1.7. Classical logic.
DERIVABILITY. A σ-*proof* of a formula $\phi \in F_\sigma^\mathcal{L}$ from a theory $T \subseteq F_\sigma^\mathcal{L}$ using the logic $\mathcal{L}$, is a well-founded, upwardly branching tree of which the nodes are labelled by σ-formulas, such that

- the root is labelled by ϕ
- and if ψ is the label of a node q and H is the set of labels of the nodes directly above q then
  - either $\psi \in T$ and $H = \varnothing$,
  - or $\dfrac{H}{\psi} \in I_\sigma^\mathcal{L}$.

If a σ-proof of ϕ from T using $\mathcal{L}$ exists, we say that ϕ is σ-*provable* from T by means of $\mathcal{L}$, notation $T \vdash_\sigma^\mathcal{L} \phi$.

TRUTH. Let $\mathcal{C}$ be a class of σ-algebras and $\phi \in F_\sigma^\mathcal{L}$. Then ϕ is said to be *true* in $\mathcal{C}$, notation $\mathcal{C} \models_\sigma^\mathcal{L} \phi$, if ϕ holds in all σ-algebras $\mathcal{A} \in \mathcal{C}$. Let $Alg(\sigma, T)$ be the class of all σ-algebras satisfying T.

SOUNDNESS THEOREM. *If $\mathcal{L}$ is sound then $T \vdash_\sigma^\mathcal{L} \phi$ implies $Alg(\sigma,T) \models_\sigma^\mathcal{L} \phi$.*
PROOF. Straightforward with induction. □

If no confusion is likely to result, the sub- and superscripts of $\models$ and $\vdash$ may be dropped without further warning.

1.8. Module logic. The set $\mathcal{M}$ of modules is defined inductively as follows:

- If σ is a signature and T a theory over σ, then $(\sigma,T)\in\mathcal{M}$,
- If M and $N\in\mathcal{M}$ then $M+N\in\mathcal{M}$,
- If σ is a signature and $M\in\mathcal{M}$ then $\sigma\Box M\in\mathcal{M}$,
- If $M\in\mathcal{M}$ then $H(M)\in\mathcal{M}$,
- If $M\in\mathcal{M}$ then $S(M)\in\mathcal{M}$.

Here $+$ is the composition operator, allowing to organise specifications in a modular way, and $\Box$ is the export operator, restricting the visible signature of a module, thereby hiding auxiliary items. These operators occur in some form or other frequently in the literature on software engineering. Our notation is taken from Bergstra, Heering & Klint [17] in which also additional references can be found. The homomorphism operator H and the subalgebra operator S are, as far as we know, new in the context of algebraic specifications. Of course they are well known in model theory, see for instance Monk [98].

The *visible signature* $\Sigma(M)$ of a module M is defined inductively by:

- $\Sigma(\sigma,T) = \sigma$,
- $\Sigma(M+N) = \Sigma(M)\cup\Sigma(N)$,
- $\Sigma(\sigma\Box M) = \sigma\cap\Sigma(M)$,
- $\Sigma(H(M)) = \Sigma(M)$,
- $\Sigma(S(M)) = \Sigma(M)$.

Truth. The class $Alg(M)$ of models of a module M is defined inductively by:

- $\mathcal{A}$ is a model of (σ,T) if it is a σ-algebra, satisfying T;
- $\mathcal{A}$ is a model of $M+N$ if it is a $\Sigma(M+N)$-algebra, such that $\Sigma(M)\Box\mathcal{A}$ is a model of M and $\Sigma(N)\Box\mathcal{A}$ is a model of N;
- $\mathcal{A}$ is a model of $\sigma\Box M$ if it is the restriction of a model $\mathcal{B}$ of M to the signature σ;
- $\mathcal{A}$ is a model of $H(M)$ if it is a homomorphic image of a model $\mathcal{B}$ of M;
- $\mathcal{A}$ is a model of $S(M)$ if it is a subalgebra of a model $\mathcal{B}$ of M.

Note that $Alg(M)$ is a generalisation of $Alg(\sigma,T)$ as defined earlier. All the elements of $Alg(M)$ are $\Sigma(M)$-algebras. A $\Sigma(M)$-algebra $\mathcal{A}\in Alg(M)$ is said to *satisfy* M. A formula $\phi\in F^{\mathcal{L}}_{\Sigma(M)}$ is *satisfied* by a module M, notation $M\models^{\mathcal{L}}\phi$, if $Alg(M)\models^{\mathcal{L}}_{\Sigma(M)}\phi$, thus if ϕ holds in all $\Sigma(M)$-algebras satisfying M.

Derivability. A *proof* of a formula $\phi\in F^{\mathcal{L}}_{\Sigma(M)}$ from a module M using the logic $\mathcal{L}$, is a well-founded, upwardly branching tree of which the nodes are labelled by assertions $N\vdash\psi$, such that

- the root is labelled by $M\vdash\phi$
- if $N\vdash\psi$ is the label of a node q and H is the set of labels of the nodes directly above q then $\dfrac{H}{N\vdash\psi}$ is one of the inference rules of Table 1.

$(\sigma,T) \vdash \phi$	if $\phi \in T$
$\dfrac{M \vdash \phi_j \ \ (j \in J)}{M \vdash \phi}$	whenever $\dfrac{\phi_j \ \ (j \in J)}{\phi} \in I^{\mathcal{L}}_{\Sigma(M)}$
$\dfrac{M \vdash \phi}{M+N \vdash \phi}$	$\dfrac{N \vdash \phi}{M+N \vdash \phi}$
$\dfrac{M \vdash \phi}{\sigma \Box M \vdash \phi}$	if $\phi \in F^{\mathcal{L}}_{\sigma}$
$\dfrac{M \vdash \phi}{H(M) \vdash \phi}$	if ϕ is *positive*
$\dfrac{M \vdash \phi}{S(M) \vdash \phi}$	if ϕ is *universal*

TABLE 1

Here *positive* and *universal* are syntactic criteria, to be defined for each logic $\mathcal{L}$ separately, ensuring that a formula is preserved under homomorphisms and subalgebras respectively. We write $N \vdash \psi$ for $\dfrac{\varnothing}{N \vdash \psi}$, and omit braces in the conditions of inference rules. If a proof of ϕ from M using $\mathcal{L}$ exists, we say that ϕ is *provable* from M by means of $\mathcal{L}$, notation $M \vdash^{\mathcal{L}} \phi$.

LEMMA. *If* $M \vdash^{\mathcal{L}} \phi$ *then* $\phi \in F^{\mathcal{L}}_{\Sigma(M)}$.
PROOF. With induction. The only nontrivial cases are the rules for $+$ and $\Box$. These follow from $F^{\mathcal{L}}_{\sigma} \subseteq F^{\mathcal{L}}_{\sigma \cup \rho}$ and $F^{\mathcal{L}}_{\sigma} \cap F^{\mathcal{L}}_{\rho} \subseteq F^{\mathcal{L}}_{\sigma \cap \rho}$ respectively. □

SOUNDNESS THEOREM. *If* $\mathcal{L}$ *is sound then* $M \vdash^{\mathcal{L}} \phi$ *implies* $M \models^{\mathcal{L}} \phi$.
PROOF. With induction. Again the only nontrivial cases are the rules for $+$ and $\Box$. These follow since for all ρ-algebras $\mathcal{A}$ and $\phi \in F^{\mathcal{L}}_{\sigma \cap \rho}$: $\sigma \Box \mathcal{A} \models \phi \ \Rightarrow \ \mathcal{A} \models \phi$ and $\sigma \Box \mathcal{A} \models \phi \ \Leftarrow \ \mathcal{A} \models \phi$ respectively. □

2. PROCESS ALGEBRA

This thesis does not contain an introductory chapter on process algebra. We only give a listing of some important process modules. For an introduction to the ACP formalism we refer the reader to [19, 20, 22].

2.1. ACP$_\tau$. In this chapter a central role will be played by the module ACP$_\tau$, the Algebra of Communicating Processes with abstraction. A first parameter of ACP$_\tau$ is a finite set A of *actions*. For each action $a \in A$ there is a constant a in the language, representing the process, starting with an a-action and terminating (successfully) after some time.

The first two composition operations we consider are $\cdot$, denoting *sequential composition*, and $+$ for *alternative composition.* If x and y are two processes, then $x \cdot y$ is the process that starts execution of y after successful completion of x, and $x+y$ is the process that either behaves like x or like y. We do not specify whether the choice between x and y is made by the process itsself, or by the environment.

We have a special constant δ, denoting *deadlock, inaction,* a process that cannot do anything at all. In particular δ does not terminate succesfully. We write $A_\delta = A \cup \{\delta\}$.

Next we have a *parallel composition* operator $\|$. $x \| y$ denotes the process corresponding to the parallel execution of x and y. Execution of $x \| y$ either starts with a step from x, or with a step from y, or with a *synchronisation* of an action from x and an action from y. Synchronisation of actions is described by the second parameter of ACP$_\tau$, which is is a binary communication function $\gamma : A_\delta \times A_\delta \rightarrow A_\delta$ that is commutative, associative and has δ as zero element:

$$\gamma(a,b) = \gamma(b,a) \quad \gamma(a,\gamma(b,c)) = \gamma(\gamma(a,b),c) \quad \gamma(a,\delta) = \delta$$

If $\gamma(a,b) = c \neq \delta$ this means that actions a and b can synchronise. The synchronous performance of a and b is then regarded as a performance of the communication action c. Formally we should add the parameters to the name of a module: ACP$_\tau(A,\gamma)$. However, in order to keep notation simple, we will always omit the parameters if this can be done without causing confusion. In order to axiomatise the $\|$-operator we use two auxiliary operators $\mathbin{\|\!\_}$ (*left-merge*) and $|$ (*communication merge*). $x \mathbin{\|\!\_} y$ is $x \| y$, but with the restriction that the first step comes from x, and $x | y$ is $x \| y$ but with a synchronisation action as the first step.

Next we have for each $H \subseteq A$ an *encapsulation* operator ∂_H. The operator ∂_H blocks actions from H. The operator is used to encapsulate a process, i.e. to block synchronisation with the environment.

When describing concurrent systems and reasoning about their behaviour, it is often useful to have a distinguished action that cannot synchronise with any other action. Such an action is denoted by the constant $\tau \notin A_\delta$. The fact that τ cannot synchronise makes that in some sense this action is not observable. Therefore it is often called the *silent* action. For each $I \subseteq A$ the language contains an *abstraction* or *hiding* operator τ_I. This operator hides actions in I by renaming them into τ, thus expressing that certain actions in a system behaviour cannot be observed.

In Table 2 we summarize the signature of module ACP$_\tau$.

$\mathbb{S}$ (sort):	P		the set of processes
$\mathbb{F}$ (functions):	$+$:	$P\times P\rightarrow P$	alternative composition (sum)
	$\cdot$:	$P\times P\rightarrow P$	sequential composition (product)
	$\Vert$:	$P\times P\rightarrow P$	parallel composition (merge)
	$\Vert\!\!\lfloor$:	$P\times P\rightarrow P$	left-merge
	$\vert$:	$P\times P\rightarrow P$	communication-merge
	∂_H:	$P\rightarrow P$	encapsulation, for any $H\subseteq A$
	τ_I:	$P\rightarrow P$	abstraction, for any $I\subseteq A$
	a	$\in P$	for any atomic action $a\in A$
	δ	$\in P$	inaction, deadlock
	τ	$\in P$	silent action

TABLE 2

Table 3 contains the theory of the module ACP$_\tau$. In this chapter we present ACP$_\tau$ as a monolithic module. In [22] however, it is shown that ACP$_\tau$ can be viewed as the sum of a large number of sub-modules which are interesting in their own right. The module consisting of axioms A1-5 only is called BPA (from Basic Process Algebra). If we add axioms A6-7 we obtain BPA$_\delta$, and BPA$_\delta$ plus axioms T1-3 gives BPA$_{\tau\delta}$. The module ACP consists of the axioms A1-7, CF, CM1-9 and D1-4, i.e. the left column of Table 3. All axioms in Table 3 are in fact axiom schemes in a, b, H and I. Here a and b range over A_δ (unless further restrictions are made in the table) and $H,I\subseteq A$. In a product $x\cdot y$ we will often omit the $\cdot$. We take $\cdot$ to be more binding than other operations and $+$ to be less binding than other operations. In case we are dealing with an associative operator, we also leave out parentheses.

2.1.1. Note. Let $n>0$. Let $D=\{d_1,...,d_n\}$ be a finite set. Let $t_{d_1},...,t_{d_n}$ be process expressions. We use the notation $\sum_{d\in D} t_d$ for the sum $t_{d_1}+\cdots+t_{d_n}$. $\sum_{d\in\varnothing} t_d=\delta$ by definition.

2.1.2. Summand inclusion. In process verifications the summand inclusion predicate $\subseteq$ turns out to be a useful notation. It is defined by: $x\subseteq y \Leftrightarrow x+y=y$. From the ACP$_\tau$-axioms A1, A2 and A3 respectively it follows that $\subseteq$ is antisymmetrical, transitive and reflexive, and hence a partial order.

2.1.3. PROPOSITION. ACP$_\tau$ $\vdash \tau x\Vert y=\tau(x\Vert y)$.

PROOF. $\tau x\Vert y \supseteq \tau x\Vert\!\!\lfloor y=\tau(x\Vert y)=\tau x\Vert\!\!\lfloor y=\tau\tau x\Vert\!\!\lfloor y=\tau(\tau x\Vert y)\supseteq \tau x\Vert y$. Now use the fact that $\subseteq$ is a partial order. □

$x+y = y+x$	A1	$x\tau = x$	T1
$x+(y+z) = (x+y)+z$	A2	$\tau x+x = \tau x$	T2
$x+x = x$	A3	$a(\tau x+y) = a(\tau x+y)+ax$	T3
$(x+y)z = xz+yz$	A4		
$(xy)z = x(yz)$	A5		
$x+\delta = x$	A6		
$\delta x = \delta$	A7		
$a \mid b = \gamma(a,b)$	CF		
$x \parallel y = x \mathbin{\lfloor\!\lfloor} y + y \mathbin{\lfloor\!\lfloor} x + x \mid y$	CM1		
$a \mathbin{\lfloor\!\lfloor} x = ax$	CM2	$\tau \mathbin{\lfloor\!\lfloor} x = \tau x$	TM1
$(ax) \mathbin{\lfloor\!\lfloor} y = a(x \parallel y)$	CM3	$(\tau x) \mathbin{\lfloor\!\lfloor} y = \tau(x \parallel y)$	TM2
$(x+y) \mathbin{\lfloor\!\lfloor} z = x \mathbin{\lfloor\!\lfloor} z + y \mathbin{\lfloor\!\lfloor} z$	CM4	$\tau \mid x = \delta$	TC1
$(ax) \mid b = (a \mid b)x$	CM5	$x \mid \tau = \delta$	TC2
$a \mid (bx) = (a \mid b)x$	CM6	$(\tau x) \mid y = x \mid y$	TC3
$(ax) \mid (by) = (a \mid b)(x \parallel y)$	CM7	$x \mid (\tau y) = x \mid y$	TC4
$(x+y) \mid z = x \mid z + y \mid z$	CM8		
$x \mid (y+z) = x \mid y + x \mid z$	CM9		
		$\partial_H(\tau) = \tau$	DT
		$\tau_I(\tau) = \tau$	TI1
$\partial_H(a) = a \quad \text{if } a \notin H$	D1	$\tau_I(a) = a \quad \text{if } a \notin I$	TI2
$\partial_H(a) = \delta \quad \text{if } a \in H$	D2	$\tau_I(a) = \tau \quad \text{if } a \in I$	TI3
$\partial_H(x+y) = \partial_H(x)+\partial_H(y)$	D3	$\tau_I(x+y) = \tau_I(x)+\tau_I(y)$	TI4
$\partial_H(xy) = \partial_H(x) \cdot \partial_H(y)$	D4	$\tau_I(xy) = \tau_I(x) \cdot \tau_I(y)$	TI5

TABLE 3

2.1.4. Monotony. Most of the operators of ACP_τ are monotonous with respect to the summand inclusion ordering. Using essentially the distributivity of the operators over $+$, one can show that if $x \subseteq y$, ACP_τ proves:

- $x+z \subseteq y+z$,
- $x \cdot z \subseteq y \cdot z$,
- $x \mathbin{\lfloor\!\lfloor} z \subseteq y \mathbin{\lfloor\!\lfloor} z$,
- $x \mid z \subseteq y \mid z$,
- $\partial_H(x) \subseteq \partial_H(y)$,
- $\tau_I(x) \subseteq \tau_I(y)$.

Due to branching time, in general $z \cdot x \not\subseteq z \cdot y$, $x \parallel z \not\subseteq y \parallel z$ and $z \mathbin{\lfloor\!\lfloor} x \not\subseteq z \mathbin{\lfloor\!\lfloor} y$. However, we do have monotony of the merge for the case were x is of the form $\tau x'$. If $\tau x' \subseteq y$, then $\mathrm{ACP}_\tau \vdash \tau x' \parallel z \subseteq y \parallel z$:

$$\tau x' \parallel z \overset{2.1.3}{=} \tau(x' \parallel z) = \tau x' \mathbin{\lfloor\!\lfloor} z \subseteq y \mathbin{\lfloor\!\lfloor} z \subseteq y \parallel z.$$

2.2. Standard Concurrency. Often one adds to ACP$_\tau$ the following module SC of Standard Concurrency ($a \in A_\delta$), which is parametrised by A. A proof that these axioms hold for all closed recursion-free terms can be found in [20].

SC	$(x \mathbin{\|\!\!\lfloor} y) \mathbin{\|\!\!\lfloor} z = x \mathbin{\|\!\!\lfloor} (y \| z)$	SC1
	$(x \mid ay) \mathbin{\|\!\!\lfloor} z = x \mid (ay \mathbin{\|\!\!\lfloor} z)$	SC2
	$x \mid y = y \mid x$	SC3
	$x \| y = y \| x$	SC4
	$x \mid (y \mid z) = (x \mid y) \mid z$	SC5
	$x \| (y \| z) = (x \| y) \| z$	SC6

TABLE 4

2.3. Renamings. Let $A_{\tau\delta} = A_\delta \cup \{\tau\}$. For every function $f: A_{\tau\delta} \rightarrow A_{\tau\delta}$ with the property that $f(\delta) = \delta$ and $f(\tau) = \tau$, we introduce an operator $\rho_f : P \rightarrow P$. Axioms for ρ_f are given in Table 5 (Here $a \in A_{\tau\delta}$ and *id* is the identity). Module RN is parametrised by A.

RN	$\rho_f(a) = f(a)$	RN1
	$\rho_f(x+y) = \rho_f(x) + \rho_f(y)$	RN2
	$\rho_f(xy) = \rho_f(x) \cdot \rho_f(y)$	RN3
	$\rho_{id}(x) = x$	RN4
	$\rho_f \circ \rho_g(x) = \rho_{f \circ g}(x)$	RN5

TABLE 5

For $t \in A_{\tau\delta}$ and $H \subseteq A$ we define mappings $r_{t,H} : A_{\tau\delta} \rightarrow A_{\tau\delta}$ as follows:

$$r_{t,H}(a) = \begin{cases} t & \text{if } a \in H \\ a & \text{otherwise} \end{cases}$$

In the following we will implicitly identify the operators ∂_H and $\rho_{r_{\delta,H}}$, and also the operators τ_I and $\rho_{r_{\tau,I}}$: encapsulation is just renaming of actions into δ, and abstraction is renaming of actions into the silent step τ.

2.4. Chaining operators. A basic situation we will encounter is one in which processes input and output values in a domain D. Often we want to 'chain' two processes in such a way that the output of the first one becomes the input of the second. In order to describe this, we define *chaining* operators $\ggg$ and $\gg$. In the process $x \ggg y$ the output of process x serves as input of process y. Operator $\gg$ is identical to operator $\ggg$, but hides in addition the communications that take place at the internal communication port. The reason for introducing two operators is a technical one: the operator $\gg$ (in which we are interested most) often leads to the possibility of an infinite sequence of internal actions corresponding to hidden synchronisations between the two arguments of the operator (a form of *unguarded recursion*, cf. Sections 2.8.1 and 2.12.1). In order to deal with such behaviours, it is useful to view $\gg$ as the composition of two operators: the $\ggg$ operator and an abstraction operator that hides the communications of $\ggg$. We will define the chaining operators in terms of the operators of $\mathrm{ACP}_\tau + \mathrm{RN}$. In this way we obtain a simple, finite axiomatisation of the operators. The operator $\gg$ occurs (in a different notation) already in HOARE [75] and MILNER [92].

Let for $d \in D$, $\downarrow d$ be the action of reading d, and $\uparrow d$ be the action of sending d. Furthermore let $ch(D)$ be the following set:

$$ch(D) = \{\uparrow d, \downarrow d, s(d), r(d), c(d) \mid d \in D\}.$$

Here $r(d)$, $s(d)$ and $c(d)$ $(d \in D)$ are auxiliary actions which play a role in the definition of the chaining operators. The module for the chaining operators is parametrised by an action alphabet A satisfying $ch(D) \subseteq A$. The module should occur in a context with a module $\mathrm{ACP}_\tau(A, \gamma)$ where

$$range(\gamma) \cap \{\downarrow d, \uparrow d, s(d), r(d) \mid d \in D\} = \varnothing$$

and communication on $ch(D)$ is defined by

$$\gamma(s(d), r(d)) = c(d)$$

(all other communications give δ). The renaming functions $\uparrow s$ and $\downarrow r$ are defined by

$$\uparrow s(\uparrow d) = s(d) \quad \text{and} \quad \downarrow r(\downarrow d) = r(d) \quad (d \in D)$$

and $\uparrow s(a) = \downarrow r(a) = a$ for every other $a \in A_{\tau\delta}$. Now the 'concrete' chaining of processes x and y, notation $x \ggg y$, is defined by means of the axiom $(H = \{s(d), r(d) \mid d \in D\})$:

$$\boxed{x \ggg y = \partial_H(\rho_{\uparrow s}(x) \| \rho_{\downarrow r}(y)) \qquad \text{CH1}}$$

The 'abstract' chaining of processes x and y, notation $x \gg y$, is defined by means of the axiom $(I = \{c(d) \mid d \in D\})$:

$$\boxed{x \gg y = \tau_I(x \ggg y) \qquad \text{CH2}}$$

The module CH^+ consists of axioms CH1 and CH2, and is parametrised by A. The '+' in CH^+ refers to the auxiliary actions in the module, which will be removed in Section 3.

2.4.1. EXAMPLE. Let $D = \{0,1\}$. Process AND reads two bits and then outputs 1 if both are 1, and 0 otherwise:

$$AND = \downarrow 0 \cdot (\downarrow 0 \cdot \uparrow 0 + \downarrow 1 \cdot \uparrow 0) + \downarrow 1 \cdot (\downarrow 0 \cdot \uparrow 0 + \downarrow 1 \cdot \uparrow 1)$$

Process OR reads two bits, outputs 0 if both are 0, and 1 otherwise:

$$OR = \downarrow 0 \cdot (\downarrow 0 \cdot \uparrow 0 + \downarrow 1 \cdot \uparrow 1) + \downarrow 1 \cdot (\downarrow 0 \cdot \uparrow 1 + \downarrow 1 \cdot \uparrow 1)$$

Process NEG reads a bit b and outputs $1-b$:

$$NEG = \downarrow 0 \cdot \uparrow 1 + \downarrow 1 \cdot \uparrow 0$$

These processes can be composed using chaining operators. It is not too hard to prove:

$$(NEG \cdot NEG \gg AND) \gg NEG = OR$$

Note however that we do not have

$$(NEG \cdot NEG \ggg AND) \ggg NEG = OR$$

since in the LHS process internal computation steps are still visible.

2.5. Recursion. A *recursive specification* E is a set of equations $\{x = t_x \mid x \in V_E\}$ with V_E a set of variables and t_x a process expression for $x \in V_E$. Only the variables of V_E may appear in t_x. A solution of E is an interpretation of the variables of V_E as processes (in a certain domain), such that the equations of E are satisfied. Recursive specifications are used to define (or specify) infinite processes.

For each recursive specification E and $x \in V_E$, the module REC introduces a constant $<x \mid E>$, denoting the x-component of a solution of E.

In most applications the variables $X \in V_E$ in a recursive specification E will be chosen fresh, so that there is no need to repeat E in each occurrence of $<X \mid E>$. Therefore the convention will be adopted that once a recursive specification has been declared, $<X \mid E>$ can be abbreviated by X. If this is done, X is called a *formal variable*. Formal variables are denoted by capital letters. So after the declaration $X = aX$, a statement $X = aaX$ should be interpreted as an abbreviation of $<X \mid X = aX> = aa<X \mid X = aX>$.

Let $E = \{x = t_x \mid x \in V_E\}$ be a recursive specification, and t a process expression. Then $<t \mid E>$ denotes the term t in which each free occurrence of $x \in V_E$ is replaced by $<x \mid E>$. In a recursive language we have for each E as above and $x \in V_E$ an axiom

$$\boxed{<x \mid E> = <t_x \mid E> \qquad \text{REC}}$$

If the above convention is used, these formulas seem to be just the equations of E. The module REC is parametrised by the signature in which the recursive equations are written. In the presence of module REC each system of recursion equations over this signature has a solution.

2.6. Projection. The operator $\pi_n : P \to P$ $(n \in \mathbb{N})$ *stops* processes after they have performed n atomic actions, with the understanding that τ-steps are transparent. The axioms for π_n are given in Table 6. Module PR is parametrised by A.

PR	$\pi_n(\tau) = \tau$	PR1
	$\pi_0(ax) = \delta$	PR2
	$\pi_{n+1}(ax) = a \cdot \pi_n(x)$	PR3
	$\pi_n(\tau x) = \tau \cdot \pi_n(x)$	PR4
	$\pi_n(x+y) = \pi_n(x) + \pi_n(y)$	PR5

TABLE 6

In this chapter, as in other papers on process algebra, we have an infinite collection of unary projection operators. Another option, which we do not pursue here, but which might be more fruitful if one is interested in finitary process algebra proofs, is to introduce a single binary projection operator $\mathbb{F} : \pi : \mathbb{N} \times P \to P$.

2.7. Boundedness. The predicate $B_n \subseteq P$ $(n \in \mathbb{N})$ states that the nondeterminism displayed by a process before its n^{th} atomic steps is bounded. If for all $n \in \mathbb{N}$: $B_n(x)$, we say x is bounded. Axioms for B_n are in Table 7 $(a \in A_\delta)$. Module B is parametrised by A.

B	$B_0(x)$	B1
	$B_n(\tau)$	B2
	$\dfrac{B_n(x)}{B_n(\tau x)}$	B3
	$\dfrac{B_n(x)}{B_{n+1}(ax)}$	B4
	$\dfrac{B_n(x)\ ,\ B_n(y)}{B_n(x+y)}$	B5

TABLE 7

Boundedness predicates were introduced in [52].

2.8. Approximation Induction Principle. AIP^- is a proof rule which is vital if we want to prove things about infinite processes. The rule expresses the idea that if two processes are equal to any depth, and one of them is bounded then they are equal.

$$\boxed{(\mathrm{AIP}^-) \quad \frac{\forall n \in \mathbb{N} \quad \pi_n(x) = \pi_n(y)\,,\ B_n(x)}{x = y}}$$

The '$-$' in AIP$^-$, distinguishes the rule from a variant without predicates B_n.

2.8.1. DEFINITION. Let t be an open ACP$_\tau$-term without abstraction operators. An occurrence of a variable X in t is *guarded* if t has a subterm of the form $a \cdot M$, with $a \in A_\delta$, and this X occurs in M. Otherwise, the occurrence is *unguarded.*

Let $E = \{x = t_x \mid x \in V_E\}$ be a recursive specification in which all t_x are ACP$_\tau$-terms without abstraction operators. For $X, Y \in V_E$ we define:

$$X \xrightarrow{u} Y \Leftrightarrow Y \text{ occurs unguarded in } t_X.$$

We call E *guarded* if relation $\xrightarrow{u}$ is well-founded (i.e. there is no infinite sequence $X \xrightarrow{u} Y \xrightarrow{u} Z \xrightarrow{u} \cdots$).

2.8.2. THEOREM *(Recursive Specification Principle* (RSP)*).*
ACP$_\tau$ + REC + PR + B + AIP$^-$ $\vdash$

$$\boxed{(\mathrm{RSP}) \quad \frac{E}{x = <x \mid E>} \ E \text{ guarded}}$$

In plain English the RSP rule says that every guarded recursive specification has at most one solution.

2.8.3. EXAMPLE. Let $E = \{X = (a+b) \cdot X\}$ and $F = \{Y = a \cdot (a+b) \cdot Y + b \cdot Y\}$ be two recursive specifications. Since

$$<X \mid E> \, = (a+b) \cdot <X \mid E> \, = a \cdot <X \mid E> + b \cdot <X \mid E> \, =$$
$$= a \cdot (a+b) \cdot <X \mid E> + b \cdot <X \mid E>,$$

the constant $<X \mid E>$ satisfies the equation of F. Because the specification F is guarded, RSP now gives that $<X \mid E> \, = \, <Y \mid F>$.

2.9. Koomen's Fair Abstraction Rule (KFAR). In the verification of communication protocols one often uses the following rule, called Koomen's Fair Abstraction Rule ($I \subseteq A$). Module KFAR is parametrised by A.

$$\boxed{(\mathrm{KFAR}) \quad \frac{x = ix + y \quad (i \in I)}{\tau_I(x) = \tau \cdot \tau_I(y)}}$$

Fair abstraction here means that $\tau_I(x)$ will eventually exit the hidden i-cycle. Below we will formulate a generalisation of KFAR, the Cluster Fair Abstraction Rule (CFAR), which can be derived from KFAR.

2.9.1. DEFINITION. Let $E = \{X = t_X \mid X \in V_E\}$ be a recursive specification, and let $I \subseteq A$. A subset C of V_E is called a *cluster (of I) in E* iff for all $X \in C$:

$$t_X = \sum_{k=1}^{m} i_k \cdot X_k + \sum_{l=1}^{n} Y_l$$

(For $m \geqslant 0$, $i_1,...,i_m \in I \cup \{\tau\}$, $X_1,...,X_m \in C$, $n \geqslant 0$ and $Y_1,..., Y_n \in V_E - C$). Variables $X \in C$ are called *cluster variables*. For $X \in C$ and $Y \in V_E$ we say that

$$X \rightsquigarrow Y \Leftrightarrow Y \text{ occurs in } t_X.$$

We define

$$e(C) = \{Y \in V_E - C \mid \exists X \in C : X \rightsquigarrow Y\}$$

Variables in $e(C)$ are called *exits*. $\rightsquigarrow^*$ is the transitive and reflexive closure of $\rightsquigarrow$. Cluster C is *conservative* iff every exit can be reached from every cluster variable via a path in the cluster:

$$\forall X \in C \, \forall Y \in e(C) : X \rightsquigarrow^* Y.$$

2.9.2. EXAMPLE. The transition diagram of Figure 1 represents a cluster in a recursive specification. The nodes represent variables in the recursive specification, labelled edges represent summands, and the triangles denote exits. The sets $\{1,2,3\}$, $\{4,5,6,7\}$, $\{8\}$ and $\{1,2,3,4,5,6,7,8\}$ are examples of conservative clusters. Cluster $\{1,2,3,4,5,6,7\}$ is not conservative since exit Z cannot be reached from cluster variables 4, 5, 6 and 7.

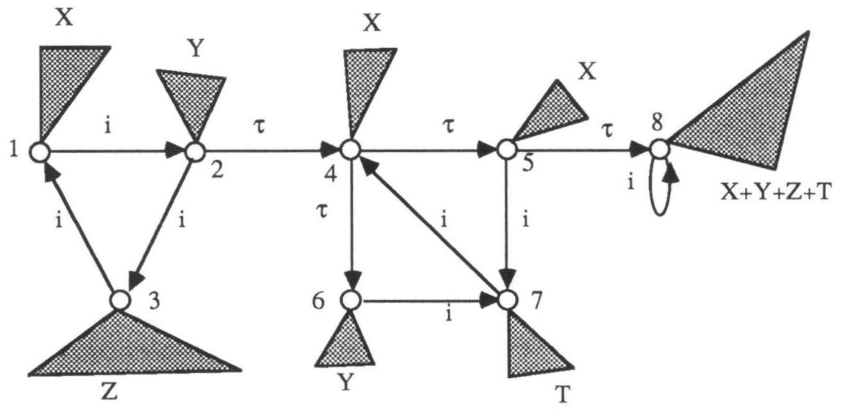


FIGURE 1

2.9.3. DEFINITION. The *Cluster Fair Abstraction Rule (CFAR)* reads as follows:

(CFAR)	Let E be a guarded recursive specification; let $I \subseteq A$ with $\|I\| \geqslant 2$; let C be a finite conservative cluster of I in E; and let $X, X' \in C$ with $X \rightsquigarrow X'$. Then: $\tau_I(X) = \tau \cdot \sum\limits_{Y \in e(C)} \tau_I(Y)$

2.9.4. THEOREM. ACP_τ + RN + REC + RSP + KFAR $\vdash$ CFAR.
PROOF. See [117]. □

2.10. Alphabets. Intuitively the alphabet of a process is the set of atomic actions which it can perform. This idea is formalised in [8], where an operator $\alpha: P \rightarrow 2^A$ is introduced, with axioms such as:

$$\alpha(\delta) = \varnothing$$

$$\alpha(ax) = \{a\} \cup \alpha(x)$$

$$\alpha(x+y) = \alpha(x) \cup \alpha(y)$$

In this approach the question arises what axioms should be adopted for the set-operators $\cup$, $\cap$, etc. One option, which is implicitly adopted in previous papers on process algebra, is to take the equalities which are true in set theory. This collection is unstructured and too large for our purposes. Therefore we propose a different, more algebraic solution. We view the alphabet of a process as a *process*; the alphabet operator α goes from sort P to sort P. Process $\alpha(x)$ is the alternative composition of the actions which can be performed by x. In this way we represent a set of actions by a process. A set B of actions is represented by the process expression $\boldsymbol{B} =_{def} \sum\limits_{b \in B} b$. So the empty set is represented by δ, a singleton-set $\{a\}$ by the expression a, and a set $\{a,b\}$ by expression $a+b$. Set union corresponds to alternative composition. The process algebra axioms A1-3 and A6 correspond to similar axioms for the set union operator. The notation $\subseteq$ for summand inclusion between processes (Section 2.1.2), fits with the notation for the subset predicate on sets.

The following axioms in Table 8 define the alphabet of finite processes ($a \in A$). Module AB is parametrised by A.

AB	$\alpha(\delta) = \delta$	AB1
	$\alpha(ax) = a + \alpha(x)$	AB2
	$\alpha(x+y) = \alpha(x) + \alpha(y)$	AB3
	$\alpha(\tau) = \delta$	AB4
	$\alpha(\tau x) = \alpha(x)$	AB5

TABLE 8

In order to compute the alphabet of infinite processes, we introduce an additional module AA which is parametrised by A.

AA	$\alpha(x) \subseteq A$	AA1
	$\alpha(x \parallel y) = \alpha(x) + \alpha(y) + \alpha(x) \mid \alpha(y)$	AA2
	$\alpha \circ \rho_f(x) \subseteq \rho_f \circ \partial_H \circ \alpha(x)$ (where $H = \{a \in A \mid f(a) = \tau\}$)	AA3
	$\dfrac{\forall n \in \mathbb{N} \;\; \alpha(\pi_n(x)) \subseteq y}{\alpha(x) \subseteq y}$	AA4

TABLE 9

It is not hard to see that the axioms of AA hold for all closed recursion-free terms.

2.10.1. EXAMPLE. (from [8]). Let $p = <X \mid \{X = aX\}>$, and define $q = \tau_{\{a\}}(p)$, $r = q \cdot b$ (with $b \neq a$). What is the alphabet of r? We derive:

$$\begin{aligned} \alpha(r) &= \alpha(qb) = \alpha(\tau_{\{a\}}(p) \cdot b) = \alpha(\tau_{\{a\}}(p) \cdot \tau_{\{a\}}(b)) = \\ &= \alpha(\tau_{\{a\}}(pb)) \overset{AA3}{\subseteq} \tau_{\{a\}} \circ \partial_{\{a\}} \circ \alpha(pb) \overset{RN5}{=} \partial_{\{a\}} \circ \alpha(pb). \end{aligned}$$

Since

$$\alpha(pb) = \alpha(apb) \overset{AB2}{=} a + \alpha(pb),$$

we have that $a \subseteq \alpha(pb)$. On the other hand we derive for $n \in \mathbb{N}$:

$$\alpha(\pi_n(pb)) = \alpha(a^n \cdot \delta) \subseteq a$$

and therefore, by application of axiom AA4, $\alpha(pb) \subseteq a$.
Consequently $\alpha(pb) = a$ and

$$\alpha(r) = \partial_{\{a\}} \circ \alpha(pb) = \partial_{\{a\}}(a) = \delta.$$

Information about alphabets must be available if we want to apply the following rules. These rules, which are a generalisation of the conditional axioms of [8], occur in a slightly different form also in [118]. Rules like these are an important tool in system verifications based on process algebra. Module RR is parametrised by A and γ. Observe that axioms AA1 and RR1 together imply axiom RN4 of Table 5. Axiom RR2, which describes the interaction between renaming and parallel composition, looks complicated, but that is only because it is so general. The axioms RR are derivable for closed recursion-free terms.

$\dfrac{\alpha(x)\subseteq \boldsymbol{B}}{\rho_f(x)=x}\forall b\in B: f(b)=b$	RR1
$\dfrac{\alpha(x)\subseteq \boldsymbol{B},\ \alpha(y)\subseteq \boldsymbol{C}}{\rho_f(x\parallel y)=\rho_f(x\parallel\rho_f(y))}\forall c\in C. f(c)=f^2(c)\wedge(\forall b\in B. f\circ\gamma(b,c)=f\circ\gamma(b,f(c)))$	RR2

TABLE 10

2.10.2. LEMMA: *(Conditional Axioms* (CA)*): Let CA be the theory consisting of the conditional axioms in Table 11. Then:* ACP$_\tau$ + RN + AB + RR $\vdash$ CA.

$\dfrac{\alpha(x)\mid(\alpha(y)\cap H)\subseteq H}{\partial_H(x\parallel y)=\partial_H(x\parallel\partial_H(y))}$	CA1	$\dfrac{\alpha(x)\mid(\alpha(y)\cap I)=\varnothing}{\tau_I(x\parallel y)=\tau_I(x\parallel\tau_I(y))}$	CA2
$\dfrac{\alpha(x)\cap H=\varnothing}{\partial_H(x)=x}$	CA3	$\dfrac{\alpha(x)\cap I=\varnothing}{\tau_I(x)=x}$	CA4
$\dfrac{H=H_1\cup H_2}{\partial_H(x)=\partial_{H_1}\circ\partial_{H_2}(x)}$	CA5	$\dfrac{I=I_1\cup I_2}{\tau_I(x)=\tau_{I_1}\circ\tau_{I_2}(x)}$	CA6
$\dfrac{H\cap I=\varnothing}{\tau_I\circ\partial_H(x)=\partial_H\circ\tau_I(x)}$	CA7		

TABLE 11

PROOF: We prove three of the rules. The others can be dealt with similarly.

CA3: Choose $a\in\alpha(x)$. Then $a\notin H$. This means that $r_{\delta,H}(a)=a$. Because a was chosen arbitrarily, we can aply rule RR1, which gives $\rho_{r_{\delta,H}}(x)=\partial_H(x)=x$.

CA5: Follows immediately from the observation

$$r_{\delta,H}=r_{\delta,H_1}\circ r_{\delta,H_2}$$

and application of axiom RN5 of Table 5.

CA1: Choose $c\in\alpha(y)$. We have:

$$r_{\delta,H}(c)=r_{\delta,H}\circ r_{\delta,H}(c)$$

Choose $b\in\alpha(x)$. If $c\notin H$ then $r_{\delta,H}(c)=c$ and the condition of rule RR2 is fulfilled. If $c\in H$ then either $\gamma(b,c)$ equals δ (so that we have $r_{\delta,H}\circ\gamma(b,c)=\delta$), or $\gamma(b,c)\in H$, so that again $r_{\delta,H}\circ\gamma(b,c)=\delta$. But in case $c\in H$ we also have

$$r_{\delta,H}\circ\gamma(b,r_{\delta,H}(c))=r_{\delta,H}\circ\gamma(b,\delta)=\delta$$

This means that we can apply rule RR2. □

2.10.3. REMARK. In most of the situations where we want to apply axiom CA1, H does not contain results of communications: $(A\,|\,A)\cap H=\varnothing$. Further actions from $\alpha(x)$ will not communicate with actions from H. In these cases the following weakened version of axiom CA1 is already strong enough:

$$\frac{\alpha(x)\,|\,H\ =\ \varnothing}{\partial_H(x\,\|\,y)\ =\ \partial_H(x\,\|\,\partial_H(y))}\ \text{CA1}\star$$

2.11. ACP$^\sharp_\tau$. The combination of all modules presented thus far, except for KFAR, will be called ACP$^\sharp_\tau$ (the system ACP$^\sharp_\tau$ as presented here slightly differs from a system with the same name occurring in [22]). The module is defined by:

$$\text{ACP}^\sharp_\tau\ =\ \text{ACP}_\tau+\text{SC}+\text{RN}+\text{CH}^+ +\text{REC}+\text{PR}+\text{B}+\text{AIP}^- +\text{AB}+\text{AA}+\text{RR}$$

Bisimulation semantics, as described in for instance [9], gives a model for the module ACP$^\sharp_\tau$ + KFAR. Work of BERGSTRA, KLOP & OLDEROG [23] showed that in a large number of interesting models KFAR is not valid. Therefore we have chosen not to include KFAR in the 'standard' module ACP$^\sharp_\tau$.

2.12. Generalised Recursive Specification Principle. For many applications the RSP is too restrictive. Therefore we will present below a more general version of this rule, called RSP$^+$.

2.12.1. DEFINITION. Let $\mathcal{P}$ be the set of closed expressions in the signature of ACP$^\sharp_\tau$. A process expression $p\in\mathcal{P}$ is called *guardedly specifiable* if there exists a guarded recursive specification F with $Y\in V_F$ such that

$$\text{ACP}^\sharp_\tau \vdash p=\,<Y\,|\,F>.$$

We have the following theorem:

2.12.2. THEOREM *(Generalised Recursive Specification Principle* (RSP$^+$)*).*
ACP$^\sharp_\tau$ ⊢

$$\boxed{(\text{RSP}^+)\qquad \frac{E}{x\ =\ <x\,|\,E>}\quad <x\,|\,E>\ \text{guardedly specifiable}}$$

2.12.3. Remarks. In the definition of the notion 'guardedly specifiable', it is essential that the identity $p=\,<Y\,|\,F>$ is *provable.* If we would only require that $p=\,<Y\,|\,F>$, then the corresponding version of RSP$^+$ would not be provable from ACP$^\sharp_\tau$, since this rule would then not be valid in the action relation model of [52]. In this model we have the identity $<X\,|\,\{X=X\}>\,=\delta$.[1]

[1] Strictly speaking, this is not correct. In [52], a recursion construct $<X\,|\,E>$ is viewed as a kind of variable which ranges over the X-components of the solutions of E. Since any process X satisfies $X=X$, the identity $<X\,|\,\{X=X\}>\,=\delta$ does not hold under this interpretation. However,

Hence $<X|\{X=X\}> = <Y|\{Y=\delta\}> = \delta$. Since the specification $\{Y=\delta\}$ is guarded, this would mean that expression $<X|\{X=X\}>$ is guardedly specifiable. But then RSP$^+$ gives that for arbitrary x: $x = <X|\{X=X\}> = \delta$. This is clearly false.

We conjecture that an expression p is guardedly specifiable iff it is provably bounded, i.e. for all $n \in \mathbb{N}$: $\mathrm{ACP}_\tau^\sharp \vdash B_n(x)$.

3. Applications of the module approach in process algebra

3.1. The auxiliary status of the left-merge.

3.1.1. Semantics. Sometimes it happens that our 'customers' complain that they do not succeed in proving the identity of two processes in $\mathrm{ACP}_\tau^\sharp$, whose behaviour is considered 'intuitively the same'. Often, this is because there are many intuitions possible, and $\mathrm{ACP}_\tau^\sharp$ happens not to represent the particular intuitions of these customers. Therefore we have defined some auxiliary modules that should bridge the gaps between intuitions.

In general a user of process algebra wants that his system proves $p = q$ (here p and q are closed process expressions in the signature of $\mathrm{ACP}_\tau^\sharp$), whenever p and q have the same interesting properties. So it depends on what properties are interesting for a particular user, whether his system should be designed to prove the equality of p and q or not. For this reason the semantical branch of process algebra research generated a variety of process algebras in which different identification strategies were pursued. In *bisimulation semantics* we find algebras that distinguish between any two processes that differ in the precise timing of internal choices; in *trace semantics* only processes are distinguished which can perform different sequences of actions; and, somewhere in between, the algebras of *failure semantics* identify processes if they have the same traces (can perform the same sequences of actions) and have the same deadlock behaviour in any context. A lot of these process algebras can be organised as homomorphic images of each other, as indicated in Figure 2. For concrete process algebra (without τ-moves) these process algebras have been defined in Chapter I. If two process expressions p and q represent the same process in bisimulation semantics with explicit divergence, they have many properties in common; if they only represent the same process in trace semantics, this only guarantees that they share some of these properties; and, descending from bisimulation semantics with explicit divergence to trace semantics, less and less distinctions are made. Now a user should state exactly in which properties of processes (s)he is interested. Suppose (s)he is only interested in traces and deadlock behaviour, then we can tell that for this purpose failure semantics suffices. This means that if processes p and q are proven equal in failure semantics, this guarantees that they have the same relevant properties. If they are only identified in trace semantics (somewhere in the lattice below failure semantics) such a conclusion cannot be drawn, but if they are identified in a semantics finer than failure semantics (such as bisimulation

if one interprets the construct $<X|E>$ as a constant in the model of [52], then the most natural choice is to relate to $<X|E>$ the bisimulation equivalence class of the term $<X|E>$. Under

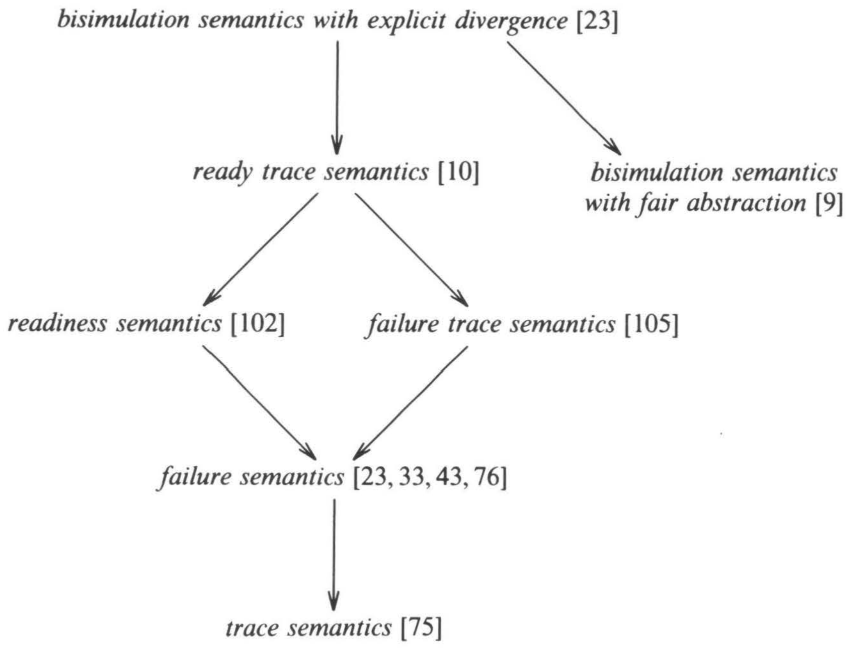

bisimulation semantics with explicit divergence [23]
ready trace semantics [10]
bisimulation semantics with fair abstraction [9]
readiness semantics [102]
failure trace semantics [105]
failure semantics [23, 33, 43, 76]
trace semantics [75]

FIGURE 2. *The linear time - branching time spectrum*

semantics with explicit divergence), then they certainly have the same interesting properties, and probably some uninteresting ones as well. Hence a proof in bisimulation semantics with explicit divergence is just as good as one in failure semantics (or even better).

This is the reason that we do our proofs mostly in bisimulation semantics: the entire module $\mathrm{ACP}^{\sharp}_{\tau}$ is sound with respect to bisimulation semantics with explicit divergence. However, if two processes are different in bisimulation semantics, we will never succeed in proving them equal from $\mathrm{ACP}^{\sharp}_{\tau}$. In such a case we might add some axioms to the system, that represent the extra identifications made in a less discriminating semantics. If we find a proof from this enriched module, it can be used by anyone satisfied with the properties of this coarser semantics.

It is in the light of the above considerations that one should judge the appearance of the following module T4:

$$\text{T4} \quad \boxed{\tau(\tau x + y) = \tau x + y}$$

The law of this module does not hold in bisimulation semantics, but it does hold in all other semantics of Figure 2. Thus any identity derived from $\mathrm{ACP}^{\sharp}_{\tau}$ + T4 holds in ready trace semantics and hence also in the courser ones like failure and trace semantics, or so it seems ....

this interpretation $<X \mid \{X = X\}> = \delta$.

3.1.2. An inconsistency.

3.1.2.1. DEFINITION. Let M be a process module with $\Sigma(M) \supseteq \Sigma(\mathrm{BPA}_{\tau\delta})$. We call M *consistent* if for all closed expressions x and y in the signature of $\mathrm{BPA}_{\tau\delta}$ with

$$M \vdash x = y,$$

the sets of complete traces agree:

$$trace(x) = trace(y).$$

A *complete trace* is a finite sequence of actions, ending with a symbol $\surd$ or δ indicating successful resp. unsuccessful termination. A formal definition of the set $trace(x)$ is given in [23]. Here we only give some examples, which should make the notion sufficiently clear:

$$trace(abc + ad\delta + a(\tau bc + d)) = \{abc\surd,\ ad\delta,\ ad\surd\}$$

$$trace(\tau) = \{\surd\} \neq \{\delta,\ \surd\} = trace(\tau + \tau\delta)$$

A model $\mathcal{Q}$ of M is *consistent* if for all closed expressions x and y in the signature of $\mathrm{BPA}_{\tau\delta}$ with

$$\mathcal{Q} \models x = y,$$

the sets of complete traces agree. The module $\mathrm{ACP}_\tau^\sharp$ + KFAR is consistent because bisimulation semantics with fair abstraction, as described in [9], gives a consistent model for this module. However, KFAR is not valid in any of the other semantics of Figure 2.

3.1.2.2. PROPOSITION.
$\mathrm{ACP}_\tau + \mathrm{T4} \vdash \tau(ac + ca) + bc = \tau(\tau(ac + ca) + bc + c(\tau a + b)).$
PROOF.

$$\tau(\tau a + b) \mathbin{\lfloor\!\lfloor} c = (\tau a + b) \mathbin{\lfloor\!\lfloor} c = \tau(a \| c) + bc = \tau(ac + ca) + bc$$

$$\tau(\tau a + b) \mathbin{\lfloor\!\lfloor} c = \tau((\tau a + b) \| c) = \tau(\tau(ac + ca) + bc + c(\tau a + b)) \qquad \square$$

Proposition 3.1.2.2 shows that module $\mathrm{ACP}_\tau + \mathrm{T4}$ is not consistent. This sudden inconsistency must be the result of a serious misunderstanding. And indeed, what's wrong is the use of ACP_τ in the less discriminating models (say in failure semantics). It happens that, in a setting with τ, failure equivalence (or ready trace equivalence for that matter) is not a congruence for the left-merge $\lfloor\!\lfloor$, and this causes all the trouble.

3.1.3. Solution. In applications we do not use the operators $\mathbin{\|\!\!\lfloor}$ and $|$ directly. In specifications we use the merge operator $\|$, and $\mathbin{\|\!\!\lfloor}$ and $|$ are only auxiliary operators, needed to give a complete axiomatisation of the merge.

Let sacp_τ be the signature obtained from $\Sigma(\mathrm{ACP}_\tau)$ by stripping the left-merge and communication-merge:

$$\mathrm{sacp}_\tau = \Sigma(\mathrm{ACP}_\tau) - \{\mathbb{F} : \mathbin{\|\!\!\lfloor} : P \times P \rightarrow P,\ \mathbb{F} : | : P \times P \rightarrow P\}$$

Failure equivalence as in [23], etc. are congruences for the operators of sacp_τ. However, the operators $\mathbin{\|\!\!\lfloor}$ and $|$ in ACP_τ are needed to axiomatise the $\|$-operator, and without them even the most elementary equations cannot be derived. Our solution to this problem is based on the following idea. Suppose we want to prove an equation $p=q$ in the signature sacp_τ that holds in ready trace semantics (and hence in failure semantics) but not in bisimulation semantics. Then we first prove an intermediate result from ACP_τ: one or more equations holding in bisimulation semantics (with explicit divergence) and in which no $\mathbin{\|\!\!\lfloor}$ and $|$ appear. This intermediate result is preserved after mapping the bisimulation model homomorphically on the ready trace or failure model, and can be combined consistently with the axiom T4. Thus the proof of $p=q$ can be completed. In our language of modules we can describe this as follows. The module

$$SACP_\tau = H(\mathrm{sacp}_\tau \square (\mathrm{ACP}_\tau + \mathrm{SC}))$$

does not contain the operators $\mathbin{\|\!\!\lfloor}$ and $|$ in its visible signature and since failure semantics can be obtained as a homomorphic image of bisimulation semantics, considering that $\mathrm{ACP}_\tau + \mathrm{SC}$ is sound w.r.t. bisimulation semantics and that the operators of sacp_τ carry over to failure semantics, we conclude that this module is sound w.r.t. failure semantics. Hence it can be combined consistently with T4, and SACP_τ is a suitable framework for proving statements in failure semantics.

We would like to stress that the use of the H-operator is essential here. The H-operator makes that from module SACP_τ only *positive* formulas are provable. The following example shows what goes wrong if we also allow non-positive formulas. From the proof of Proposition 3.1.2.2 it follows that:

$$\mathrm{sacp}_\tau \square (\mathrm{ACP}_\tau + \mathrm{SC}) \vdash \frac{\tau(\tau a + b) = \tau a + b}{c(\tau a + b) \subseteq \tau(ac + ca) + bc}$$

Consequently we can prove an inconsistency if we add law T4:

$$\mathrm{sacp}_\tau \square (\mathrm{ACP}_\tau + \mathrm{SC}) + \langle \tau(\tau x + y) = \tau x + y \rangle \vdash c(\tau a + b) \subseteq \tau(ac + ca) + bc$$

So although the formulas provable from module $\mathrm{sacp}_\tau \square (\mathrm{ACP}_\tau + \mathrm{SC})$ contain no left-merge, some of them (which are non-positive) cannot be combined consistently with the laws of ready trace semantics and failure semantics.

3.2. Associativity of the chaining operator. ACP$_\tau$ is a universal specification formalism in the sense that in bisimulation semantics every finitely branching, effectively presented process can be specified in ACP$_\tau$ by a finite system of recursion equations (see [9]). Still it often turns out that adding new operators to the theory facilitates specification and verification of concurrent systems. In general, adding new operators and laws can have far reaching consequences for the underlying mathematical theory. Often however, new operators are *definable* in terms of others operators and the axioms are *derivable* from the other axioms. In that case the new operators can be considered as notations which are useful, but do not complicate the underlying theory in any way. Examples of definable operators are the projection operators and the process creation operator of [16].

Just like the left-merge and the communication-merge are needed in order to axiomatise the parallel composition operator, new atomic actions are often needed if we want to define a new operator in terms of more elementary operators. As an example we mention the actions $s(d)$ and $r(d)$ which we need in the definition of the chaining operators. These auxiliary atoms will never be used in process specifications. Unfortunately they have the unpleasant property that they occur in some important algebraic laws for the new operators. One of the properties of the chaining operators we use most is that they are 'associative'. However, due to the auxiliary actions, the chaining operators are not associative in general. We do not have general associativity in the model of bisimulation semantics. Counterexample:

$$(r(d) \ggg (s(d)+s(e))) \ggg r(e) \;=\; c(d)\cdot\delta$$

$$r(d) \ggg ((s(d)+s(e)) \ggg r(e)) \;=\; c(e)\cdot\delta$$

However, we *do* have associativity under some very weak assumptions. In the model of bisimulation semantics, the following law is valid (here $H=\{s(d),r(d)\,|\,d\in D\}$):

$$\boxed{\frac{\partial_H(x)=x,\, \partial_H(y)=y,\, \partial_H(z)=z}{(x \ggg y) \ggg z = x \ggg (y \ggg z)}\;\; \mathrm{CC}}$$

It would be much nicer if we somehow could 'hide' the auxiliary atoms, and, for the $\ggg$-operator, have associativity in general. In this section we will see how this can be accomplished by means of the module approach.

3.2.1. The associativity of the chaining operators. Although the rule CC holds in the model of bisimulation semantics, we have not been able to prove it algebraically from module ACP$_\tau^\sharp$. However, we can prove algebraically a weaker version of rule CC if we make some additional assumptions about the alphabet. We assume that besides actions $ch(D)$, the alphabet A contains actions:

$$\overline{H} \;=\; \{\overline{s}(d),\overline{r}(d)\,|\,d\in D\} \;\text{ en }\; \underline{H} \;=\; \{\underline{s}(d),\underline{r}(d)\,|\,d\in D\}$$

One may think about these actions as special fresh atoms which are added to

A only in order to prove the associativity of the chaining operators.[1] Let $\underline{H}=\{\underline{r}(d),\underline{s}(d)\,|\,d\in D\}$ and let $\hat{H}=H\cup\overline{H}\cup\underline{H}$. We assume that actions from $\hat{H}$ do not synchronise with the other actions in the alphabet, and that $range(\gamma)\cap\hat{H}\ =\ \varnothing$. On $\hat{H}$ communication is given by $(d\in D)$:

$$\begin{aligned}\gamma(\overline{s}(d),\ \overline{r}(d)) &= \gamma(\overline{s}(d),\ r(d)) = \gamma(s(d),\ \overline{r}(d)) = \gamma(s(d),\ r(d)) = \\ &= \gamma(\underline{s}(d),\ \underline{r}(d)) = \gamma(\underline{s}(d),\ r(d)) = \gamma(s(d),\ \underline{r}(d)) = c(d)\end{aligned}$$

We define for $v,w\in\{\uparrow,\downarrow,s,r,\overline{s},\overline{r},\underline{s},\underline{r}\}$ the renaming function vw:

$$vw(a) = \begin{cases} w(d) & \text{if } a=v(d) \text{ for some } d\in D \\ a & \text{otherwise} \end{cases}$$

3.2.1.1. LEMMA. SACP_τ + RN + CH^+ + AB + AA + RR $\vdash$

$$\frac{\partial_{\hat{H}}(x)=x,\ \partial_{\hat{H}}(y)=y,\ \partial_{\hat{H}}(z)=z}{\partial_{\overline{H}}(\rho_{\uparrow\overline{s}}(x)\|\rho_{\downarrow\overline{r}}(y))=x\ggg y=\partial_{\underline{H}}(\rho_{\uparrow\underline{s}}(x)\|\rho_{\downarrow\underline{r}}(y))}$$

PROOF. We only prove the first equality. The second one follows by symmetry.

$$\begin{aligned}\partial_{\overline{H}}(\rho_{\uparrow\overline{s}}(x)\|\rho_{\downarrow\overline{r}}(y)) &= \text{(Note 1 below, RR1)} \\ &= \partial_{\overline{H}}\circ\rho_{s\overline{s}}\circ\rho_{r\overline{r}}(\rho_{\uparrow\overline{s}}(x)\|\rho_{\downarrow\overline{r}}(y)) = (\text{RN5, } y=\partial_{\hat{H}}(y)) \\ &= \partial_{\overline{H}}\circ\rho_{s\overline{s}}\circ\rho_{r\overline{r}}(\rho_{\uparrow\overline{s}}(x)\|\rho_{r\overline{r}}\circ\rho_{\downarrow r}(y)) = \text{(Note 2, RR2)} \\ &= \partial_{\overline{H}}\circ\rho_{s\overline{s}}\circ\rho_{r\overline{r}}(\rho_{\uparrow\overline{s}}(x)\|\rho_{\downarrow r}(y)) = (\text{SC4, RN5, } x=\partial_{\hat{H}}(x)) \\ &= \partial_{\overline{H}}\circ\rho_{r\overline{r}}\circ\rho_{s\overline{s}}(\rho_{\downarrow r}(y)\|\rho_{s\overline{s}}\circ\rho_{\uparrow s}(x)) = \text{(as in Note 2, RR2)} \\ &= \partial_{\overline{H}}\circ\rho_{r\overline{r}}\circ\rho_{s\overline{s}}(\rho_{\downarrow r}(y)\|\rho_{\uparrow s}(x)) = \text{(RN5)} \\ &= \partial_H\circ\partial_{\overline{H}}(\rho_{\downarrow r}(y)\|\rho_{\uparrow s}(x)) = \text{(Note 3, RR1, SC4)} \\ &= \partial_H(\rho_{\uparrow s}(x)\|\rho_{\downarrow r}(y)) \overset{CH1}{=} x\ggg y\end{aligned}$$

Note 1. Let $B=A-H$. We claim $\alpha(\rho_{\uparrow\overline{s}}(x)\|\rho_{\downarrow\overline{r}}(y))\subseteq\boldsymbol{B}$ (recall that $\boldsymbol{B}=_{def}\sum_{b\in B}b$).

PROOF: $\alpha(\rho_{\uparrow\overline{s}}(x)\|\rho_{\downarrow\overline{r}}(y))\ =$

$$\overset{AA2}{=}\ \alpha\circ\rho_{\uparrow\overline{s}}(x)+\alpha\circ\rho_{\downarrow\overline{r}}(y)+\alpha\circ\rho_{\uparrow\overline{s}}(x)\,|\,\alpha\circ\rho_{\downarrow\overline{r}}(y)\ \subseteq$$

(Use that $x\subseteq y\Rightarrow x\,|\,z\subseteq y\,|\,z$. Use further $x=\partial_{\hat{H}}(x)\overset{RN5}{=}\partial_H\circ\partial_{\hat{H}}(x)=\partial_H(x)$.)

$$\overset{AA1}{\subseteq}\ \alpha\circ\rho_{\uparrow\overline{s}}\circ\partial_H(x)+\alpha\circ\rho_{\downarrow\overline{r}}\circ\partial_H(y)+\boldsymbol{A}\,|\,\boldsymbol{A}\subseteq$$

[1] The *Fresh Atom Principle* (FAP) says that we can use new (or 'fresh') atomic actions in proofs. In [12], it is shown that FAP holds in bisimulation semantics. We have not included FAP in the theoretical framework of this chapter. Therefore, if we need certain 'fresh' atoms in a proof, we have to assume that they were in the alphabet right from the beginning.

(Use that $range(\gamma)\cap H=\varnothing$.)

$$\begin{aligned}
&\overset{RN5}{\subseteq}\ \alpha\circ\partial_H\circ\rho_{\uparrow\bar{s}}(x)+\alpha\circ\partial_H\circ\rho_{\downarrow\bar{r}}(y)+\boldsymbol{B}\subseteq\\
&\overset{AA3,RN4}{\subseteq}\ \partial_H\circ\alpha\circ\rho_{\uparrow\bar{s}}(x)+\partial_H\circ\alpha\circ\rho_{\downarrow\bar{r}}(y)+\boldsymbol{B}\subseteq
\end{aligned}$$

(Use that $x\subseteq y$ implies $\rho_f(x)\subseteq\rho_f(y)$.)

$$\overset{AA1}{\subseteq}\ \partial_H(\boldsymbol{A})+\partial_H(\boldsymbol{A})+\boldsymbol{B}=\boldsymbol{B}$$

This finishes the proof of the claim.

Note 2. Application of axiom AA1 gives: $\alpha\circ\rho_{\uparrow\bar{s}}(x)\subseteq\boldsymbol{A}$ and $\alpha\circ\rho_{\downarrow r}(y)\subseteq\boldsymbol{A}$. In order to apply axiom RR2, we first have to check that for all $c\in A$: $r\bar{r}(c)=r\bar{r}\circ r\bar{r}(c)$. This is obviously the case. Because $range(\gamma)\cap H=\varnothing$, we have for all $b,c\in A: r\bar{r}\circ\gamma(b,c)=\gamma(b,c)$. Now the last thing to be checked is that for $b,c\in A$: $\gamma(b,c)=\gamma(b,r\bar{r}(c))$. This turns out to be the case.

Note 3. Let $C=A-\overline{H}$. We claim: $\alpha(\rho_{\downarrow r}(y)\|\rho_{\uparrow s}(x))\subseteq\boldsymbol{C}$. The proof is similar to the proof in Note 1.

This finishes the proof of the lemma. □

3.2.1.2. THEOREM. SACP$_\tau$ + RN + CH$^+$ + AB + AA + RR ⊢

$$\frac{\partial_{\hat{H}}(x)=x,\ \partial_{\hat{H}}(y)=y,\ \partial_{\hat{H}}(z)=z}{x\ggg(y\ggg z)=(x\ggg y)\ggg z}$$

PROOF. This is essentially Theorem 1.12.2 of [118]. We give a sketch of the proof.

$$\begin{aligned}
x\ggg(y\ggg z)&=\partial_{\overline{H}}(\rho_{\uparrow\bar{s}}(x)\|\rho_{\downarrow\bar{r}}\circ\partial_{\underline{H}}(\rho_{\uparrow\underline{s}}(y)\|\rho_{\downarrow\underline{r}}(z)))=\\
&\overset{RN5}{=}\partial_{\overline{H}}(\rho_{\uparrow\bar{s}}(x)\|\partial_{\underline{H}}\circ\rho_{\downarrow\bar{r}}(\rho_{\uparrow\underline{s}}(y)\|\rho_{\downarrow\underline{r}}(z)))=\\
&\overset{RR1}{=}\partial_{\overline{H}}\circ\partial_{\underline{H}}(\rho_{\uparrow\bar{s}}(x)\|\partial_{\underline{H}}\circ\rho_{\downarrow\bar{r}}(\rho_{\uparrow\underline{s}}(y)\|\rho_{\downarrow\underline{r}}(z)))=\\
&\overset{RR2}{=}\partial_{\overline{H}}\circ\partial_{\underline{H}}(\rho_{\uparrow\bar{s}}(x)\|\rho_{\downarrow\bar{r}}(\rho_{\uparrow\underline{s}}(y)\|\rho_{\downarrow\underline{r}}(z)))=\\
&\overset{RR2}{=}\partial_{\overline{H}}\circ\partial_{\underline{H}}(\rho_{\uparrow\bar{s}}(x)\|\rho_{\downarrow\bar{r}}(\rho_{\downarrow\bar{r}}\circ\rho_{\uparrow\underline{s}}(y)\|\rho_{\downarrow\underline{r}}(z)))=\\
&\overset{RR1}{=}\partial_{\overline{H}}\circ\partial_{\underline{H}}(\rho_{\uparrow\bar{s}}(x)\|\rho_{\downarrow\bar{r}}\circ\rho_{\uparrow\underline{s}}(y)\|\rho_{\downarrow\underline{r}}(z))=\\
&\overset{RN5}{=}\partial_{\underline{H}}\circ\partial_{\overline{H}}(\rho_{\uparrow\bar{s}}(x)\|\rho_{\uparrow\underline{s}}\circ\rho_{\downarrow\bar{r}}(y)\|\rho_{\downarrow\underline{r}}(z))=\\
&\overset{RR1}{=}\partial_{\underline{H}}\circ\partial_{\overline{H}}(\rho_{\uparrow\underline{s}}(\rho_{\uparrow\bar{s}}(x)\|\rho_{\uparrow\underline{s}}\circ\rho_{\downarrow\bar{r}}(y))\|\rho_{\downarrow\underline{r}}(z))=\\
&\overset{RR2}{=}\partial_{\underline{H}}\circ\partial_{\overline{H}}(\rho_{\uparrow\underline{s}}(\rho_{\uparrow\bar{s}}(x)\|\rho_{\downarrow\bar{r}}(y))\|\rho_{\downarrow\underline{r}}(z))=
\end{aligned}$$

$$\begin{aligned}
&\overset{RR\,2}{=}\ \partial_{\underline{\underline{H}}}\circ\partial_{\overline{H}}(\partial_{\overline{H}}\circ\rho_{\uparrow\underline{\underline{s}}}(\rho_{\uparrow\overline{s}}(x)\|\rho_{\downarrow\overline{r}}(y))\|\rho_{\downarrow\underline{\underline{r}}}(z)) =\\
&\overset{RR\,1}{=}\ \partial_{\underline{\underline{H}}}(\partial_{\overline{H}}\circ\rho_{\uparrow\underline{\underline{s}}}(\rho_{\uparrow\overline{s}}(x)\|\rho_{\downarrow\overline{r}}(y))\|\rho_{\downarrow\underline{\underline{r}}}(z)) =\\
&\overset{RN\,5}{=}\ \partial_{\underline{\underline{H}}}(\rho_{\uparrow\underline{\underline{s}}}\circ\partial_{\overline{H}}(\rho_{\uparrow\overline{s}}(x)\|\rho_{\downarrow\overline{r}}(y))\|\rho_{\downarrow\underline{\underline{r}}}(z)) = (x\ggg y)\ggg z
\end{aligned}$$

□

3.2.1.3. THEOREM. SACP_τ + RN + CH^+ + AB + AA + RR ⊢

$$\frac{\partial_{\hat{H}}(x)=x,\ \partial_{\hat{H}}(y)=y,\ \partial_{\hat{H}}(z)=z}{x\gg(y\gg z) = (x\gg y)\gg z}$$

PROOF. Let $I=\{c(d)\,|\,d\in D\}$.

$$\begin{aligned}
x\gg(y\gg z) &\overset{CH\,2}{=}\ \tau_I(x\ggg(\tau_I(y\ggg z))) =\\
&\overset{CH\,1}{=}\ \tau_I\circ\partial_H(\rho_{\uparrow s}(x)\|\rho_{\downarrow r}\circ\tau_I(y\ggg z)) =\\
&\overset{RN\,5}{=}\ \partial_H\circ\tau_I(\rho_{\uparrow s}(x)\|\tau_I\circ\rho_{\downarrow r}(y\ggg z)) =\\
&\overset{RR\,2}{=}\ \partial_H\circ\tau_I(\rho_{\uparrow s}(x)\|\rho_{\downarrow r}(y\ggg z)) =\\
&\overset{RN\,5}{=}\ \tau_I\circ\partial_H(\rho_{\uparrow s}(x)\|\rho_{\downarrow r}(y\ggg z)) =\\
&\overset{CH\,1}{=}\ \tau_I(x\ggg(y\ggg z)) =\\
&\overset{3.2.1.2}{=}\ \tau_I((x\ggg y)\ggg z) = \cdots = (x\gg y)\gg z
\end{aligned}$$

□

3.2.2. Removing auxiliary atoms. We will now apply the module approach to remove completely the auxiliary atoms which were used in the definition of the chaining operators. What we want to obtain is a module where 'inside' the auxiliary atoms are used to define the chaining operator but where 'outside' they are no longer visible and moreover chaining is associative in general. Below we will employ the notation:

$$\sigma\Delta M \equiv (\Sigma(M)-\sigma)\square M.$$

Consider the module:

$$\begin{aligned}
\mathrm{CH}^- = {}&(\{\mathbb{F}: a\in P\,|\,a\in\hat{H}\}\cup\{\mathbb{F}:\rho_f: P\rightarrow P\,|\,f: A_{\tau\delta}\rightarrow A_{\tau\delta}\})\\
&\Delta(\mathrm{SACP}_\tau+\mathrm{RN}+\mathrm{CH}^+ +\mathrm{AB}+\mathrm{AA}+\mathrm{RR}).
\end{aligned}$$

This module cannot be used to prove any formula containing atoms in $\hat{H}$. But unfortunately module CH^- still does not prove the general associativity of the chaining operators:

$$\mathrm{CH}^- \nvdash x\ggg(y\ggg z)=(x\ggg y)\ggg z$$

The reason is that the auxiliary atoms, although removed from the language, are still present in the models of module CH$^-$. Thus the counterexample $(r(d) \ggg (s(d)+s(e))) \ggg r(e)$ still works in the models. Let $A^- = A - \hat{H}$. We are interested in consistent models which only contain actions of A^-. The module CH$^-$ $+ <\alpha(x) \subseteq \boldsymbol{A}^->$ does not denote such models: all consistent models of CH$^-$ contain the process $\boldsymbol{A}$ with $\alpha(\boldsymbol{A}) = \boldsymbol{A} \nsubseteq A^-$. Adding the law $\alpha(x) \subseteq \boldsymbol{A}^-$ therefore throws away all consistent models. The right class of models can be denoted with the help of operator S. We consider the module

$$\mathrm{CH} = S(\mathrm{CH}^-) + <\alpha(x) \subseteq \boldsymbol{A}^->.$$

Some models of module CH$^-$ have consistent submodels which do not contain auxiliary atoms. In these models the law $\alpha(x) \subseteq \boldsymbol{A}^-$ holds. Thus module CH has consistent models.

From Theorems 3.2.1.2 and 3.2.1.3, together with axiom RR1, it follows that:

$$CH^- \vdash \frac{\alpha(x) \subseteq \boldsymbol{A}^-,\ \alpha(y) \subseteq \boldsymbol{A}^-,\ \alpha(z) \subseteq \boldsymbol{A}^-}{(x \ggg y) \ggg z = x \ggg (y \ggg z)} \quad \text{and}$$

$$CH^- \vdash \frac{\alpha(x) \subseteq \boldsymbol{A}^-,\ \alpha(y) \subseteq \boldsymbol{A}^-,\ \alpha(z) \subseteq \boldsymbol{A}^-}{(x \gg y) \gg z = x \gg (y \gg z)}.$$

From this we can easily see that module CH proves the general associativity of the chaining operators:

$$\mathrm{CH} \vdash x \ggg (y \ggg z) = (x \ggg y) \ggg x \quad \text{and}$$

$$\mathrm{CH} \vdash x \gg (y \gg z) = (x \gg y) \gg x.$$

3.2.3. The following laws can be easily proven from module CH (here $d,e \in D$):

$$\uparrow d \cdot x \gg (\sum_{e \in D} \downarrow e \cdot y^e) = \tau \cdot (x \gg y^d) \qquad \text{L1}$$

$$\uparrow d \cdot x \gg \uparrow e \cdot y = \uparrow e \cdot (\uparrow d \cdot x \gg y) \qquad \text{L2}$$

$$(\sum_{d \in D} \downarrow d \cdot x^d) \gg (\sum_{e \in D} \downarrow e \cdot y^e) = \sum_{d \in D} \downarrow d \cdot (x^d \gg (\sum_{e \in D} \downarrow e \cdot y^e)) \qquad \text{L3}$$

$$(\sum_{d \in D} \downarrow d \cdot x^d) \gg \uparrow e \cdot y = \sum_{d \in D} \downarrow d \cdot (x^d \gg \uparrow e \cdot y) + \uparrow e \cdot ((\sum_{d \in D} \downarrow d \cdot x^d) \gg y) \qquad \text{L4}$$

The laws are equally valid when the operator $\gg$ is replaced by $\ggg$, except for law L1 where in addition the τ has to be replaced by $c(d)$.

3.3. SACP$^\sharp_\tau$. Module SACP$^\sharp_\tau$ is an 'improved' version of module ACP$^\sharp_\tau$. It is defined by:

$$\mathrm{SACP}^\sharp_\tau = \mathrm{SACP}_\tau + \mathrm{RN} + \mathrm{CH} + \mathrm{REC} + \mathrm{PR} + \mathrm{B} + \mathrm{AIP}^- + \mathrm{AB} + \mathrm{AA} + \mathrm{RR}.$$

If modules in the above defining equation have an alphabet as parameter, this is A^-, and if they are parametrised by a communication function this is the restriction γ^- of γ to $(A^- \cup \{\delta\}) \times (A^- \cup \{\delta\})$. The rules RSP, RSP$^+$ and CFAR can still be used in a setting with module SACP$^\sharp_\tau$. We have SACP$^\sharp_\tau$ $\vdash$ RSP, SACP$^\sharp_\tau$ $\vdash$ RSP$^+$ and SACP$^\sharp_\tau$ + KFAR $\vdash$ CFAR.

4. Queues

In the specification of concurrent systems FIFO queues with unbounded capacity often play an important role. We give some examples:

- The semantical description of languages with asynchronous message passing such as CHILL (see Recommendation Z.200 (CHILL language definition), CCITT Study Group XI, 1980),
- The modelling of communication channels occurring in computer networks (see Larsen & Milner [85] and Vaandrager [117]),
- The implementation of languages with many-to-one synchronous communication, such as POOL (see America [5] and Vaandrager [118]).

Consequently the questions how queues can be specified, and how one can prove properties of systems containing queues, are important. For a nice sample of queue-specifications we refer to the solutions of the first problem of the STL/SERC workshop [46]. Some other references are Broy [35], Hoare [76] and Pratt [108].

4.1. Also in the setting of ACP a lot of attention has been paid to the specification of queues. Below we give an infinite specification of the process behaviour of a queue. Here D is a finite set of data, D^* is the set of finite sequences σ of elements from D, the empty sequence is ϵ. Sequence $\sigma * \sigma'$ is the concatenation of sequences σ and σ'. The sequence, only consisting of $d \in D$ is denoted by d as well.

$$\boxed{\begin{array}{l} QUEUE = Q_\epsilon = \sum_{d \in D} \downarrow d \cdot Q_d \\ \\ Q_{\sigma * d} = \sum_{e \in D} \downarrow e \cdot Q_{e * \sigma * d} + \uparrow d \cdot Q_\sigma \end{array}}$$

Note that this infinite specification uses only the signature of BPA$_\delta$ (see Section 2.1). We have the following fact:

4.1.1. Theorem: *Using read/send communication, the process QUEUE cannot be specified in ACP by finitely many recursion equations.*
Proof: See Baeten & Bergstra [7] and Bergstra & Tiuryn [25]. □

It turns out that if one allows an arbitrary communication function, or extends the signature with an (almost) arbitrary additional operator, the process *QUEUE can* be specified by finitely many recursion equations. For some nice examples we refer to BERGSTRA & KLOP [22].

4.2. Definition of the queue by means of chaining. A problem we had with all ACP-specifications of the queue is that they are difficult to deal with in process verifications. For example, let $BUF1$ be a buffer with capacity one:

$$\boxed{\begin{aligned} BUF1 &= \sum_{d\in D} \downarrow d \cdot BUF1^d \\ BUF1^d &= \uparrow d \cdot BUF1 \end{aligned}}$$

In process verifications we need propositions like $QUEUE \gg BUF1 = QUEUE$ (in Section 5 we present a protocol verification where a similar fact is actually used). However, the proof of this fact starting from the infinite specification is rather complicated. Now the following specification of a queue by means of the (abstract) chaining operator allows for a simple proof of the proposition and numerous other useful identities involving queues. This specification is also described by HOARE [76] (p. 158).

$$\boxed{Q = \sum_{d\in D} \downarrow d \cdot (Q \gg BUF1^d)}$$

The first thing we have to prove is that the process described above really is a queue.

4.2.1. THEOREM: $Q = QUEUE$.

PROOF: Define for every $n \in \mathbb{N}$ and $\sigma = d_1, \ldots, d_m \in D^*$ processes D^n_σ as follows:

$$D^n_\sigma = Q \gg BUF1 \cdots_{n \text{ times}} \gg BUF1^{d_1} \cdots \gg BUF1^{d_m}$$

So by definition $D^0_\epsilon = Q$. Using the laws of Section 3.2.3, we derive the following recursion equations:

$$D^0_\epsilon = Q = \sum_{d\in D} \downarrow d \cdot (Q \gg BUF1^d) = \sum_{d\in D} \downarrow d \cdot D^0_d$$

$$\begin{aligned} D^n_{\sigma * d} &= Q \gg BUF1 \cdots_{n \text{ times}} \gg BUF1^{d_1} \cdots \gg BUF1^{d_m} \gg BUF1^d = \\ &= \sum_{e\in D} \downarrow e \cdot (Q \gg BUF1^e \gg BUF1 \cdots_{n \text{ times}} \gg BUF1^{d_1} \cdots \gg BUF1^{d_m} \gg BUF1^d) + \\ &\quad + \uparrow d \cdot (Q \gg BUF1 \cdots_{n \text{ times}} \gg BUF1^{d_1} \cdots \gg BUF1^{d_m} \gg BUF1) = \\ &\overset{note}{=} \sum_{e\in D} \downarrow e \cdot (Q \gg BUF1 \cdots_{n \text{ times}} \gg BUF1^e \gg BUF1^{d_1} \cdots \gg BUF1^{d_m} \gg BUF1^d) + \end{aligned}$$

$$+ \uparrow d \cdot (Q \gg BUF1 \cdots_{n+1 \text{ times}} \gg BUF1^{d_1} \cdots \gg BUF1^{d_m}) =$$
$$= \sum_{e \in D} \downarrow e \cdot D^n_{e*\sigma*d} + \uparrow d \cdot D^{n+1}_\sigma$$

Note. In the second last step we moved the data in the sequence of 1-datum-buffers to the right as far as possible. It is easy to see that this is allowed. Suppose that not all data are moved to the right. By applying the associativity of the chaining operator we can rewrite the expression in such a way that we get a subterm of the form $BUF1^d \gg BUF1$. This subterm can be rewritten into $\tau \cdot (BUF1 \gg BUF1^d)$. Next we move the initial τ to the front of the sequence using the identity $\tau x \| y = \tau(x \| y)$ of Proposition 2.1.3, and remove it by means of axiom T1 $(x\tau = x)$ of ACP_τ. Now we have moved one datum one place to the right in the queue. We can iterate this procedure until the desired result is obtained.

Define the process Q^0_ϵ by:

$$Q^0_\epsilon = \sum_{d \in D} \downarrow d \cdot Q^0_d$$
$$Q^n_{\sigma*d} = \sum_{e \in D} \downarrow e \cdot Q^n_{e*\sigma*d} + \uparrow d \cdot Q^{n+1}_\sigma$$

The specification of process Q^0_ϵ is clearly guarded. Applying RSP gives us on the one hand that $QUEUE = Q^0_\epsilon$, and on the other hand that $Q = D^0_\epsilon = Q^0_\epsilon$. Consequently $QUEUE = Q$. □

The proof above shows the 'view of a queue' that lies behind the specification of Q. During execution there is a long chain of 1-datum buffers passing messages from 'the left to the right'. After the input of a new datum on the left, a new buffer is created, containing the new datum and placed at the leftmost position in the chain. Because no buffer is ever removed from the system, the number of empty buffers increases after every output of a datum.

4.2.2. LEMMA: $Q \gg BUF1 = Q$.
PROOF:

$$Q \gg BUF1 = \sum_{d \in D} \downarrow d \cdot ((Q \gg BUF1^d) \gg BUF1) =$$
$$= \sum_{d \in D} \downarrow d \cdot (Q \gg (BUF1^d \gg BUF1)) =$$
$$= \sum_{d \in D} \downarrow d \cdot (Q \gg \tau \cdot (BUF1 \gg BUF1^d)) =$$
$$\overset{2.1.3}{=} \sum_{d \in D} \downarrow d \cdot (Q \gg (BUF1 \gg BUF1^d)) =$$
$$= \sum_{d \in D} \downarrow d \cdot ((Q \gg BUF1) \gg BUF1^d)$$

Now apply RSP$^+$ (from the proof of Theorem 4.2.1 it follows that Q is guardedly specifiable). □

By means of an inductive argument we can easily prove the following corollary of Lemma 4.2.2.

4.2.3. Corollary: *Let for $\sigma \in D^*$, Q^σ be a queue with content σ:*

$$\boxed{\begin{aligned} Q^\epsilon &= Q \\ Q^{\sigma * d} &= Q^\sigma \gg BUF1^d \end{aligned}}$$

Then: $\tau\cdot(Q^\sigma \gg BUF1) = \tau\cdot Q^\sigma$.

4.2.4. Proposition: $Q \gg Q = Q$.
Proof:

$$\begin{aligned} Q \gg Q &= \sum_{d\in D} \downarrow d\cdot((Q \gg BUF1^d) \gg Q) = \\ &= \sum_{d\in D} \downarrow d\cdot(Q \gg (BUF1^d \gg Q)) = \\ &= \sum_{d\in D} \downarrow d\cdot(Q \gg \tau\cdot(BUF1 \gg (Q \gg BUF1^d))) = \\ &\overset{2.1.3}{=} \sum_{d\in D} \downarrow d\cdot(Q \gg (BUF1 \gg (Q \gg BUF1^d))) = \\ &= \sum_{d\in D} \downarrow d\cdot((Q \gg BUF1) \gg (Q \gg BUF1^d)) = \\ &\overset{4.2.2}{=} \sum_{d\in D} \downarrow d\cdot(Q \gg (Q \gg BUF1^d)) = \\ &= \sum_{d\in D} \downarrow d\cdot((Q \gg Q) \gg BUF1^d) \end{aligned}$$

Now apply RSP$^+$. □

4.2.5. Corollary: *Let $\sigma,\rho \in D^*$. Then:* $\tau(Q^\sigma \gg Q^\rho) = \tau Q^{\sigma * \rho}$.

4.2.6. Remark. It will be clear that the implementation which is suggested by the specification of process Q is not very efficient: at each time the number of empty storage elements equals the number of data that have left the queue. But we can do it even more inefficiently: the following queue doubles the number of empty storage elements each time a datum is written.

$$\boxed{\bar{Q} = \sum_{d\in D} \downarrow d\cdot(\bar{Q} \gg \uparrow d\cdot\bar{Q})}$$

A standard proof gives that $\bar{Q} = QUEUE$. From the point of view of process algebra this specification is very efficient. It is the shortest specification of a FIFO-queue known to the authors, except for a 5-character specification of PRATT [108]: $\downarrow\uparrow \times D^{*}$. A problem with Pratt's specification is that a neat axiomatisation of the orthocurrence operator $\times$ is not available. Our $\overline{Q}$-specification has the disadvantage that it does not allow for simple proofs of identities like $\overline{Q} \gg \overline{Q} = \overline{Q}$.

4.3. Bags. In [18] a bag over data domain D is defined by:

$$\boxed{BAG = \sum_{d \in D} \downarrow d \cdot (\uparrow d \| BAG)}$$

4.3.1. THEOREM: $Q \gg BAG = BAG$.
PROOF:

$$\begin{aligned}
Q \gg BAG &= \sum_{d \in D} \downarrow d \cdot ((Q \gg \uparrow d \cdot BUF1) \gg BAG) = \\
&= \sum_{d \in D} \downarrow d \cdot (Q \gg (\uparrow d \cdot BUF1 \gg BAG)) = \\
&= \sum_{d \in D} \downarrow d \cdot (Q \gg \tau \cdot (BUF1 \gg (BAG \| \uparrow d))) = \\
&= \sum_{d \in D} \downarrow d \cdot (Q \gg (BUF1 \gg (BAG \| \uparrow d))) = \\
&= \sum_{d \in D} \downarrow d \cdot ((Q \gg BUF1) \gg (BAG \| \uparrow d)) = \\
&= \sum_{d \in D} \downarrow d \cdot (Q \gg (BAG \| \uparrow d)) = \\
&\overset{note}{=} \sum_{d \in D} \downarrow d \cdot ((Q \gg BAG) \| \uparrow d)
\end{aligned}$$

Now apply RSP.

Note. We claim that $\mathrm{SACP}_{\tau} + \mathrm{RN} + \mathrm{CH}^{+} + \mathrm{AB} + \mathrm{AA} + \mathrm{RR} \vdash (Q \gg (BAG \| \uparrow d)) = ((Q \gg BAG) \| \uparrow d)$. Let $I = \{c(d) \mid d \in D\}$ and $H = \{r(d), s(d) \mid d \in D\}$. Then:

$$(Q \gg (BAG \| \uparrow d)) = \tau_I \circ \partial_H(\rho_{\uparrow s}(Q) \| \rho_{\downarrow r}(BAG \| \uparrow d)) = ,$$

(straightforward application of axioms of AB + AA + RR + SC6)

$$\begin{aligned}
&= \tau_I \circ \partial_H(\rho_{\uparrow s}(Q) \| \rho_{\downarrow r}(BAG)) \| \uparrow d = \\
&= ((Q \gg BAG) \| \uparrow d)
\end{aligned}$$

From the claim it follows that $\mathrm{CH} \vdash (Q \gg (BAG \| \uparrow d)) = ((Q \gg BAG) \| \uparrow d)$ and consequently $\mathrm{SACP}_{\tau}^{\sharp} \vdash (Q \gg (BAG \| \uparrow d)) = ((Q \gg BAG) \| \uparrow d)$. □

4.3.2. Remark. The identity $BAG \gg Q = BAG$ does not hold. The intuitive argument for this is as follows: if a bag contains an apple and an orange, and the environment wants an apple, then it can just take this apple from the bag. In the case where a system, consisting of the chaining of a bag and a queue, contains an apple and an orange, it can occur that the first element in the queue is an orange. In this situation the environment *has* to take the orange first. The argument that processes $Q \gg BAG$ and BAG are different, because in the first process the environment is not able to pick an apple that is still in the queue, does not hold. In ACP_τ we abstract from the real-time behaviour of concurrent systems. If the environment waits long enough then the apple will be in the bag.

4.4. A queue that can lose data. In the specification of communication protocols, we often encounter transmission channels that can make errors: they can lose, damage or duplicate data. All process algebra specifications of these channels we have seen thus far were lengthy and often incomprehensible. Consequently it was difficult to prove properties of systems containing these queues. Now, interestingly, the same idea that was used to specify the normal queue by means of the chaining operator, can also be used to specify the various faulty queues. One just has to replace the process $BUF1$ in the definition by a process that behaves like a buffer but can lose, damage or duplicate data.

First we describe a queue FQ that can lose every datum contained in it at every moment, without any possibilities for the environment to prevent this from happening. The basic component of this queue is the following Faulty Buffer with capacity one:

$$\boxed{\begin{aligned} FBUF1 &= \sum_{d \in D} \downarrow d \cdot FBUF1^d \\ FBUF1^d &= (\uparrow d + \tau) \cdot FBUF1 \end{aligned}}$$

If the faulty buffer contains a datum, then this can get lost at any moment through the occurrence of a τ-action. In the equation for $FBUF1^d$ there is no τ-action before the $\uparrow d$-action because this would make it possible for the buffer to reach a state where datum d could not get lost.

We use the above specification in the definition of the faulty queue FQ:

$$\boxed{FQ = \sum_{d \in D} \downarrow d \cdot (FQ \gg FBUF1^d)}$$

The idea behind this specification of the faulty queue is illustrated in Figure 3.

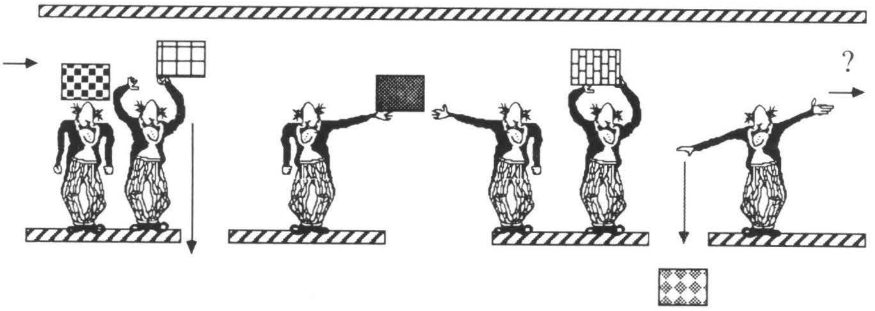


FIGURE 3. *The faulty queue*

4.4.1. LEMMA: $FBUF\,1^d \gg FBUF\,1 \;=\; \tau\cdot(FBUF\,1 \gg FBUF\,1^d)$.
PROOF:

$$FBUF\,1^d \gg FBUF\,1 \;=\; \tau\cdot(FBUF\,1 \gg FBUF\,1^d) + \tau\cdot(FBUF\,1 \gg FBUF\,1) =$$
$$= \tau\cdot(FBUF\,1 \gg FBUF\,1^d)$$

In the last step we use that:
$\tau\cdot(FBUF\,1 \gg FBUF\,1) \subseteq FBUF\,1 \gg FBUF\,1^d \subseteq \tau\cdot(FBUF\,1 \gg FBUF\,1^d)$. □

Compare the simple definition of FQ with the following BPA$_{\tau\delta}$-specification of the same process.

4.4.2. Let $\sigma,\rho \in D^*$. We write $\sigma \rightarrow \rho$ if ρ can be obtained from σ by deleting one datum. Let $R(\sigma) = \{\rho \mid \sigma \rightarrow \rho\}$ be the finite set of residues of σ after one deletion. Now $FQUEUE$ is the following process.

$$\boxed{\begin{aligned} &FQUEUE = FQ_\epsilon = \sum_{d\in D} \downarrow d\cdot FQ_d \\ &FQ_{\sigma * d} = \sum_{e\in D} \downarrow e\cdot FQ_{e*\sigma*d} + \uparrow d\cdot FQ_\sigma + \sum_{\rho\in R(\sigma*d)} \tau\cdot FQ_\rho \end{aligned}}$$

4.4.3. THEOREM: $FQ \;=\; FQUEUE$.
PROOF: Analogously to the proof of Theorem 4.2.1. Use Lemma 4.4.1. □

Analogous versions of the identities we derived for the normal queue can be derived for the faulty queue in the same way. In the proofs we use Lemma 4.4.1.

4.4.4. PROPOSITION:

i) $FQ \gg FBUF1 = FQ$,

ii) *Let for* $\sigma \in D^*$, FQ^σ *be a faulty queue with content* σ:

$$\boxed{\begin{aligned} FQ^\epsilon &= FQ \\ FQ^{\sigma * d} &= FQ^\sigma \gg FBUF1^d \end{aligned}}$$

Then: $\tau \cdot (FQ^\sigma \gg FBUF1) = \tau \cdot FQ^\sigma$,

iii) $FBUF1^d \gg FQ = \tau \cdot (FBUF1 \gg (FQ \gg FBUF1^d))$,

iv) $Q \gg FQ = FQ \gg FQ = FQ$,

v) *Let* $\sigma, \rho \in D^*$. *Then:* $\tau \cdot (FQ^\sigma \gg FQ^\rho) = \tau \cdot FQ^{\sigma * \rho}$.

4.5. An identity that does nót hold. In this subsection we will discuss the identity

$$FQ = Q \gg FBUF1.$$

'Intuitively' the processes FQ and $Q \gg FBUF1$ are equal since both behave like a FIFO-queue that can lose data. Furthermore, with both processes the environment cannot prevent in any way that a datum gets lost. Unlike the situation with the processes $BAG \gg Q$ and BAG which we discussed in Section 4.3, we can think of no 'experiment' that distinguishes between the two processes. Still the identity cannot be proved with the axioms presented thus far.

4.5.1. THEOREM: *If parameter D of operator* $\gg$ *contains more than one element, then* $SACP^{\#}_{\tau} \nvdash FQ = Q \gg FBUF1$.

PROOF: We show that the identity is not valid in the model of process graphs modulo bisimulation congruence as presented in BAETEN, BERGSTRA & KLOP [9]. Suppose that there exists a bisimulation between processes FQ and $Q \gg FBUF1$. Suppose that process FQ reads successively two different data, starting from the initial state. Because of the bisimulation it must be possible for the process $Q \gg FBUF1$ to read the same data in such a way that the resulting state is bisimilar to the state process FQ has reached. Now process FQ executes a τ-step and forgets the second datum. We claim that process $Q \gg FBUF1$ is not capable to perform a corresponding sequence of zero or more τ-step. This is because there are only two possibilities:

1) $Q \gg FBUF1$ forgets the second datum. But this means that also the first datum is forgotten. In the resulting state $Q \gg FBUF1$ cannot output any datum (before reading one), whereas process FQ can do this.
2) $Q \gg FBUF1$ does not forget the second datum. In the resulting state $Q \gg FBUF1$ can output this datum. Process FQ cannot do that.

The argument is illustrated in Figure 4.

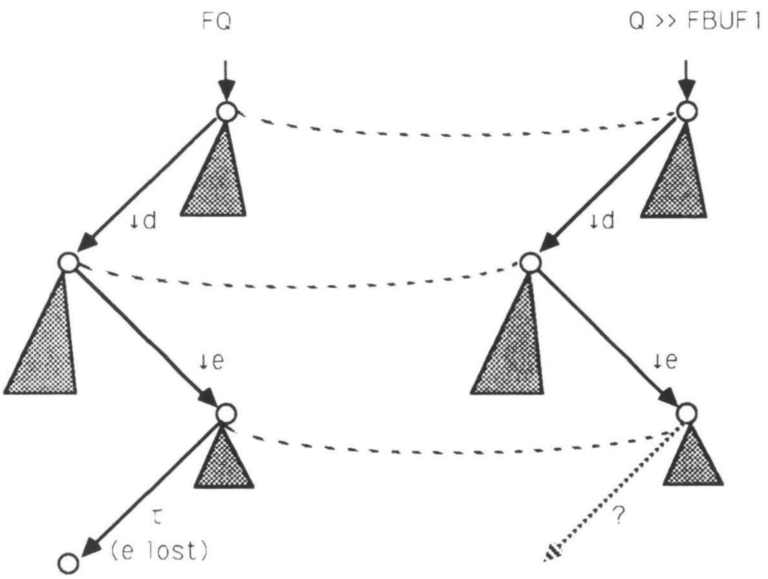


FIGURE 4.

The next theorem shows that, if we add law T4, the two faulty queues can be proven equivalent.

4.5.2. THEOREM: $SACP^{\sharp}_{\tau}$ + T4 ⊢ $FQ = Q \gg FBUF1$.
PROOF: Define the process QF by:

$$QF = QF_{\epsilon} = \sum_{d \in D} \downarrow d \cdot QF_{d}$$

$$QF_{\sigma * d} = \sum_{e \in D} \downarrow e \cdot QF_{e * \sigma * d} + (\uparrow d + \tau) \cdot QF_{\sigma}$$

Analogous to the proof of Theorem 4.2.1, using in addition the identity $Q \gg BUF1 = Q$, we prove that $Q \gg FBUF1 = QF$. For this we do not need the additional axiom.

The main trick in the proof is that we introduce yet another 'view of queues': process QF_{σ} is split in two parts, a read-process QF^{r}_{ϵ} and a write process QF^{w}_{σ}. The read-process takes care of reading new data, whereas the write process outputs the data in σ. When the write process is ready, it sends a message *ready* to the read-process and dies. When the read-process, after reading a sequence ρ of messages, receives the *ready*-signal it behaves again like process QF_{ρ}. The fact that the length of the sequence of data σ in QF^{w}_{σ} can only decrease, allows us to use induction.

We extend the alphabet[1] with actions *ready*, *ready** and $\overline{ready}$, and define

[1] See Note 1 in Section 3.2.1.

communication by $\gamma(ready,ready^*)=\overline{ready}$. For $\sigma\in D^*$ and $d\in D$ we define:

$$QF^r_\epsilon = \sum_{d\in D}\downarrow d\cdot QF^r_d$$

$$QF^r_{\sigma*d} = \sum_{e\in D}\downarrow e\cdot QF^r_{e*\sigma*d} + ready^*\cdot QF_{\sigma*d}$$

$$QF^s_\epsilon = ready$$

$$QF^s_{\sigma*d} = (\uparrow d + \tau)\cdot QF^s_\sigma$$

A short but nontrivial proof gives:

$$QF_\sigma = \tau_{\{\overline{ready}\}}\circ\partial_{\{ready,ready^*\}}(QF^r_\epsilon \| QF^s_\sigma)$$

Also in this step we do not use the extra axiom. We claim that: $\tau\cdot QF^s_{\sigma*\sigma'} \subseteq QF^s_{\sigma*d*\sigma'}$.

The proof of the claim goes with induction to the length of σ'. If $|\sigma'| = 0$ the the claim holds trivially. Now suppose the claim is proved for $|\sigma'|\leqslant n$. Choose $\overline{\sigma}$ with length n, and $e\in D$. We have that:

$$\tau\cdot QF^s_{\sigma*\overline{\sigma}*e} = \tau\cdot(\uparrow e\cdot QF^s_{\sigma*\overline{\sigma}} + \tau\cdot QF^s_{\sigma*\overline{\sigma}}) =$$

(this is the only step where we use axiom T4)

$$= \uparrow e\cdot QF^s_{\sigma*\overline{\sigma}} + \tau\cdot QF^s_{\sigma*\overline{\sigma}} \subseteq$$

(because on the one hand $\uparrow e\cdot QF^s_{\sigma*\overline{\sigma}} \subseteq \uparrow e\cdot QF^s_{\sigma*d*\overline{\sigma}}$ because of the induction hypothesis and axiom T3, and on the other hand $\tau\cdot QF^s_{\sigma*\overline{\sigma}} \subseteq \tau\cdot QF^s_{\sigma*d*\overline{\sigma}}$ because of the induction hypothesis and axiom T2)

$$\subseteq \uparrow e\cdot QF^s_{\sigma*d*\overline{\sigma}} + \tau\cdot QF^s_{\sigma*d*\overline{\sigma}} = QF^s_{\sigma*d*\overline{\sigma}*e}$$

This finishes the proof of the claim. A corollary is that $\tau\cdot QF_{\sigma*\sigma'} \subseteq QF_{\sigma*d*\sigma'}$:

$$\tau\cdot QF_{\sigma*\sigma'} = \tau\cdot\tau_{\{\overline{ready}\}}\circ\partial_{\{ready,ready^*\}}(QF^r_\epsilon \| QF^s_{\sigma*\sigma'}) \subseteq$$

(Use the observation of Section 2.1.4 that $\tau x \subseteq y$ implies $\tau x \| z \subseteq y \| z$)

$$\subseteq \tau_{\{\overline{ready}\}}\circ\partial_{\{ready,ready^*\}}(QF^r_\epsilon \| QF^s_{\sigma*d*\sigma'}) = QF_{\sigma*d*\sigma'}$$

We have shown that process QF_σ is indistinguishable from a process that can lose each datum at every moment. Using the notation of Section 4.4.2 we can write down the following equation for processes $QF_{\sigma*d}$:

$$QF_{\sigma*d} = \sum_{e\in D}\downarrow e\cdot QF_{e*\sigma*d} + \uparrow d\cdot QF_\sigma + \sum_{\rho\in R(\sigma*d)}\tau\cdot QF_\rho$$

Application of RSP gives that the process *FQUEUE* of Section 4.4.2 equals process *QF*. But according to Theorem 4.4.3 also *FQUEUE* = *FQ*. □

4.6. The faulty and damaging queue. In the specification of certain link layer protocols we have to deal with a communication channel that behaves like a FIFO-queue with unbounded capacity (this is of course a simplifying assumption), but has some additional properties: (1) a datum can be damaged at every moment it is in the queue; the environment cannot prevent this event, and (2) a datum can be lost at every moment it is in the queue. We give a process algebra specification of this process in two steps. First we specify the Faulty and Damaging Buffer with capacity one (FDBUF1). We assume that the domain of data D contains a special element er, representing a damaged datum.

$$\boxed{\begin{aligned} FDBUF1 &= \sum_{d\in D} \downarrow d\cdot FDBUF1^{d} \\ FDBUF1^{d} &= \uparrow d\cdot FDBUF1 \;+\; \tau\cdot(\uparrow er+\tau)\cdot FDBUF1 \end{aligned}}$$

With the help of this process we can now easily define the Faulty and Damaging Queue (FDQ):

$$\boxed{FDQ \;=\; \sum_{d\in D} \downarrow d\cdot(FDQ \gg FDBUF1^{d})}$$

4.6.1. Lemma: $FDBUF1^{d} \gg FDBUF1 \;=\; \tau\cdot(FDBUF1 \gg FDBUF1^{d})$.
Proof: $FDBUF1^{d} \gg FDBUF1 =$

$$\begin{aligned}
&= \tau\cdot(FDBUF1 \gg FDBUF1^{d}) \;+\; \tau\cdot((\uparrow er+\tau)\cdot FDBUF1 \gg FDBUF1) = \\
&= \tau\cdot(FDBUF1 \gg FDBUF1^{d}) \;+\; \tau\cdot(\tau\cdot(FDBUF1 \gg FDBUF1^{er}) \;+ \\
&\qquad +\; \tau\cdot(FDBUF1 \gg FDBUF1)) = \\
&\overset{note}{=} \tau\cdot(FDBUF1 \gg FDBUF1^{d}) \;+\; \tau\cdot(\tau\cdot(FDBUF1 \gg (\tau\cdot(\uparrow er+\tau)\cdot FDBUF1)) \;+ \\
&\qquad +\; \tau\cdot(FDBUF1 \gg FDBUF1)) = \\
&= \tau\cdot(FDBUF1 \gg FDBUF1^{d}) \;+\; \tau\cdot(\tau\cdot(FDBUF1 \gg (\uparrow er+\tau)\cdot FDBUF1) \;+ \\
&\qquad +\; \tau\cdot(FDBUF1 \gg FDBUF1)) = \\
&= \tau\cdot(FDBUF1 \gg FDBUF1^{d}) \;+\; \tau\cdot(FDBUF1 \gg (\uparrow er+\tau)\cdot FDBUF1) = \\
&= \tau\cdot(FDBUF1 \gg FDBUF1^{d})
\end{aligned}$$

Note. $FDBUF1^{er} \;=\; \uparrow er\cdot FDBUF1 \;+\; \tau\cdot(\uparrow er+\tau)\cdot FDBUF1 \;=$

$$\overset{T2}{=} \tau\cdot(\uparrow er+\tau)\cdot FDBUF1. \qquad \square$$

Once we have Lemma 4.6.1, it is standard to prove that process FDQ is guardedly specifiable. It is moreover easy to derive an analogous version of Proposition 4.4.4 for FDQ.

4.6.2. Remark. One might ask if there is not a τ too many in the specification of process $FDBUF1$. Why not specify the faulty and damaging buffer simply as follows?

$$\boxed{\begin{aligned} FDB1 &= \sum_{d \in D} \downarrow \cdot FDB1^d \\ FDB1^d &= (\uparrow d + \uparrow er + \tau) \cdot FDB1 \end{aligned}}$$

A first observation we make is that if $D \neq \{er\}$:

$$\mathrm{SACP}^{\#}_{\tau} \nvdash FDBUF1 = FDB1$$

This is because the two processes are different in bisimulation semantics. Process $FDBUF1$ can input a datum d different from er, and then get into a state where either an output action $\uparrow er$ will be performed or no output action at all. This means that it is possible that a datum is first damaged and then lost. Process $FDB1$ does not have such a state.

For similar reasons we also have the following fact:

$$\mathrm{SACP}^{\#}_{\tau} \nvdash FDB1^d \gg FDB1 = \tau \cdot (FDB1 \gg FDB1^d)$$

This means that if we work with a queue defined with the help of $FDB1$, our standard technique to prove facts about queues is not applicable. Note that processes $FDB1$ and $FDBUF1$ are trivially equal if we work in a setting where the law T4 ($\tau(\tau x + y) = \tau x + y$) is valid.

4.7. The faulty and stuttering queue. This section is about a very curious queue: a FIFO-queue that can lose or duplicate any element contained in it at every moment. An infinite specification of this process can be found in LARSEN & MILNER [85]. The basic component we use in the specification of the Faulty and Stuttering Queue is a Faulty and Stuttering Buffer with capacity 1:

$$\boxed{\begin{aligned} FSBUF1 &= \sum_{d \in D} \downarrow d \cdot FSBUF1^d \\ FSBUF1^d &= \uparrow d \cdot FSBUF1^d + \tau \cdot FSBUF1 \end{aligned}}$$

$$\boxed{FSQ = \sum_{d \in D} \downarrow d \cdot (FSQ \gg FSBUF1^d)}$$

When we place two faulty and stuttering buffers in a chain, then we have the possibility of an infinite number of internal actions (the first buffer stutters and the second one loses all its input). This implies that, in the specification of the faulty and stuttering queue, we have to guard against unguarded recursion. We need a fairness assumption if we want to exclude the possibility of infinite stuttering.

First we prove a simple lemma:

4.7.1. LEMMA: $FSBUF\,1^d \gg FSBUF\,1^d \;=\; \tau\cdot(FSBUF\,1^d \gg FSBUF\,1^d) \;=$
$= FSBUF\,1^d \gg FSBUF\,1 \;=\; \tau\cdot(FSBUF\,1^d \gg FSBUF\,1)$.
PROOF:

$$FSBUF\,1^d \gg FSBUF\,1^d \;\supseteq\; \tau\cdot(FSBUF\,1^d \gg FSBUF\,1) \;\supseteq\; FSBUF\,1^d \gg FSBUF\,1 \;\supseteq$$
$$\supseteq\; \tau\cdot(FSBUF\,1^d \gg FSBUF\,1^d) \;\supseteq\; FSBUF\,1^d \gg FSBUF\,1^d \qquad \square$$

The proof of the next lemma is more involved.

4.7.2. LEMMA:

$$\mathrm{SACP}^{\#}_{\tau} + \mathrm{KFAR} \vdash FSBUF\,1^d \gg FSBUF\,1 \;=\; \tau\cdot(FSBUF\,1 \gg FSBUF\,1^d).$$

PROOF:

$$FSBUF\,1^d \ggg FSBUF\,1 \;=$$
$$=\; c(d)\cdot(FSBUF\,1^d \ggg FSBUF\,1^d) \;+\; \tau\cdot(FSBUF\,1 \ggg FSBUF\,1)$$
$$FSBUF\,1^d \ggg FSBUF\,1^d \;=$$
$$=\; \tau\cdot(FSBUF\,1^d \ggg FSBUF\,1) \;+\; \tau\cdot(FSBUF\,1 \ggg FSBUF\,1^d) \;+$$
$$+\; \uparrow d\cdot(FSBUF\,1^d \ggg FSBUF\,1^d)$$

Application of CFAR gives ($I = \{c(d) \mid d \in D\}$):

$$FSBUF\,1^d \gg FSBUF\,1 \;=\; \tau_I(FSBUF\,1^d \ggg FSBUF\,1) \;=$$
$$=\; \tau\cdot\tau_I(\tau\cdot(FSBUF\,1 \ggg FSBUF\,1) \;+\; \tau\cdot(FSBUF\,1 \ggg FSBUF\,1^d) \;+$$
$$+\; \uparrow d\cdot(FSBUF\,1^d \ggg FSBUF\,1^d)) \;=$$
$$\overset{4.7.1}{=}\; \tau\cdot(\tau\cdot(FSBUF\,1 \gg FSBUF\,1) \;+\; \tau\cdot(FSBUF\,1 \gg FSBUF\,1^d) \;+$$
$$+\; \uparrow d\cdot(FSBUF\,1^d \gg FSBUF\,1))$$

In addition we derive:

$$FSBUF\,1 \gg FSBUF\,1^d \;=\; \sum_{e\in D} \downarrow e\cdot(FSBUF\,1^e \gg FSBUF\,1^d) \;+$$
$$+\; \tau\cdot(FSBUF\,1 \gg FSBUF\,1) \;+\; \uparrow d\cdot(FSBUF\,1 \gg FSBUF\,1^d)$$
$$FSBUF\,1^e \gg FSBUF\,1^d \;=\; \tau\cdot(FSBUF\,1 \gg FSBUF\,1^d) \;+$$
$$+\; \tau\cdot(FSBUF\,1^e \gg FSBUF\,1) \;+\; \uparrow d\cdot(FSBUF\,1^e \gg FSBUF\,1^d)$$
$$FSBUF\,1 \gg FSBUF\,1 \;=\; \sum_{d\in D} \downarrow d\cdot(FSBUF\,1^d \gg FSBUF\,1)$$

Let E be the following guarded system of recursion equations:

$$Y^{d.} \;=\; \tau\cdot(\tau\cdot Y \;+\; \tau\cdot Y^{.d} \;+\; \uparrow d\cdot Y^{d.})$$
$$Y^{.d} \;=\; \sum_{e\in D} \downarrow e\cdot Y^{ed} \;+\; \tau\cdot Y \;+\; \uparrow d\cdot Y^{.d}$$

$$Y^{ed} = \tau \cdot Y^{\cdot d} + \tau \cdot Y^{e\cdot} + \uparrow d \cdot Y^{ed}$$

$$Y = \sum_{d \in D} \downarrow d \cdot Y^{d\cdot}$$

RSP gives that $FSBUF1^{d} \gg FSBUF1 = Y^{d\cdot}$ and $FSBUF1 \gg FSBUF1^{d} = Y^{\cdot d}$. Thus it suffices to prove that $Y^{d\cdot} = \tau \cdot Y^{\cdot d}$. Let F be the following guarded system of recursion equations:

$$Z^{d} = \sum_{e \in D} \downarrow e \cdot Z^{ed} + \tau \cdot Z + \uparrow d \cdot Z^{d}$$

$$Z^{ed} = \tau \cdot Z^{d} + \tau \cdot Z^{e} + \uparrow d \cdot Z^{ed}$$

$$Z = \sum_{d \in D} \downarrow d \cdot Z^{d}$$

We derive:

$$\tau \cdot Z^{d} = \tau \cdot \tau \cdot Z^{d} = \tau \cdot (\tau \cdot Z + \uparrow d \cdot Z^{d} + \tau \cdot Z^{d})$$

If we substitute $\tau \cdot Z^{d}$ for $Y^{d\cdot}$, Z^{d} for $Y^{\cdot d}$, Z^{ed} for Y^{ed} and Z for Y, then RSP gives: $\tau \cdot Z^{d} = Y^{d\cdot}$ and $Z^{d} = Y^{\cdot d}$. Consequently $Y^{d\cdot} = \tau \cdot Y^{\cdot d}$. □

From Lemma 4.7.2 all the rest follows: process *FSQ* is guardedly specifiable and we can derive an analogous version of Proposition 4.4.4.

5. A protocol verification

In this section we present the specification and verification of a variant of the Alternating Bit Protocol, resembling the ones discussed in KOYMANS & MULDER [81] and LARSEN & MILNER [85]. The aim of this exercise is to illustrate the usefulness of the proof technique developed in the previous section. The architecture of the *Concurrent Alternating Bit Protocol (CABP)* is as follows:

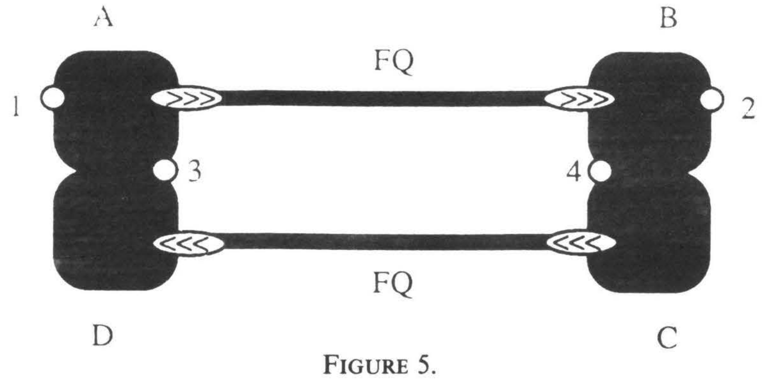


FIGURE 5.

Elements of a finite set of data are to be transmitted by the CABP from port 1 to port 2. Verification of the CABP amounts to a proof that (1) the protocol will eventually send at port 2 all and only data it has read at port 1, and (2) the protocol will send the data at port 2 in the same order as it has read them at port 1.

In the CABP sender and receiver send frames continuously. Since sender and receiver will have a different clock in general, the number of data that can be in the channels at a certain moment is in principle unlimited. In this section we assume that the channels behave like the process *FQ* as described in Section 4.4: a FIFO-queue with unbounded capacity that can either lose frames or pass them on correctly.

In the protocol, the sender consists of two components A and D, whereas the receiver consists of components B and C. One might propose to collapse A and D into a sender process, and B plus C into a receiver process. The resulting processes would be more complicated and in the correctness proof we would have to decompose them again.

5.1. Specification. Let D be a finite set of data which have to be sent by the CABP from port 1 to port 2. Let $B = \{0,1\}$. $\mathfrak{D} = (D \times B) \cup B$ is the set of data which occur as parameter in the actions of the chaining operators. The set of ports is $\mathbb{P} = \{1,2,3,4\}$, the set of data that can be communicated at these ports is $\mathbb{D} = D \cup \{next\}$. Alphabet A and communication function γ are now defined by the standard scheme for the chaining operators, augmented with actions $ri(d)$, $si(d)$ and $ci(d)$, for which we have communications $\gamma(ri(d), si(d)) = ci(d)$ ($i \in \mathbb{P}$ and $d \in \mathbb{D}$).

We now give the specifications of processes A, B, C and D. Here b ranges over $B = \{0,1\}$ and d over D (the overloading of names B and D should cause no confusion). The specifications are standard and need no further comment.

$A = A^0$ $A^b = \sum_{d \in D} r1(d) \cdot A^{db}$ $A^{db} = \uparrow db \cdot A^{db} + r3(next) \cdot A^{1-b}$	$B = B^0$ $B^b = \sum_{d \in D} \downarrow(d, 1-b) \cdot B^b + \sum_{d \in D} \downarrow db \cdot B^{db}$ $B^{db} = s2(d) \cdot s4(next) \cdot B^{1-b}$
$D = D^0$ $D^b = \downarrow(1-b) \cdot D^b +$ $+ \downarrow b \cdot s3(next) \cdot D^{1-b}$	$C = C^1$ (not C^0!) $C^b = \uparrow b \cdot C^b + r4(next) \cdot C^{1-b}$

Let H and I be the following sets of actions:

$$H = \{r3(next), s3(next), r4(next), s4(next)\}$$

$$I = \{c3(next), c4(next)\}$$

The Concurrent Alternating Bit Protocol is defined by:

$$\boxed{CABP = \tau_I \circ \partial_H((A \gg FQ \gg B) \| (C \gg FQ \gg D))}$$

5.2. Verification. If we do not abstract from the internal actions of the protocol, then the number of states is infinite. This means that a straightforward calculation of the state graph is not possible. A strategy which is often applied in cases like this is that one substitutes a buffer with capacity 1 for the communication channels. As a result the system is finite and can be verified automatically. Next a buffer with capacity 2 is substituted, followed by another automatic verification, etc.. The verification for the case of buffers with capacity 155 takes 23 hours CPU time. Thereafter it is decided that 'the protocol is correct'.

Of course it is not so difficult to specify a protocol that is correct for buffers with capacity less or equal than 155, but fails when the capacity is 156. The conclusion that the protocol is correct for arbitrary buffer size because it works in the cases where the buffer size is less than 156, is therefore influenced by other observations. It is for example intuitively not very plausible that the CABP works for buffer size 155, but not for buffer size 156, because the specification is so short and the only numbers which occur in it are 0 and 1.

Because intuitions can be wrong people look for formal techniques which tell in which situations induction over certain protocol parameters is allowed.

The basic merit of the results of Section 4 is that they make it possible to use inductive arguments when dealing with the length of queues in protocol systems. In the verification below we show that the protocol is correct if the channels behave as faulty FIFO-queues with unbounded capacity. However, a minor change in the proof is enough to show that the protocol also works if the channels behave as n-buffers, faulty n-buffers, perfect queues, faulty and stuttering queues, etc.

The following two lemmas will be used to show that, after abstraction, the number of states of the protocol is finite. The first lemma says that if, at the head of the queue, there is a datum that will be thrown away by the receiver because it is of the wrong type, this datum can be thrown away immediately.

5.2.1. LEMMA:
i) $FBUF1^{db} \gg B^{1-b} = \tau \cdot (FBUF1 \gg B^{1-b})$;
ii) $FBUF1^{db} \gg s4(next) \cdot B^{1-b} = \tau \cdot (FBUF1 \gg s4(next) \cdot B^{1-b})$;
iii) $FBUF1^{db} \gg B^{db} = \tau \cdot (FBUF1 \gg B^{db})$.

PROOF: The proof of (i) is trivial. Part (ii) goes as follows:

$$\begin{aligned} &FBUF1^{db} \gg s4(next) \cdot B^{1-b} = \\ &\qquad = \tau \cdot (FBUF1 \gg s4(next) \cdot B^{1-b}) + s4(next) \cdot (FBUF1^{db} \gg B^{1-b}) = \\ &\qquad \overset{(i)}{=} \tau \cdot (FBUF1 \gg s4(next) \cdot B^{1-b}) + s4(next) \cdot (FBUF1 \gg B^{1-b}) = \\ &\qquad = \tau \cdot (FBUF1 \gg s4(next) \cdot B^{1-b}) \text{ (summand inclusion)} \end{aligned}$$

The proof of part (iii) is similar:

$$\begin{aligned}
FBUF1^{db} \gg B^{db} &= \tau\cdot(FBUF1 \gg B^{db}) + s2(d)\cdot(FBUF1^{db} \gg s4(next)\cdot B^{1-b}) = \\
&\overset{(ii)}{=} \tau\cdot(FBUF1 \gg B^{db}) + s2(d)\cdot(FBUF1 \gg s4(next)\cdot B^{1-b}) = \\
&= \tau\cdot(FBUF1 \gg B^{db})
\end{aligned}$$

□

The next lemma says that if two frames, of a type that the receiver is willing to accept, are at the head of the queue, one of these can be deleted without changing the process (modulo an initial τ).

5.2.2. LEMMA: $FBUF1^{db} \gg FBUF1^{db} \gg B^{b} = \tau\cdot(FBUF1 \gg FBUF1^{db} \gg B^{b})$.
PROOF: $FBUF1^{db} \gg FBUF1^{db} \gg B^{b} =$

$$\begin{aligned}
&= \tau\cdot(FBUF1 \gg FBUF1^{db} \gg B^{b}) + \tau\cdot(FBUF1^{db} \gg FBUF1 \gg B^{b}) + \\
&\qquad + \tau\cdot(FBUF1^{db} \gg FBUF1 \gg B^{db}) = \\
&\overset{4.4.1}{=} \tau\cdot(FBUF1 \gg FBUF1^{db} \gg B^{b}) + \tau\cdot(FBUF1 \gg FBUF1^{db} \gg B^{b}) + \\
&\qquad + \tau\cdot(FBUF1 \gg FBUF1^{db} \gg B^{db}) = \\
&\overset{5.2.1}{=} \tau\cdot(FBUF1 \gg FBUF1^{db} \gg B^{b}) + \tau\cdot(FBUF1 \gg FBUF1 \gg B^{db}) = \\
&= \tau\cdot(FBUF1 \gg FBUF1^{db} \gg B^{b})
\end{aligned}$$

□

5.2.3. We will now derive a transition diagram for process $A \ggg FQ \gg B$. In the derivation we use Lemmas 5.2.1 and 5.2.2 to keep the diagram finite. Furthermore we stop the derivation at those places where an action is performed that corresponds to the acknowledgement of a frame that has not yet arrived. The result of the calculations is presented in Figure 6. The grey arcs correspond to places where we stopped the derivation.

$$A \ggg FQ \gg B = A^{0} \ggg FQ \gg B^{0} \tag{0}$$

$$A^{b} \ggg FQ \gg B^{b} = \sum_{d \in D} r1(d)\cdot(A^{db} \ggg FQ \gg B^{b}) \tag{1}$$

$$\begin{aligned}
A^{db} \ggg FQ \gg B^{b} &= c(db)\cdot(A^{db} \ggg FQ \gg FBUF1^{db} \gg B^{b}) + \\
&\quad + r3(next)\cdot(A^{1-b} \ggg FQ \gg B^{b})
\end{aligned} \tag{2}$$

$$\begin{aligned}
A^{db} \ggg FQ \gg FBUF1^{db} \gg B^{b} &= \\
&c(db)\cdot(A^{db} \ggg FQ \gg FBUF1^{db} \gg FBUF1^{db} \gg B^{b}) + \\
&+ \tau\cdot(A^{db} \ggg FQ \gg FBUF1 \gg B^{b}) + \\
&+ \tau\cdot(A^{db} \ggg FQ \gg FBUF1 \gg B^{db}) + \\
&+ r3(next)\cdot(A^{1-b} \ggg FQ \gg FBUF1^{db} \gg B^{b}) =
\end{aligned} \tag{3}$$

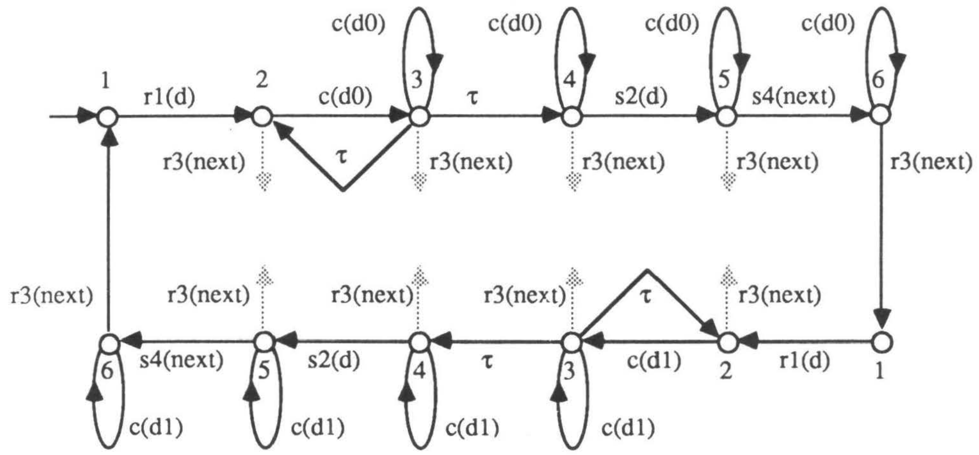


FIGURE 6. *Transition diagram of process* $A \ggg FQ \gg B$

(Lemmas 5.2.2 and 4.4.4(i))

$$= c(db)\cdot(A^{db} \ggg FQ \gg FBUF1^{db} \gg B^{b}) +$$
$$+\ \tau\cdot(A^{db} \ggg FQ \gg B^{b}) + \tau\cdot(A^{db} \ggg FQ \gg B^{db}) +$$
$$+\ r3(next)\cdot(A^{1-b} \ggg FQ \gg FBUF1^{db} \gg B^{b})$$

$$A^{db} \ggg FQ \gg B^{db} = c(db)\cdot(A^{db} \ggg FQ \gg FBUF1^{db} \gg B^{db}) + \tag{4}$$
$$+\ s2(d)\cdot(A^{db} \ggg FQ \gg s4(next)\cdot B^{1-b}) +$$
$$+\ r3(next)\cdot(A^{1-b} \ggg FQ \gg B^{db}) =$$

(Lemmas 5.2.1(iii) and 4.4.4(i))

$$= c(db)\cdot(A^{db} \ggg FQ \gg B^{db}) +$$
$$+\ s2(d)\cdot(A^{db} \ggg FQ \gg s4(next)\cdot B^{1-b}) +$$
$$+\ r3(next)\cdot(A^{1-b} \ggg FQ \gg B^{db})$$

$$A^{db} \ggg FQ \gg s4(next)\cdot B^{1-b} = \tag{5}$$
$$c(db)\cdot(A^{db} \ggg FQ \gg FBUF1^{db} \gg s4(next)\cdot B^{1-b}) +$$
$$+\ s4(next)\cdot(A^{db} \ggg FQ \gg B^{1-b}) +$$
$$+\ r3(next)\cdot(A^{1-b} \ggg FQ \gg s4(next)\cdot B^{1-b}) =$$

(Lemmas 5.2.1(ii) and 4.4.4(i))

$$= c(db)\cdot(A^{db} \ggg FQ \gg s4(next)\cdot B^{1-b}) +$$

$$+\ s4(next)\cdot(A^{db} \ggg FQ \gg B^{1-b}) +$$
$$+\ r3(next)\cdot(A^{1-b} \ggg FQ \gg s4(next)\cdot B^{1-b})$$

$$A^{db} \ggg FQ \gg B^{1-b} = c(db)\cdot(A^{db} \ggg FQ \gg FBUF1^{db} \gg B^{1-b}) + \qquad (6)$$
$$+\ r3(next)\cdot(A^{1-b} \ggg FQ \gg B^{1-b})$$

(Lemmas 5.2.1(i) and 4.4.4(i))

$$= c(db)\cdot(A^{db} \ggg FQ \gg B^{1-b}) + r3(next)\cdot(A^{1-b} \ggg FQ \gg B^{1-b})$$

5.2.4. Summarising, we have shown that $A \ggg FQ \gg B$ satisfies the following system of recursion equations.

$$\boxed{\begin{aligned}
X &= X_1^0 \\
X_1^b &= \sum_{d \in D} r1(d)\cdot X_2^{db} \\
X_2^{db} &= c(db)\cdot X_3^{db} + Y_2^b \\
&\qquad Y_2^b = r3(next)\cdot(A^{1-b} \ggg FQ \gg B^b) \\
X_3^{db} &= \tau\cdot X_2^{db} + c(db)\cdot X_3^{db} + \tau\cdot X_4^{db} + Y_3^{db} \\
&\qquad Y_3^{db} = r3(next)\cdot(A^{1-b} \ggg FQ \gg FBUF1^{db} \gg B^b) \\
X_4^{db} &= c(db)\cdot X_4^{db} + s2(d)\cdot X_5^{db} + Y_4^{db} \\
&\qquad Y_4^{db} = r3(next)\cdot(A^{1-b} \ggg FQ \gg B^{db}) \\
X_5^{db} &= c(db)\cdot X_5^{db} + s4(next)\cdot X_6^{db} + Y_5^{db} \\
&\qquad Y_5^{db} = r3(next)\cdot(A^{1-b} \ggg FQ \gg s4(next)\cdot B^{1-b}) \\
X_6^{db} &= r3(next)\cdot X_1^{1-b} + c(db)\cdot X_6^{db}
\end{aligned}}$$

Using CFAR immediately gives the next lemma.

5.2.5. LEMMA: *Let U be specified by:*

$U = U_1^0$	
$U_1^b = \sum_{d \in D} r\,1(d) \cdot U_2^{db}$	
$U_2^{db} = \tau \cdot U_4^{db} + V_2^b + V_3^{db}$	$V_2^b = r\,3(\mathit{next}) \cdot (A^{1-b} \gg FQ \gg B^b)$
	$V_3^{db} = r\,3(\mathit{next}) \cdot (A^{1-b} \gg FQ \gg FBUF\,1^{db} \gg B^b)$
$U_4^{db} = s\,2(d) \cdot U_5^{db} + V_4^{db}$	$V_4^{db} = r\,3(\mathit{next}) \cdot (A^{1-b} \gg FQ \gg B^{db})$
$U_5^{db} = s\,4(\mathit{next}) \cdot U_6^{db} + V_5^{db}$	$V_5^{db} = r\,3(\mathit{next}) \cdot (A^{1-b} \gg FQ \gg s\,4(\mathit{next}) \cdot B^{1-b})$
$U_6^{db} = r\,3(\mathit{next}) \cdot U_1^{1-b}$	

Then: $\mathrm{SACP}_\tau^\sharp + \mathrm{KFAR} \vdash U = A \gg FQ \gg B$.

In the same way we can derive similar lemmas for 'the other side' of the protocol.

5.2.6. LEMMA:
i) $FBUF\,1^b \gg D^{1-b} = \tau \cdot (FBUF\,1 \gg D^{1-b})$;
ii) $FBUF\,1^b \gg s\,3(\mathit{next}) \cdot D^{1-b} = \tau \cdot (FBUF\,1 \gg s\,3(\mathit{next}) \cdot D^{1-b})$;
iii) $FBUF\,1^b \gg FBUF\,1^b \gg D^b = \tau \cdot (FBUF\,1 \gg FBUF\,1^b \gg D^b)$.

5.2.7. LEMMA: *Let W be specified by:*

$W = W_1^1$	
$W_1^b = \tau \cdot r\,4(\mathit{next}) \cdot W_2^{1-b}$	$Z_1^b = r\,4(\mathit{next}) \cdot (C^{1-b} \gg FQ \gg D^b)$
$W_2^b = \tau \cdot W_3^b + Z_1^b + Z_2^b$	$Z_2^b = r\,4(\mathit{next}) \cdot (C^{1-b} \gg FQ \gg FBUF\,1^b \gg D^b)$
$W_3^b = s\,3(\mathit{next}) \cdot W_1^b + Z_3^b$	$Z_3^b = r\,4(\mathit{next}) \cdot (C^{1-b} \gg FQ \gg s\,3(\mathit{next}) \cdot D^{1-b})$

Then: $\mathrm{SACP}_\tau^\sharp + \mathrm{KFAR} \vdash C \gg FQ \gg D = W$.

The fact that CABP is a correct protocol is asserted by

5.2.8. THEOREM: $\mathrm{SACP}_\tau^\sharp + \mathrm{KFAR} \vdash CABP = \tau \cdot (\sum_{d \in D} r\,1(d) \cdot s\,2(d)) \cdot CABP$.

PROOF: Lemmas 5.2.5 and 5.2.7 together give that we can write *CABP* as:

$$CABP = \tau_I \circ \partial_H(U \| W)$$

A straightforward expansion gives:

$$\tau_I \circ \partial_H(U \| W) = \tau \cdot (\sum_{d \in D} r1(d) \cdot s2(d)) \cdot (\sum_{e \in D} r1(e) \cdot s2(e)) \cdot \tau_I \circ \partial_H(U \| W)$$

The variables V and Z vanish in the expansion, due to the fact that they only occur in situations where a receiver component sends a premature acknowledgement. An application of RSP concludes the proof of the theorem. □

5.2.9. Remark. A serious problem that has to be faced in the context of algebraic protocol verification is the fairness issue. In the verifications of this chapter we used KFAR to deal with fairness. KFAR is the algebraic equivalent of the statement: 'if anything can go well infinitely often, it will go well infinitely often'. In most applications a more subtle treatment of fairness is desirable. Moreover KFAR is incompatible with lots of semantics between bisimulation and trace semantics. In [23] it is proved that failure semantics is inconsistent with the rule KFAR. In the same paper a restricted version KFAR$^-$ of KFAR is presented which *is* consistent with the axioms of failure semantics, but this version is not powerful enough to allow for a verification of the CABP. The argument for this is simple: KFAR$^-$ allows for the fair abstraction of *unstable divergence*. This means that a process will never stay forever in a conservative cluster of internal τ-steps if it can be exited by another internal τ-step. Since in the CABP component C can always perform an internal step, and since the protocol is finite state (after suitable abstraction), there must be a conservative cluster of internal steps which can only be exited by performing an observable action. Thus the CABP contains stable divergence.

Conclusions and open problems

In this chapter we presented a language making it possible to give modular specifications of process algebras. The language contains operations $+$ and □, which are standard in the theory of structured algebraic specifications, and moreover two new operators H and S. Two applications have been presented of the new operators: we showed how the left-merge operator can be hidden if this is needed and we described how the chaining operator can be defined in a clean way in terms of more elementary operators. It is clear that there are much more applications of our approach. Numerous other process combinators can be defined in terms of more elementary operators in the same way as we did with the chaining operators. Maybe also other model theoretic operations can be used in a process algebra setting (cartesian products?).

Strictly speaking we have not introduced a 'module algebra' as in [17]: we do not interpret module expressions in an algebra. However, this can be done without any problem. An interesting topic of research is to look for axioms to manipulate module expressions. Due to the presence of the operators H and S, an elimination theorem for module expressions as in [17] will probably not be achievable.

An important open problem for us is the question whether the proof system of Table 1 is complete for first order logic.

In this chapter the modules are parametrised by a set of actions. These actions themselves do not have any structure. The most natural way to look towards actions like $s\,1(d_0)$ however, is to see them as actions parametrised by data. We would like to include the notion of a parametrised action in our framework but it turns out that this is not trivial. Related work in this area has been done by MAUW [87] and MAUW & VELTINK [88].

In order to prove the associativity of the chaining operators, we needed auxiliary actions $\overline{s}(d)$, $\overline{r}(d)$, etc. Also in other situations it often turns out to be useful to introduce auxiliary actions in verifications. At present we have to introduce these actions right at the beginning of a specification. This is embarrassing for a reader who does not know about the future use of these actions in the verification. But of course also the authors don't like to rewrite their specification all the time when they work on the verification. Therefore we would like to have a proof principle saying that it is allowed to use 'fresh' atomic actions in proofs. We think that it is possible to add a 'Fresh Atom Principle' (FAP) to our formal setting, but some work still has to be done.

In our view Section 4 convincingly shows that chaining operators are useful in dealing with FIFO-queues. We think that in general it will be often the case that a new application requires new operators and laws.

In Section 4.5 we presented a simple example of a realistic situation where bisimulation semantics does not work: a FIFO-queue which can loose data at every place is different from a FIFO-queue which can only loose data at the end. Adding the law T4, which holds in ready trace semantics (and hence in failure semantics), made it possible to prove the two queues equal.

For the correctness of protocols which involve faulty queues one normally needs some fairness assumption. Koomen's Fair Abstraction Rule (KFAR) often forms an adequate, although not optimal, way to model fairness. An interesting open problem is therefore the question whether the module $\mathrm{SACP}_{\tau}^{\sharp}$ + T4 + KFAR is consistent (conjecture: yes).

The verification of the Concurrent Alternating Bit Protocol as presented here takes 4 pages (or 5 if the proofs of the standard facts about the queues are included). Our proof is considerably shorter than the proof of similar protocols in [81] and [85] (15 and 11 pages respectively). But maybe this comparison is not altogether fair because the proofs in these papers were meant as an illustration of new modular proof techniques. Our proof shows that the axioms of bisimulation semantics with fair abstraction are sufficient for the modular verification of simple protocols like this. The axioms of bisimulation semantics will turn out to be not sufficient for more substantial modular verifications because bisimulation semantics is not fully abstract. We could give a shorter and simpler proof of the protocol by using the notion of redundancy in context of [119]: the grey arcs in Figure 6 all correspond to summands which are redundant in the context in which they occur. Additional proof techniques will certainly be needed for the modular verification of more complex protocols.

ACKNOWLEDGEMENTS

Our thanks to Jan Bergstra for his help in the development of the H-operator and to Kees Middelburg for helpful comments on an earlier version.

APPENDIX: LOGICS

In this appendix equational, conditional equational and first order logic are defined. Since all these logics share the concepts of variables and terms, these will be treated first.

1. Variables and terms. Let σ be a signature. A *σ-variable* is an expression x_S with $x \in$ NAMES and $(\mathbb{S}{:}S) \in \sigma$. A *valuation* of the σ-variables in a σ-algebra $\mathcal{A}$ is a function ξ that takes every σ-variable x_S into an element of $S^{\mathcal{A}}$.

For any $(\mathbb{S}{:}S) \in \sigma$ the set T_S^σ of *σ-terms* of sort S is defined inductively by:

- $x_S \in T_S^\sigma$ for any σ-variable x_S.
- If $\mathbb{F}{:}f{:}S_1 \times \cdots \times S_n \rightarrow S$ is in σ and $t_i \in T_{S_i}^\sigma$ for $i = 1,...,n$ then $f_{S_1 \times \cdots \times S_n \rightarrow S}(t_1,...,t_n) \in T_S^\sigma$.

The *ξ-evaluation* $[\![t]\!]^\xi \in S^{\mathcal{A}}$ of a σ-term $t \in T_S^\sigma$ in a σ-algebra $\mathcal{A}$ (with ξ a valuation) is defined by:

- $[\![x_S]\!]^\xi = \xi(x_S) \in S^{\mathcal{A}}$.
- $[\![f_{S_1 \times \cdots \times S_n \rightarrow S}(t_1,...,t_n)]\!]^\xi = f^{\mathcal{A}}_{S_1 \times \cdots \times S_n \rightarrow S}([\![t_1]\!]^\xi,...,[\![t_n]\!]^\xi)$.

2. Equational logic. The set F_σ^{eql} of *equations* or *equational formulas* over σ is defined by:

- If $t_i \in T_S^\sigma$ for $i = 1,2$ and certain $\mathbb{S}{:}S$ in σ, then $(t_1 = t_2) \in F_\sigma^{eql}$.

An equation $(t_1 = t_2) \in F_\sigma^{eql}$ is *ξ-true* in a σ-algebra $\mathcal{A}$, notation $\mathcal{A},\xi \models_\sigma^{eql} t_1 = t_2$, if $[\![t_1]\!]^\xi = [\![t_2]\!]^\xi$.
Such an equation $\phi \in F_\sigma^{eql}$ is *true* in $\mathcal{A}$, notation $\mathcal{A} \models_\sigma^{eql} \phi$, if $\mathcal{A},\xi \models_\sigma^{eql} \phi$ for all valuations ξ.

An inference system I_σ^{eql} for equational logic is displayed in Table 12 below. There t, u and v are terms over σ and x is a variable. Furthermore $t[u/x]$ is the result of substituting u for all occurrences of x in t. Of course u and x should be of the same sort. Finally an inference rule $\dfrac{H}{\phi}$ with $H = \varnothing$ is called an *axiom* and denoted simply by ϕ.

$t = t$	$\dfrac{u = v}{v = u}$	$\dfrac{t = u,\ u = v}{t = v}$	$\dfrac{u = v}{t[u/x] = t[v/x]}$	$\dfrac{u = v}{u[t/x] = v[t/x]}$

TABLE 12

3. Conditional equational logic. The set F_σ^{at} of *atomic formulas* over σ is defined by:

- If $t_i \in T_S^\sigma$ for $i=1,2$ and certain $\mathbb{S}{:}S$ in σ, then $(t_1=t_2) \in F_\sigma^{at}$.
- If $\mathbb{R}{:}p \subseteq S_1 \times \cdots \times S_n$ is in σ and $t_i \in T_{S_i}^\sigma$ for $i=1,\ldots,n$ then $p_{S_1 \times \cdots \times S_n}(t_1,\ldots,t_n) \in F_\sigma^{at}$.

The set F_σ^{ceql} of *conditional equational formulas* over σ is defined by:

- If $C \subseteq F_\sigma^{at}$ and $\alpha \in F_\sigma^{at}$ then $(C \Rightarrow \alpha) \in F_\sigma^{ceql}$.

The *ξ-truth* of formulas $\phi \in F_\sigma^{at} \cup F_\sigma^{ceql}$ in a σ-algebra $\mathcal{A}$ is defined by:

- $\mathcal{A},\xi \models_\sigma^{ceql} t_1=t_2$ if $[\![t_1]\!]^\xi = [\![t_2]\!]^\xi$.
- $\mathcal{A},\xi \models_\sigma^{ceql} p_{S_1 \times \cdots \times S_n}(t_1,\ldots,t_n)$ if $p^{\mathcal{A}}_{S_1 \times \cdots \times S_n}([\![t_1]\!]^\xi,\ldots,[\![t_n]\!]^\xi)$.
- $\mathcal{A},\xi \models_\sigma^{ceql} C \Rightarrow \alpha$ if $\mathcal{A},\xi \not\models_\sigma^{ceql} \beta$ for some $\beta \in C$ or $\mathcal{A},\xi \models_\sigma^{ceql} \alpha$.

ϕ is *true* in $\mathcal{A}$, notation $\mathcal{A} \models_\sigma^{ceql} \phi$, if $\mathcal{A},\xi \models_\sigma^{ceql} \phi$ for all valuations ξ.

An inference system I_σ^{ceql} for conditional equational logic is displayed in Table 13 below. There α and α_i are atomic formulas, C is a set of atomic formulas, ϕ is a conditional equational formula, t_i, t, u and v are terms over σ and x_i and x are variables. Furthermore $\alpha[u/x]$ is the result of substituting u for all occurrences of x in α. Of course u and x should be of the same sort. Likewise $\phi[t_i/x_i\ (i \in I)]$ is the result of simultaneous substitution for $i \in I$ of t_i for all occurrences of x_i in ϕ. An inference rule $\dfrac{\varnothing}{\phi}$ is again denoted by ϕ and a conditional equational formula $\varnothing \Rightarrow \alpha$ by α.

$C \Rightarrow \alpha$ if $\alpha \in C$	$\dfrac{C \Rightarrow \alpha_i\ (i \in I),\ \{\alpha_i \mid i \in I\} \Rightarrow \alpha}{C \Rightarrow \alpha}$	$\dfrac{\phi}{\phi[t_i/x_i\ (i \in I)]}$
$t=t$	$\{u=v\} \Rightarrow (v=u)$	$\{t=u,\ u=v\} \Rightarrow (t=u)$
	$\{u=v,\ \alpha[u/x]\} \Rightarrow (\alpha[v/x])$	

TABLE 13

The logic described above is *infinitary conditional equational logic*. *Finitary conditional equational logic* is obtained by the extra requirement that in conditional equational formulas $C \Rightarrow \alpha$ the set of conditions C should be finite. In that case the inference rule

$$\frac{\phi}{\phi[t_i/x_i\ (i \in I)]} \quad \text{can be replaced by} \quad \frac{\phi}{\phi[t/x]}.$$

Furthermore *(in)finitary conditional logic* is obtained by omitting all reference to the equality predicate $=$.

4. First order logic. The set F_σ^{foleq} of *first order formulas with equality* over σ is defined by:

- If $t_i \in T_S^\sigma$ for $i=1,2$ and certain $\mathbb{S}:S$ in σ, then $(t_1=t_2)\in F_\sigma^{foleq}$.
- If $\mathbb{R}:p \subseteq S_1 \times \cdots \times S_n$ is in σ and $t_i \in T_{S_i}^\sigma$ for $i=1,...,n$ then $p_{S_1\times\cdots\times S_n}(t_1,...,t_n) \in F_\sigma^{foleq}$.
- If $\phi \in F_\sigma^{foleq}$ then $\neg\phi \in F_\sigma^{foleq}$.
- If ϕ and $\psi \in F_\sigma^{foleq}$ then $(\phi\rightarrow\psi)\in F_\sigma^{foleq}$.
- If ϕ and $\psi \in F_\sigma^{foleq}$ then $(\phi\wedge\psi)\in F_\sigma^{foleq}$.
- If ϕ and $\psi \in F_\sigma^{foleq}$ then $(\phi\vee\psi)\in F_\sigma^{foleq}$.
- If ϕ and $\psi \in F_\sigma^{foleq}$ then $(\phi\leftrightarrow\psi)\in F_\sigma^{foleq}$.
- If x_S is a σ-variable and $\phi\in F_\sigma^{foleq}$ then $\forall x_S(\phi)\in F_\sigma^{foleq}$.
- If x_S is a σ-variable and $\phi\in F_\sigma^{foleq}$ then $\exists x_S(\phi)\in F_\sigma^{foleq}$.

The *ξ-truth* of a formula $\phi\in F_\sigma^{foleq}$ in a σ-algebra $\mathcal{A}$ is defined inductively by:

- $\mathcal{A},\xi \models_\sigma^{foleq} t_1=t_2$ if $[\![t_1]\!]^\xi=[\![t_2]\!]^\xi$.
- $\mathcal{A},\xi \models_\sigma^{foleq} p_{S_1\times\cdots\times S_n}(t_1,\cdots,t_n)$ if $p_{S_1\times\cdots\times S_n}^{\mathcal{A}}([\![t_1]\!]^\xi,...,[\![t_n]\!]^\xi)$.
- $\mathcal{A},\xi \models_\sigma^{foleq} \neg\phi$ if $\mathcal{A},\xi \not\models_\sigma^{foleq} \phi$.
- $\mathcal{A},\xi \models_\sigma^{foleq} \phi\rightarrow\psi$ if $\mathcal{A},\xi \not\models_\sigma^{foleq} \phi$ or $\mathcal{A},\xi \models_\sigma^{foleq} \psi$.
- $\mathcal{A},\xi \models_\sigma^{foleq} \phi\wedge\psi$ if $\mathcal{A},\xi \models_\sigma^{foleq} \phi$ and $\mathcal{A},\xi \models_\sigma^{foleq} \psi$.
- $\mathcal{A},\xi \models_\sigma^{foleq} \phi\vee\psi$ if $\mathcal{A},\xi \models_\sigma^{foleq} \phi$ or $\mathcal{A},\xi \models_\sigma^{foleq} \psi$.
- $\mathcal{A},\xi \models_\sigma^{foleq} \phi\leftrightarrow\psi$ if $\mathcal{A},\xi \models_\sigma^{foleq} \phi$ if and only if $\mathcal{A},\xi \models_\sigma^{foleq} \psi$.
- $\mathcal{A},\xi \models_\sigma^{foleq} \forall x_S(\phi)$ if $\mathcal{A},\xi' \models_\sigma^{foleq} \phi$ for all valuations ξ' with $\xi'(y_{S'})=\xi(y_{S'})$ for all variables $y_{S'}\neq x_S$.
- $\mathcal{A},\xi \models_\sigma^{foleq} \exists x_S(\phi)$ if $\mathcal{A},\xi' \models_\sigma^{foleq} \phi$ for some valuation ξ' with $\xi'(y_{S'})=\xi(y_{S'})$ for all variables $y_{S'}\neq x_S$.

ϕ is *true* is $\mathcal{A}$, notation $\mathcal{A} \models_\sigma^{foleq} \phi$, if $\mathcal{A},\xi \models_\sigma^{foleq} \phi$ for all valuations ξ.

An inference system I_σ^{foleq} for first order logic with equality is displayed in Table 14 on the next page. There ϕ, ψ and ρ are elements of F_σ^{foleq}, α is an atomic formula (constructed by means of the first two clauses in the definition of F_σ^{foleq} only), t, u and v are terms over σ and x is a variable. An occurrence of a variable x in a formula ϕ is *bound* if it occurs in a subformula $\forall x(\psi)$ or $\exists x(\psi)$ of ϕ. Otherwise it is *free*. A variable is *free* in a formula ϕ if all its occurrences in ϕ are free. $\phi[t/x]$ denotes the result of substituting u for all free occurrences of x in t. Of course u and x should be of the same sort. Now t *is free for* x *in* ϕ if all free occurrences of variables in t remain free in $\phi[t/x]$. As before an inference rule $\dfrac{H}{\phi}$ with $H=\varnothing$ is called an *axiom* and denoted simply by ϕ.

$\dfrac{\phi,\ \phi\to\psi}{\psi}$ *modus ponens*		$\dfrac{\phi}{\forall x(\phi)}$ *generalisation*
$\phi\to(\psi\to\phi)$		*deduction axioms*
$\{\phi\to(\psi\to\rho)\}\to\{(\phi\to\psi)\to(\phi\to\rho)\}$		
$\{\forall x(\phi\to\psi)\}\to\{\phi\to\forall x(\psi)\}$, if x is not free in ϕ		
$(\neg\phi\to\phi)\to\phi$		*axiom of the excluded middle*
$\neg\phi\to(\phi\to\psi)$		*axiom of contradiction*
$\forall x(\phi)\to\phi[t/x]$, if t is free for x in ϕ		*axiom of specialisation*
$(\phi\wedge\psi)\to\phi$	$\phi\to(\phi\vee\psi)$	$(\phi\leftrightarrow\psi)\to\{(\phi\to\psi)\wedge(\psi\to\phi)\}$
$(\phi\wedge\psi)\to\psi$	$\psi\to(\phi\vee\psi)$	$\{(\phi\to\psi)\wedge(\psi\to\phi)\}\to(\phi\leftrightarrow\psi)$
$\phi\to\{\psi\to(\phi\wedge\psi)\}$	$(\phi\vee\psi)\to(\neg\phi\to\psi)$	$\exists x(\phi)\leftrightarrow\neg\forall x(\neg\phi)$
$t=t$	$(u=v)\to(v=u)$	$\{(t=u)\wedge(u=v)\}\to(t=v)$
	$(u=v)\to(\alpha[u/x]\leftrightarrow\alpha[v/x])$	

TABLE 14

First order logic is obtained from first order logic with equality by omitting all reference to $=$. It is also possible to present first order logic without the connectives $\wedge$, $\vee$ and $\leftrightarrow$ and the quantifier $\exists$, and introduce them as notational abbreviations. In that case the third block of Table 13 can be omitted.

5. Expressiveness. One can translate an equation $\alpha\in F_\sigma^{eql}$ by a (finitary) conditional equational formula $\varnothing\Rightarrow\alpha$ and a finitary conditional equational formula $\{\alpha_1,...,\alpha_n\}\Rightarrow\alpha$ into a first order formula $(\alpha_1\wedge\cdots\wedge\alpha_n)\to\alpha$. Using this translation we have $F_\sigma^{eql}\subset F_\sigma^{fceql}\subset F_\sigma^{foleq}$ and furthermore $\mathcal{A}\models_\sigma^{eql}\phi \Leftrightarrow \mathcal{A}\models_\sigma^{ceql}\phi$ for $\phi\in F_\sigma^{eql}$ and $\mathcal{A}\models_\sigma^{ceql}\phi \Leftrightarrow \mathcal{A}\models_\sigma^{foleq}\phi$ for $\phi\in F_\sigma^{fceql}$. This means that first order logic with equality is more expressive then equational logic and finitary conditional equational logic is somewhere in between. However first order logic with equality and infinitary conditional equational logic have incomparable expressive power.

6. Completeness. For all logics mentioned above the following completeness result is known to hold: $Alg(\sigma,T)\models_\sigma^{\mathcal{L}}\phi \Rightarrow T\vdash_\sigma^{\mathcal{L}}\phi$. The reverse direction also holds, since all these logics are obviously sound. As a corollary we have

$$\begin{aligned} &T\vdash_\sigma^{eql}\phi \Leftrightarrow T\vdash_\sigma^{ceql}\phi &&\text{for } \phi\in F_\sigma^{eql} \text{ and}\\ &T\vdash_\sigma^{ceql}\phi \Leftrightarrow T\vdash_\sigma^{foleq}\phi &&\text{for } \phi\in F_\sigma^{fceql}. \end{aligned}$$

For this reason in most process algebra papers it is not made explicit which logic is used in verifications: the space needed for stating this could be saved, since the resulting notion of provability would be the same anyway. However, the situation changes when formulas are proved from modules. Equational logic and conditional equational logic are not complete anymore and for first order logic with equality this is still an open problem (as far as we know). Here a logic $\mathcal{L}$ is complete if $M \models^{\mathcal{L}} \phi \Rightarrow M \vdash^{\mathcal{L}} \phi$. It is easily shown that

$$M \vdash^{eql} \phi \Rightarrow M \vdash^{ceql} \phi \qquad \text{for } \phi \in F^{eql}_{\Sigma(M)} \text{ and}$$
$$M \vdash^{ceql} \phi \Rightarrow M \vdash^{foleq} \phi \qquad \text{for } \phi \in F^{fceql}_{\Sigma(M)},$$

but the reverse directions do not hold. Thus we should state exactly in which logic our results are proved.

7. Notation. This chapter employs infinitary conditional equational logic. However, no proof trees are constructed; proofs are given in a slightly informal way, that allows a straightforward translation into formal proofs by the reader. Furthermore all type information given in the subscripts of variables, function and predicate symbols is omitted, since confusion about the correct types is almost impossible. Outside Section 1 and this appendix inference rules $\frac{H}{\phi}$ do not occur, but all conditional equational formulas $C \Rightarrow \alpha$ are written $\frac{C}{\alpha}$, as is usual. However, the suggested similarity between inference rules and conditional equational formulas is misleading: $\frac{H}{\phi}$ holds in an algebra $\mathcal{A}$ if ($\mathcal{A}, \xi \models \psi$ for all $\psi \in H$ and all valuations ξ) implies ($\mathcal{A}, \xi \models \phi$ for all valuations ξ), while $\frac{C}{\alpha}$ holds in $\mathcal{A}$ if for all valuations ξ: ($\mathcal{A}, \xi \models \beta$ for all $\beta \in C$ implies $\mathcal{A}, \xi \models \alpha$).

8. Positive and universal formulas. In equational logic all formulas are both positive and universal. In conditional equational logic all formulas are universal and the positive formulas are the atomic ones. In first order logic with equality the positive formulas are the ones without the connectives $\neg$ and $\rightarrow$ and the universal ones are the formulas without quantifiers. Model theory (see for instance [98]) teaches us that a formula ϕ is preserved under homomorphisms (respectively subalgebras) iff there is a positive (respectively universal) formula ψ with $\vdash^{foleq} \psi \leftrightarrow \phi$.

Chapter III

Branching time and abstraction in bisimulation semantics

Rob van Glabbeek & Peter Weijland

Abstract: In comparative concurrency semantics one usually distinguishes between *linear time* and *branching time* semantic equivalences. Milner's notion of *observation equivalence* is often mentioned as the standard example of a branching time equivalence. In this chapter we investigate whether observation equivalence really does respect the branching structure of processes, and find that in the presence of the unobservable action τ of CCS this is not the case.

Therefore the notion of *branching bisimulation equivalence* is introduced which strongly preserves the branching structure of processes, in the sense that it preserves computations together with the potentials in all intermediate states that are passed through, even if silent moves are involved. On closed CCS-terms branching bisimulation can be completely axiomatized by single axiom scheme:

$$a.(\tau.(y + z) + y) = a.(y + z)$$

(where a ranges over all actions) and the usual laws for strong congruence.

For a large class of processes it turns out that branching bisimulation and observation equivalence are the same. All protocols known to the authors that have been verified in the setting of observation equivalence happen to fit in this class, and hence are also valid in the stronger setting of branching bisimulation equivalence.

Notes: An extended abstract of this chapter has been published in the Proceedings of the IFIP 11th World Computer Congress, San Fransisco 1989, pp. 613-618; the last section appeared in the Proceedings of the AMAST Conference, Iowa 1989, pp. 197-201. Moreover, the first two sections partly appeared in the Ph.D. Thesis of Peter Weijland [127].

The Research of the authors was supported by ESPRIT project 432 (METEOR). The first author was also supported by Sonderforschungsbereich 342 of the TU Muenchen, and the second author by ESPRIT project 3006 (CONCUR).

TABLE OF CONTENTS

INTRODUCTION

When comparing semantic equivalences for concurrency, it is common practice to distinguish between *linear time* and *branching time* equivalences (see for instance DE BAKKER, BERGSTRA, KLOP & MEYER [13], PNUELI [106]). In the former, a process is determined by its possible executions, whereas in the latter also the branching structure of processes is taken into account. The standard example of a linear time equivalence is *trace equivalence* as employed in HOARE [75]; the standard example of a branching time equivalence is *observation equivalence* or *bisimulation equivalence* as defined by MILNER [92] and PARK [103] (cf. MILNER [94-96]). Furthermore, there are several *decorated trace equivalences* in between (see Chapter I), preserving part of the branching structure of processes but for the rest resembling trace equivalence.

Originally, the most popular argument for employing branching time semantics was the fact that it allows a proper modelling of *deadlock behaviour*, whereas linear time semantics does not. However, this advantage is shared with the decorated trace semantics which have the additional advantage of only distinguishing between processes that can be told apart by some notion of *observation* or *testing*. The main criticism on observation equivalence - and branching time equivalences in general - is that it is not an observational equivalence in that sense: distinctions between processes are made that cannot be observed or tested, unless observers are equipped with extraordinary abilities like that of a copying facility together with the capability of global testing as in ABRAMSKY [1].

Nevertheless, branching time semantics is of fundamental importance in concurrency, exactly because it is independent of the precise nature of observability. Which one of the decorated trace equivalences provides a suitable modelling of observable behaviour depends in great extend on the tools an observer has, to test processes. And in general a protocol verification in a particular decorated trace semantics, does not carry over to a setting in which observers are a bit more powerful. On the other hand, branching time semantics preserves the internal branching structure of processes and thus certainly their observable behaviour as far as it can be captured by decorated traces. A protocol, verified in branching time semantics, is automatically valid in each of the decorated trace semantics.

Probably one of the most important features in process algebra is that of abstraction, since it provides us with a mechanism to *hide* actions that are not observable, or not interesting for any other reason. By abstraction, some of the actions in a process are made *invisible* or *silent*. Consequently, any consecutive execution of hidden steps cannot be recognized since we simply do not 'see' anything happen.

Algebraically, in ACP$_\tau$ of BERGSTRA & KLOP [20] abstraction has the form of a renaming operator which renames actions into a *silent move* called τ. In MILNER's CCS [92] these silent moves result from synchronization. This new constant τ is introduced in the algebraic models as well: for instance in the *graph models* (cf. [20,92]) we find the existence of τ-edges, and so the question was how to find a satisfactory extension of the original definition of *bisimulation equivalence* that we had on process graphs without τ.

It turns out that there exist many possibilities for extending bisimulation equivalence to process graphs with τ-steps. One such possible extension is incorporated in Milner's notion of *observation equivalence* - called *τ-bisimulation equivalence* in [20] -, which resembles ordinary bisimulation, but permits arbitrary sequences of τ-steps to precede or follow corresponding atomic actions. A different notion of so-called *η-bisimulation* was suggested by BAETEN & VAN GLABBEEK [11] invoking a weaker set of abstraction axioms. In MILNER [93] another notion of observational equivalence was introduced which in this chapter is referred to as *delay bisimulation equivalence*. As we will show, the treatments of Milner and Beaten & Van Glabbeek fit into a natural structure of four possible variations of bisimulation equivalence involving silent steps. The structure is completed by defining *branching bisimulation equivalence*. As it turns out, observation equivalence is the coarsest equivalence of the four, in the sense of identifying most processes. η- and delay bisimulation equivalence are two incomparable finer notions whereas branching bisimulation equivalence is the finest of all.

In a certain sense the usual notion of observation equivalence does not preserve the branching structure of a process. For instance, the processes $a\cdot(\tau\cdot b + c)$ and $a\cdot(\tau\cdot b + c) + a\cdot b$ are observation equivalent. However, in the first term, in each computation the choice between b and c is made after the a-step, whereas the second term has a computation in which b is already chosen when the a-step occurs. For this reason one may wonder whether or not we should accept the so-called third τ-law - $a\cdot(\tau\cdot x + y) = a\cdot(\tau\cdot x + y) + ax$ - (responsible for the former equivalence) and for similar reasons the second - $\tau\cdot x = \tau\cdot x + x$.

The previous example shows us that while preserving observation equivalence, we can introduce new paths in a graph that were not there before. To be precise: the *traces* are the same, but the sequences of intermediate nodes are different (modulo observation equivalence), since in the definition of observation equivalence there is no restriction whatsoever on the nature of the nodes that are passed through during the execution of a sequence of τ-steps, preceding or following corresponding atomic actions. This is the key point in our definition of branching bisimulation equivalence: in two bisimilar processes every computation in the one process corresponds to a computation in the other, in such a way that all intermediate states of these computations correspond as well, due to the bisimulation relation. It turns out that it can be defined by a small change in the definition of observation equivalence.

The fact that observation equivalence is too rigid in its identifications is even stronger illustrated by the problems that it may cause in practical applications and analysis. As pointed out by GRAF & SIFAKIS [66] there is no modal logic with eventually operator $\blacklozenge$ which is adequate for observation equivalence. Here $\blacklozenge\phi$ means that all paths will eventually pass to a state were ϕ holds, and a logic $\mathcal{L}$ is *adequate* for an equivalence $\approx$ if $(\forall\phi\in\mathcal{L}: (p \models \phi \Leftrightarrow q \models \phi)) \Leftrightarrow p \approx q$. Indeed, suppose that such a logic *would* exist, then this means that two processes are observation equivalent iff they satisfy the same modal formulas. Thus, with respect to processes in CCS there exists a formula f such that: (c.nil + τ.b.nil) $\models \phi$ and b.nil $\not\models \phi$ since obviously both processes are not observation equivalent. However, from (c.nil + τ.b.nil) $\models \phi$ it follows that we have a.(c.nil + τ.b.nil) $\models \blacklozenge\phi$ whereas from b.nil $\not\models \phi$ we find a.(c.nil + τ.b.nil) + a.b.nil $\not\models \blacklozenge\phi$ although both processes are observation equivalent. Obviously, this inconsistency is due to the third τ-law.

Another paper by JONSSON & PARROW [78] on deciding bisimulation equivalence shows a different kind of struggle with the third τ-law (as was pointed out to us by VAANDRAGER [121]). In this paper, infinite value passing is turned into a finite state representation by considering symbolic transitions. This provides us with a method to decide on the equivalence of certain infinite-state programs. It turns out to work easily for strong equivalence, but in observation equivalence there is no straightforward generalization of the former results and a less intuitive transition system is needed to fix this problem. Using branching bisimulation may serve as a key to a more natural solution of this problem.

Finally, if the actions a,b,c ... are not required to be atomic, one may want that two equivalent processes remain equivalent after replacing actions my more complex processes. In a setting with parallel operators this requirement leads to non-interleaving equivalences (Chapter VI), but also in a setting with only sequential processes this requirement is not met by observation equivalence, as can be seen by replacing a by $a_1.a_2$ in the third τ-law mentioned above. Again, branching bisimulation does not suffer from this problem, as will be demonstrated in Section 6.

Having at least four options for the definition of bisimulation congruence involving τ-steps, in any particular application it becomes important to have a clear intuition about which kind of abstraction is preferable. In an important class of problems one can prove however, that all four notions of bisimulation yield the same equivalence. In particular this is the case if one of the two bisimulating graphs does not have any τ-steps. It is interesting to observe that all case studies on protocol verification known to the authors fit into this class of problems, hence all of their proofs that have been given in the setting of observation equivalence still hold in branching bisimulation semantics .

1. BRANCHING AND ABSTRACTION

In this section we define the semantic equivalences that we want to discuss on a domain of *process graphs*. Since we focus on branching and abstraction, we have chosen to abstain from a proper modelling of divergence, concurrency, real-time behaviour and stochastical aspects of processes. Moreover, we will disregard the nature of the actions that our processes may perform: they will be modelled as uninterpreted symbols a,b,c,... from a given set Act. We have chosen process graphs (or labelled transition systems) to represent processes, since they clearly visualize the aspects of the modelled systems' behaviour we are interested in. The nodes in our graphs (or states in our transition systems) remain anonymous. A common alternative is to use closed expressions in a process specification language like CCS or ACP as nodes in process graphs, but here we prefer to separate the semantic issues from the treatment of a particular language. In the next section, however, we will give an interpretation of certain subsets of CCS and ACP in (parts of) the graph model and discuss the algebraic aspects of our equivalences.

DEFINITION 1.1 A *process graph* is a connected, rooted, edge-labelled and directed graph.

In an edge-labelled directed graph, edges go from one node to another (or the same) node and are labelled with elements from a certain set Act. One can have more than one edge between two nodes as long as they carry different labels. A rooted graph has one special node which is indicated as the root node. We require process graphs to be connected: they need not be finite, but one must be able to reach every node from the root node by following a finite path. If r and s are nodes in a graph, then $r \xrightarrow{a} s$ denotes an edge from r to s with label a or it will be used as a *proposition* saying that such an edge exists. Process graphs represent concurrent systems in the following way: the elements of Act are *actions* a system may perform; the nodes of a process graph represent the states of a concurrent system; the root is the initial state and if $r \xrightarrow{a} s$, then the system can evolve from state r to state s by performing an action a. The domain of process graphs will be denoted by ***G***.

On ***G*** we consider the notion of *bisimulation equivalence*, which originally was due to PARK [103] and used in MILNER [94-96] and in a different formulation already in MILNER [92]. On the domain of process graphs, a bisimulation usually is defined as a relation $R \subseteq \text{nodes}(g) \times \text{nodes}(h)$ on the nodes of graphs g and h satisfying:

i. The roots of g and h are related by R

ii. If R(r,s) and $r \xrightarrow{a} r'$, then there exists a node s' such that $s \xrightarrow{a} s'$ and R(r',s')

iii. If R(r,s) and $s \xrightarrow{a} s'$, then there exists a node r' such that $r \xrightarrow{a} r'$ and R(r',s').

Equivalently - as is done in this chapter - one can obtain bisimulation equivalence from a *symmetric* relation R between nodes of g and h, only satisfying (i) and (ii). Such a symmetric relation can be defined as a relation $R \subseteq \text{nodes}(g) \times \text{nodes}(h) \cup \text{nodes}(h) \times \text{nodes}(g)$ such that $R(r,s) \Leftrightarrow R(s,r)$, or alternatively, as a set of unordered pairs of nodes $R \subseteq \{\{r,s\}: r \in \text{nodes}(g),\ s \in \text{nodes}(h)\}$. In the latter case R(r,s) abbreviates $\{r,s\} \in R$. Note that this restriction to symmetric relations does not cause any loss of generality.

DEFINITION 1.2 Two graphs g and h in ***G*** are *bisimilar* - notation: $g \underline{\leftrightarrow} h$ - if there exists a symmetric relation R between the nodes of g and h (called a *bisimulation*) such that:

i. The roots of g and h are related by R

ii. If R(r,s) and $r \xrightarrow{a} r'$, then there exists a node s' such that $s \xrightarrow{a} s'$ and R(r',s').

Bisimilarity turns out to be an equivalence relation on ***G*** which is called *bisimulation equivalence*. Depending on the context we will sometimes use Milner's terminology and refer to bisimulation equivalence as *strong equivalence* or *strong congruence*.

Now let us postulate the existence of a special action $\tau \in \text{Act}$, that represents an unobservable, internal move of a process. We write $r \Rightarrow s$ for a path from r to s consisting of an arbitrary number (≥ 0) of τ-steps.

The definition of strong congruence was the starting point of MILNER [92] when he considered abstraction in CCS. Having in mind that τ-steps are not observable, he suggested to simply require that for g and h to be equivalent, (i) every possible a-step ($a \neq \tau$) in the one graph should correspond with an a-step in the other (as for usual bisimulation equivalence), apart from some arbitrary long sequences of τ-steps that are allowed to precede or follow, and (ii) every τ-step should correspond to an arbitrary long (≥ 0) τ-sequence. This way he obtained his notion of *observation equivalence* (cf. MILNER [92,94-96]) - or *τ-bisimulation equivalence* - which can be defined as follows:

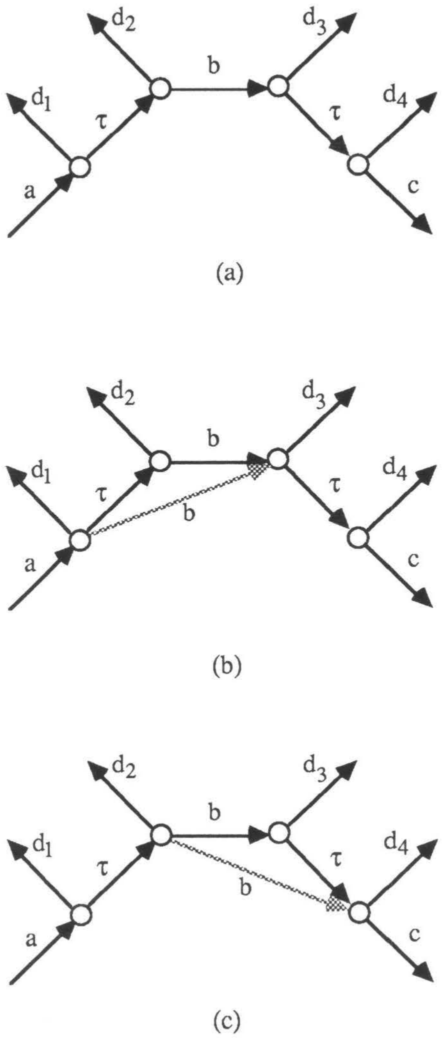
DEFINITION 1.3 Two graphs g and h are *τ-bisimilar* - notation: $g \leftrightarrow_\tau h$ - if there exists a symmetric relation R (called a *τ-bisimulation*) between the nodes of g and h such that:

i. The roots are related by R

ii. If R(r,s) and $r \rightarrow^a r'$, then either $a=\tau$ and R(r',s), or there exists a path $s \Rightarrow s_1 \rightarrow^a s_2 \Rightarrow s'$ such that R(r',s').

Again, $\leftrightarrow_\tau$ is an equivalence on $\boldsymbol{G}$ which is called *τ-bisimulation equivalence*, also known as *observation equivalence*.

To some extend, the notion of τ-bisimulation cannot be regarded as the natural generalization of ordinary bisimulation to an abstract setting with hidden steps. The reason for this is the fact that an important feature of a bisimulation is missing for τ-bisimulation, which is the property that any computation in the one process corresponds to a computation in the other, in such a way that all intermediate states of these computations correspond as well, due to the bisimulation relation. When HENNESSY & MILNER [72] introduced the first version of observation equivalence, they also insisted on relating the intermediate states of computations, as they tell us: "... any satisfactory comparison of the behaviour of concurrent programs must take into account their intermediate states as they progress through a computation, because differing intermediate states can be exploited in different program contexts to produce different overall behaviour ..." and: "If we consider a computation as a sequence of experiments (or communications), then the above remarks show that the intermediate states are compared. In fact, if p is to be equivalent to q, there must be a strong relationship between their respective intermediate states. At each intermediate stage in the computations, the respective "potentials" must also be the same". However, in Milner's observation equivalence, when satisfying the second requirement of Definition 1.3 one may execute arbitrarily many τ-steps in a graph without worrying about the status of the nodes that are passed through in the meantime.



(a)

(b)

(c)

FIGURE 1. Observation equivalence.

As an illustration, in Figure 1 we consider a path $a\cdot\tau\cdot b\cdot\tau\cdot c$ with outgoing edges $d_1,\ldots,d_4$, and it follows easily that all three graphs are observation equivalent. Note that one may add extra b-edges as in (b) and (c) without disturbing equivalence. However, in both (b) and (c) a new computation path is introduced - in which the outgoing edge d_2 (or d_3 respectively) is missing - and such a path did not occur in (a). Or - to put it differently - in the path introduced in (b) the options d_1 and d_2 are discarded simultaneously, whereas in (a) it corresponds to a path containing a state where the option d_1 is already discarded but d_2 is still possible. Also in the path introduced in (c) the choice not to perform d_3 is already made with the execution of the b-step, whereas in (a) it corresponds to a path in which this choice is made only after the b-step. Thus we argue that observation equivalence does not preserve the branching structure of processes and hence lacks one of the main characteristics of bisimulation semantics.

Consider the following alternative definition of bisimulation in order to see how we can overcome this deficit.

DEFINITION 1.4 Two graphs g and h are *branching bisimilar* - notation: $g \underline{\leftrightarrow}_b h$ - if there exists a symmetric relation R (called a *branching bisimulation*) between the nodes of g and h such that:

i. The roots are related by R
ii. If R(r,s) and $r \rightarrow^a r'$, then either $a=\tau$ and R(r',s), or there exists a path $s \Rightarrow s_1 \rightarrow^a s_2 \Rightarrow s'$ such that $R(r,s_1)$, $R(r',s_2)$ and $R(r',s')$.

In a picture, the difference between branching and τ-bisimulation can be characterized as follows:

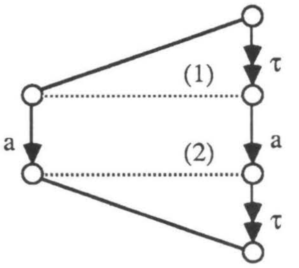


FIGURE 2. Bisimulations with τ.

The double arrow corresponds to the symbol $\Rightarrow$. Ordinary τ-bisimulation (Definition 1.3) says that every a-step $r \rightarrow^a r'$ corresponds with a path $s \Rightarrow s_1 \rightarrow^a s_2 \Rightarrow s'$ and so we obtain Figure 2 *without* the lines marked with (1) and (2). Branching bisimulation moreover requires relations between r and s_1 and between r' and' s_2 and thus we obtain Figure 2 *with* (1) and (2). Note that if $g \underline{\leftrightarrow}_b h$ then there exists a *largest* branching bisimulation between g and h, since the set of branching bisimulations is closed under arbitrary union. One can easily check that branching bisimilarity is an equivalence on ***G***, referred to as *branching bisimulation equivalence* or *branching equivalence* for short.

Obviously, branching equivalence more strongly preserves the branching structure of a graph since the starting and endnodes of the τ-paths $s \Rightarrow s_1$ and $s_2 \Rightarrow s$ are related to the same nodes. Observe that in Figure 1 there are no branching bisimulations between any of the graphs (a), (b) and (c). In particular, adding extra edges as in (b) and (c) no longer preserves branching equivalence. Equivalently, we could have strengthened Definition 1.3 (ii) by requiring *all* intermediate nodes in $s \Rightarrow s_1$ and $s_2 \Rightarrow s$ to be related with r and r' respectively. The fact that this alternative definition yields the same equivalence relation can be seen by use of the following lemma:

LEMMA 1.1 (stuttering lemma)
Let R *be the largest branching bisimulation between* g *and* h.
If $r \rightarrow^{\tau} r_1 \rightarrow^{\tau} \cdots \rightarrow^{\tau} r_m \rightarrow^{\tau} r'$ $(m \geq 0)$ *is a path such that* R(r,s) *and* R(r',s) *then* $\forall 1 \leq i \leq m$: $R(r_i,s)$.

PROOF First we prove Lemma 1.1 for a slightly different kind of bisimulation, defined as follows:

DEFINITION A *semi branching bisimulation* between two graphs g and h is a symmetric relation R between the nodes of g and h such that:
i. The roots are related by R
ii. If R(v,w) and $v \rightarrow^{a} v'$ then either
(a) $a=\tau$ and there exists a path $w \Rightarrow w'$ such that R(v,w') and R(v',w'), or:
(b) there exists a path $w \Rightarrow w_1 \rightarrow^{a} w_2 \Rightarrow w'$ such that $R(v,w_1)$, $R(v',w_2)$ and R(v',w').

The difference with branching bisimulation is in case (a), which can be illustrated by:

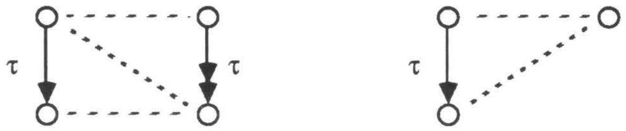


FIGURE 3. Semi branching (left) and branching bisimulation.

Now let (*) denote the property, mentioned in the lemma. Observe that (a) any branching bisimulation is a semi branching bisimulation and (b) any semi branching bisimulation satisfying (*) is a branching bisimulation.

CLAIM *The largest semi branching bisimulation between* g *and* h *satisfies* (*).

Let R be the largest semi branching bisimulation between g and h, let s be a node and let $r \rightarrow^{\tau} r_1 \rightarrow^{\tau} \cdots \rightarrow^{\tau} r_m \rightarrow^{\tau} r'$ $(m \geq 0)$ be a path such that R(r,s) and R(r',s). Then we prove that $R' = R \cup \{\{r_i,s\}: 1 \leq i \leq m\}$ is a semi branching bisimulation. We check the conditions:
(i) Clearly, the root nodes of g and h are related by R' (since by R).
(ii) Suppose R'(v,w) and $v \rightarrow^{a} v'$. If R(v,w) then it follows that either (a) $a=\tau$ and there exists a path $w \Rightarrow w'$ such that R(v,w') and R(v',w'), or (b) there exists a

path $w \Rightarrow w_1 \rightarrow^a w_2 \Rightarrow w'$ such that $R(v,w_1)$, $R(v',w_2)$ and $R(v',w')$. Hence, from $R \subseteq R'$ we find that R' satisfies the requirements in the definition above.

So assume *not* $R(v,w)$, then we find that either (1) $v=s$ and $w=r_i$ or (2) $v=r_i$ and $w=s$.

(1) If $s \rightarrow^a s'$ then it follows from $R(r',s)$ that

either: $a=\tau$ and there is a path $r' \Rightarrow r''$ such that $R(r'',s)$ and $R(r'',s')$. Hence there is a path $r_i \Rightarrow r' \Rightarrow r''$ such that $R'(r'',s)$ and $R'(r'',s')$ as required.

or: there is a path $r' \Rightarrow t_1 \rightarrow^a t_2 \Rightarrow r''$ such that $R(t_1,s)$, $R(t_2,s')$ and $R(r'',s')$ and hence

$r_i \Rightarrow r' \Rightarrow t_1 \rightarrow^a t_2 \Rightarrow r''$ with $R'(t_1,s)$, $R'(t_2,s')$ and $R'(r'',s')$.

(2) If $r_i \rightarrow^a r''$ then $r \rightarrow^\tau r_1 \rightarrow^\tau \cdots \rightarrow^\tau r_i \rightarrow^a r''$ and since $R(r,s)$ we find that there exists a sequence $s \Rightarrow s_1 \Rightarrow \cdots \Rightarrow s_i$ such that $R(r_1,s_1),\ldots,R(r_i,s_i)$. It follows from $R(r_i,s_i)$ that

either: $a=\tau$ and there exists a path $s_i \Rightarrow s'$ such that $R(r_i,s')$ and $R(r'',s')$. Hence $s \Rightarrow s'$ with $R'(r_i,s')$ and $R'(r'',s')$ as required.

or: there exists a path $s_i \Rightarrow t_1 \rightarrow^a t_2 \Rightarrow s''$ such that $R(r_i,t_1)$, $R(r'',t_2)$ and $R(r'',s'')$, and hence $s \Rightarrow s_i \Rightarrow t_1 \rightarrow^a t_2 \Rightarrow s''$ with $R'(r_i,t_1)$, $R'(r'',t_2)$ and $R'(r'',s'')$.

This proves that R' is a semi branching bisimulation. Since R is the largest we find $R=R'$.

So we proved the claim. Finally, conclude that the largest semi branching bisimulation is equal to the largest branching bisimulation, and thus we proved the lemma. □

The stuttering lemma will play a crucial role in some of the results we will present later.

It follows from Figure 2 that we can find two more kinds of bisimulation with τ, since we can leave out (1) while still having (2) and vice versa. Consider the following two definitions:

DEFINITION 1.5 Two graphs g and h are *η-bisimilar* - notation $g \leftrightarrow_\eta h$ - if there exists a symmetric relation R (called an *η-bisimulation*) between the nodes of g and h such that:

i. The roots are related by R
ii. If $R(r,s)$ and $r \rightarrow^a r'$, then either $a=\tau$ and $R(r',s)$, or there exists a path $s \Rightarrow s_1 \rightarrow^a s_2 \Rightarrow s'$ such that $R(r,s_1)$ and $R(r',s')$.

DEFINITION 1.6 Two graphs g and h are *delay bisimilar* - notation $g \leftrightarrow_d h$ - if there exists a symmetric relation R (called a *delay bisimulation*) between the nodes of g and h such that:

i. The roots are related by R
ii. If $R(r,s)$ and $r \rightarrow^a r'$, then either $a=\tau$ and $R(r',s)$, or there exists a path $s \Rightarrow s_1 \rightarrow^a s_2 \Rightarrow s'$ such that $R(r',s_2)$ and $R(r',s')$.

Notice the subtle differences between both definitions (and Definition 1.4). In Definition 1.5 the notion of η-bisimulation corresponds to Figure 2 without the relation (2) but with (1). Similarly, with delay bisimulation we have (2) but not (1). It is easy to see that in the definition of both branching and delay bisimulation the existence requirement of a node s' such that $s_2 \Rightarrow s'$ and R(r',s') is redundant.
From the definitions we find immediately that $g \underline{\leftrightarrow}_b h \Rightarrow g \underline{\leftrightarrow}_\eta h \Rightarrow g \underline{\leftrightarrow}_\tau h$ and similarly
$g \underline{\leftrightarrow}_b h \Rightarrow g \underline{\leftrightarrow}_d h \Rightarrow g \underline{\leftrightarrow}_\tau h$. Observe that in Figure 1 we find an η-bisimulation between (a) and (c) and a delay bisimulation between (a) and (b). Conversely, there is no η-bisimulation between (a) and (b) and no delay bisimulation between (a) and (c), so all implications are strict.
The notion of η-bisimulation was first introduced by BAETEN & VAN GLABBEEK [11] as a finer version of observation equivalence. A variant of delay bisimulation - only differing in the treatment of divergence - first appeared in MILNER [93], also under the name observational equivalence.

HISTORICAL NOTE:
The first semantic equivalence preserving the branching structure of processes was defined in HENNESSY & MILNER [72] and MILNER [92]. In [92] it was called *strong equivalence* or *strong congruence*. It was defined in terms of a decreasing sequence $\sim_0, \sim_1, ..., \sim_k, ...$ of equivalence relations. Originally, these relations where defined on CCS expressions that figured as states in transition systems, but one can also define them on nodes of (possibly different) process graphs.

DEFINITION 1.7 Let r and s be nodes of process graphs. Then:
$r \sim_0 s$ is always true
$r \sim_{k+1} s$ iff for all $a \in Act$
(i) if $r \rightarrow^a r'$ then there exists a node s' such that $s \rightarrow^a s'$ and $r' \sim_k s'$
(ii) if $s \rightarrow^a s'$ then there exists a node r' such that $r \rightarrow^a r'$ and $r' \sim_k s'$
$r \sim s$ iff for all $k \in \mathbf{N}$: $r \sim_k s$.
Two graphs g and h are *strongly equivalent*, notation $g \sim h$, if $root(g) \sim root(h)$.

A process graph is *finitely branching* if each node has only finitely many outgoing edges. In [72] and [92] strong congruence was defined only on CCS expressions corresponding with finitely branching graphs. On this domain, as was shown in [92], strong congruence 'satisfies its definition' in the following sense:

PROPOSITION 1.2 *Let* r *and* s *be nodes of finitely branching process graphs.*
Then $r \sim s$ *iff for all* $a \in Act$:
i. *if* $r \rightarrow^a r'$ *then there exists a node* s' *such that* $s \rightarrow^a s'$ *and* $r' \sim s'$
ii. *if* $s \rightarrow^a s'$ *then there exists a node* r' *such that* $r \rightarrow^a r'$ *and* $r' \sim s'$.

Strong equivalence is closely related to the notion of *bisimulation*, introduced by PARK [103] (cf. Definition 1.2). It is easy to verify that any bisimulation is included in each of the relations $\sim_k$ for $k \in \mathbf{N}$. Hence bisimulation equivalence is at least as discriminating as strong equivalence. On the other hand, from the former proposition it follows that with respect to finitely branching process graphs strong equivalence is a bisimulation, and hence the two notions coincide. With respect to infinitely branching graphs, $\sim$ is strictly coarser than bisimulation equivalence as can be seen from the following example. Consider the graphs

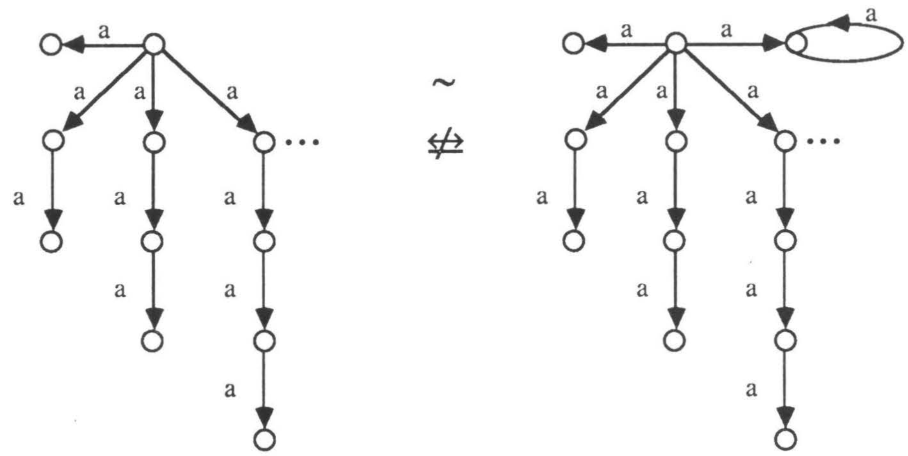


FIGURE 4. 'Strongly equivalent' graphs that are not bisimilar.

One can easily verify that these graphs are strongly equivalent in the sense of Definition 1.7, but not bisimilar.

PROPOSITION 1.3

i. With respect to finitely branching process graphs the notions $\sim$ and $\underline{\leftrightarrow}$ coincide;

ii. With respect to infinitely branching process graphs $\underline{\leftrightarrow}$ is strictly contained in $\sim$.

Starting from this observation, there are two different ways in which the notion of strong equivalence (in HENNESSY & MILNER [72] and MILNER [92] defined on finitely branching processes only) can be extended to infinitely branching process graphs. In MILNER [94] strong equivalence is chosen to be the relation of Definition 1.2, so strong equivalence and bisimulation equivalence are synonyms.

In the presence of a special action τ, representing an unobservable move of a process, one looks for a semantic equivalence that abstracts from internal moves in a process and for the rest resembles bisimulation equivalence. In particular, such an abstract equivalence has to satisfy some requirements such as:

- it is coarser than bisimulation equivalence

- it is equal to bisimulation equivalence with respect to processes not containing τ-edges
- it does not discriminate between the graphs

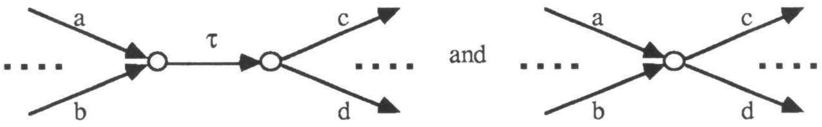


FIGURE 5. Contraction of internal moves.

The definition of strong congruence (~) was the starting point of HENNESSY & MILNER [72] when they introduced abstraction in CCS. Having in mind that τ-steps are not observable, they proposed that two process graph g and h are equivalent if every visible step in the one graph corresponds with a similar step in the other, apart from some arbitrarily long sequences of τ-steps that are allowed to precede or follow. This way they obtained a notion of *observational equivalence.* Originally, this relation was defined in the style of Definition 1.7, but in order to facilitate comparison with the other equivalences, we will present it in bisimulation style.

DEFINITION 1.8 Two graphs g and h are *observational equivalent* in the sense of HENNESSY & MILNER if there exists a symmetric relation R between the nodes of g and h such that:

i. The roots are related by R
ii. If R(r,s) and $r \rightarrow^{a} r'$ $(a \neq \tau)$, then there exists a path $s \Rightarrow s_1 \rightarrow^{a} s_2 \Rightarrow s'$ such that R(r',s').

Unfortunately, this type of observational equivalence turned out not to be a congruence for the CCS parallel composition operator, the free merge, or any other operator representing concurrent activity (cf. HENNESSY & MILNER [72]). Furthermore, we argue that it is not resistent against *refusal testing* as developed in PHILIPS [105] (Refusal testing is essentially the testing notion of MILNER [92], but without replication facility).

EXAMPLE Consider the two process graphs on the next page:

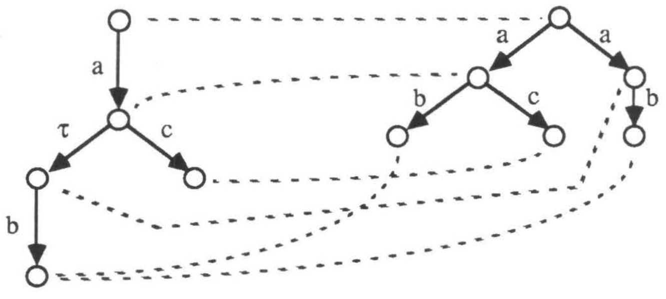


FIGURE 6. Observational equivalence does not respect refusal testing.

> These graphs are observational equivalent in the sense of HENNESSY & MILNER [72]; the relation R has been indicated in the figure above. Now expose them to the experiments a, d and c (in this order). The process on the right may respond as follows: a is accepted, d is refused and c is accepted (another possible respons would be: a accepted, d refused and c refused). However, this respons would not be possible in the process on the left: the attempt to execute the action d would cause the τ-edge to be executed, and then c cannot happen anymore.

Hence MILNER's version of observation equivalence [92] (which we call τ-bisimulation equivalence) can be regarded as an improvement. Both notions satisfy the requirements mentioned above, but additionally τ-bisimulation equivalence is a congruence for the CCS parallel composition operator and is resistent against refusal testing. Since observational equivalence in the sense of HENNESSY & MILNER [72] is coarser than τ-bisimulation equivalence, the criticism that τ-bisimulation equivalence does not preserve the branching structure of processes also applies to the variant of HENNESSY & MILNER [72].

2. AXIOMS

In this section we will turn several parts of our graph domain $\mathbb{G}$ into algebras, by defining some operations on them. This will enable us to give equational characterizations of the equivalences studied in the previous section. In the first subsection we use the operators of the axiom system BPA_τ (cf. BERGSTRA & KLOP [20]): action constants, alternative and sequential composition. In the second subsection we take the operators inaction, prefixing and alternative composition of CCS (cf. MILNER [92]). Finally, in the third subsection we combine the features of the previous two approaches, thereby obtaining the kernel of the extended algebra ACP_τ (cf. BERGSTRA & KLOP [20]). We will not consider parallel composition, restriction (or encapsulation), hiding and relabeling. However, we claim that these can be added without problem.

2.1. BASIC PROCESS ALGEBRA

For sake of convenience, in this subsection we will only consider *root unwound* process graphs, i.e. process graphs with no incoming edges at the root. Since each bisimulation equivalence class of process graphs contains a root unwound graph, this does not cause any loss of generality. Furthermore, we restrict ourselves to *non-trivial graphs* - having at least one edge - and we assume our graphs to be *divergence free*, meaning that they do not contain infinite τ-paths. The latter restriction will be cancelled later, but for the time being it suits us since having it we can stay closer to CCS in our presentation. (NOTE: apart from arguments about presentation, one may argue that there is still discussion about the role of divergence in bisimulation equivalence on processes, such as the dichotomy between explicit divergence MILNER [93], WALKER [126] and fair abstraction MILNER [92], BAETEN, BERGSTRA & KLOP [9], see also Section 5.3). The domain of root unwound, non-trivial and divergence free process graphs will be denoted by $\boldsymbol{G}_{\mathrm{BPA}}$. Clearly $\boldsymbol{G}_{\mathrm{BPA}} \subseteq \boldsymbol{G}$.

In order to equip $\boldsymbol{G}_{\mathrm{BPA}}$ with some structure, we introduce two binary infix written operators + and · and constants for every element in Act.

DEFINITION 2.1 The constants a∈ Act and the operators + and · are defined on $\boldsymbol{G}_{\mathrm{BPA}}$ as follows:

(i) Constants a∈ Act are interpreted by one-edge graphs labelled with a

(ii) (g + h) can be constructed by identifying the root nodes of g and h

(iii) (g·h) is constructed by identifying all endnodes (leaves) in g with the root of h. If g is without endnodes, then the result is just g.

As in regular algebra we will often leave out brackets and ·, assuming that · will always bind stronger than +.

The operators + and · are well-defined, even after deviding out *bisimulation equivalence* on $\boldsymbol{G}_{\mathrm{BPA}}$, as follows from the following proposition, the proof of which is straightforward and omitted.

PROPOSITION 2.1 *Bisimulation equivalence is a congruence with respect to the operators + and ·.*

Hence the structure ($\boldsymbol{G}_{\mathrm{BPA}}/\leftrightarrow$,+,·,Act) is a well-defined algebra. Considering its first order equational theory we find the axiom system BPA (cf. BERGSTRA & KLOP [19]) which stands for *Basic Process Algebra.*

x + y = y + x	A1
(x + y) + z = x + (y + z)	A2
x + x = x	A3
(x + y)z = xz + yz	A4
(xy)z = x(yz)	A5

Table 1. BPA.

As usual, we assume the axioms from Table 1 to be universally quantified.
Now let us say that a theory Γ is a *complete axiomatization* of a model $\boldsymbol{M}$ if for every pair of closed terms p and q we have: $\Gamma \vdash$ p=q if and only if $\boldsymbol{M} \models$ p=q. This definition deviates from the standard one, since usually also open terms are considered. Then the following theorem is due to BERGSTRA & KLOP [20]:

THEOREM 2.2 BPA *is a complete axiomatization of* ($G_{\mathrm{BPA}}/\underline{\leftrightarrow}$,+,·,Act).

Observe that in the presence of the trivial graph, BPA is not sound with respect to ($G_{\mathrm{BPA}}/\underline{\leftrightarrow}$,+,·,Act): axiom A4 no longer holds, with the trivial graph substituted for the variable y. For this reason it was excluded from G_{BPA} from the beginning.
In the same way one may wish to find axiomatizations for algebras resulting from deviding out the other equivalences of Section 1. However, as it turns out these equivalences are no congruences with respect to the operator +. In the case of observation equivalence this problem was solved by MILNER [92] by simply taking the closure of $\underline{\leftrightarrow}_\tau$ with respect to all contexts in CCS, thereby obtaining *observation congruence*. Similarly in HENNESSY & MILNER [72] *observational congruence* was defined as the CCS-closure of their variant of obsevational equivalence (Definition 1.8) and this congruence coincides with the one of MILNER [92]. BERGSTRA & KLOP [20] formulated an additional condition, yielding an immediate definition of observation congruence by means of bisimulation relations.

DEFINITION 2.2 (root condition) A relation R between nodes of process graphs is called *rooted* if root nodes are related with root nodes only.

Observe that every bisimulation (see Definition 1.2) is rooted, but this is not necessarily the case for the relations defined in Definitions 1.3-1.6. For any two process graphs g and h and $* \in \{\tau,b,\eta,d\}$ we write R: g $\underline{\leftrightarrow}_{r*}$ h if R is a rooted $*$-bisimulation between g and h, and g $\underline{\leftrightarrow}_{r*}$ h if such a relation exists.

THEOREM 2.3 *For* $* \in \{\tau,b,\eta,d\}$, $\underline{\leftrightarrow}_{r*}$ *is a congruence on* G_{BPA} *with respect to* + *and* ·.

PROOF We prove Theorem 2.3 for $\underline{\leftrightarrow}_{rb}$. The other proofs proceed in the same way.
$\underline{\leftrightarrow}_{rb}$ is reflexive since the identity relation is a rooted branching bisimulation between any graph and itself, and it is symmetric by definition. Furthermore, assume that R: g $\underline{\leftrightarrow}_{rb}$ g' and S: g' $\underline{\leftrightarrow}_{rb}$ g" and define: T(r,r") :$\Leftrightarrow$ for some r' in g': R(r,r') and S(r',r"). Now one can easily prove that T: g $\underline{\leftrightarrow}_{rb}$ g", and so $\underline{\leftrightarrow}_{rb}$ is transitive. Thus we proved that $\underline{\leftrightarrow}_{rb}$ is an equivalence. So it is left to prove that $\underline{\leftrightarrow}_{rb}$ respects the operators. Suppose that R: g $\underline{\leftrightarrow}_{rb}$ g' and S: h $\underline{\leftrightarrow}_{rb}$ h'.
$\underline{+}$: We prove that (R$\cup$S): (g + h) $\underline{\leftrightarrow}_{rb}$ (g' + h').
(i) Obviously the roots of (g + h) and (g' + h') are related.
(ii) Assume that in (g + h) we have an edge r $\rightarrow^a$ r' and suppose we have (R$\cup$S)(r,s) then from the construction of (g + h) it follows that this edge either

originates from g or from h; let us say it is g. It follows from $(R\cup S)(r,s)$ that we have either $R(r,s)$ or $S(r,s)$, so we have two distinct cases:
Firstly, suppose that $R(r,s)$. Then either $a=\tau$ and $R(r',s)$ - hence $(R\cup S)(r',s)$ and $(R\cup S)$ satisfies Definition 1.4 - or there exists a path $s \Rightarrow s_1 \rightarrow^a s_2 \Rightarrow s'$ in g' such that $R(r,s_1)$, $R(r',s_2)$ and $R(r',s')$. Obviously, we can find the same path in (g' + h') and we have that $(R\cup S)(r,s_1)$, $(R\cup S)(r',s_2)$ and $(R\cup S)(r',s')$ as required.
Secondly, suppose that we do *not* have $R(r,s)$. Then we have $S(r,s)$, and since we assumed that the edge $r \rightarrow^a r'$ came from g, we find that r has to be the (joint) root node of g and h. However, in S root nodes are related with root nodes only (the root condition), and so s must be the joint root node of g' and h'. Hence $R(r,s)$, which is a contradiction.
(iii) Obviously, the root nodes of (g + h) and (g' + h') are uniquely related by $(R\cup S)$.
$\underline{\cdot}$: we prove $R\cup S$: $(g\cdot h) \leftrightarrow_{rb} (g'\cdot h')$.
(i) Clearly, the roots of both graphs are related by R, hence by $R\cup S$.
(ii) Assume that in $(g\cdot h)$ we have an edge $r \rightarrow^a r'$ and suppose we have $(R\cup S)(r,s)$ then from the construction of $(g\cdot h)$ it follows that this edge either originates from g or from h.
(1) Firstly, let us say it is from g. From $(R\cup S)(r,s)$ we find that either $R(r,s)$ or $S(r,s)$. Since r cannot be an endnode in g we have $R(r,s)$. It follows from the fact that R is a rooted branching bisimulation that either $a=\tau$ and $R(r',s)$ - hence $(R\cup S)(r',s)$ as required - or there is a path $s \Rightarrow s_1 \rightarrow^a s_2 \Rightarrow s'$ in g' such that $R(r,s_1)$, $R(r',s_2)$ and $R(r',s')$ and thus we find the same path in $(g'\cdot h')$ such that $(R\cup S)(r,s_1)$, $(R\cup S)(r',s_2)$ and $(R\cup S)(r',s')$, as is required.
(2) Secondly, assume $r \rightarrow^a r'$ is from h.
- In case $R(r,s)$, we find that r is an endnode in g (since those are the only nodes of g that are identified with nodes from h). Suppose s is an endnode in g', then it is identified with the root node of h', and since S is a rooted branching bisimulation we find:
either $a=\tau$ and $S(r',s)$, hence $(R\cup S)(r',s)$;
or there exists a path $s \Rightarrow s_1 \rightarrow s_2 \Rightarrow s'$ such that $S(r,s_1)$, $S(r',s_2)$ and $S(r',s')$ and hence $(R\cup S)(r,s_1)$, $(R\cup S)(r',s_2)$ and $(R\cup S)(r',s')$ as required.
So let us assume that s is *not* an endnode in g', then it has at least one outgoing edge $s \rightarrow^b s_1$. Since R is a rooted branching bisimulation and $R(r,s)$, we find that $b=\tau$ and $R(r,s_1)$. The same argument holds for s_1 and thus we find a path $s=s_0 \rightarrow^\tau s_1 \rightarrow^\tau s_2 \rightarrow^\tau\ldots$ such that $R(r,s_i)$. Since all graphs in $\mathcal{G}_{BPA}$ are divergence free we have that all nodes s_i are distinct and furthermore the sequence $s=s_0 \rightarrow^\tau s_1 \rightarrow^\tau s_2 \rightarrow^\tau\ldots$ has bounded length. Hence there exists a path $s \Rightarrow s'$ to an endnode s' in g', such that $R(r,s')$ (and hence $(R\cup S)(r,s')$). Note that s' is identified with the root node of h'. Combining this result with the former part, we find that the conditions of Definition 1.4 are satisfied as required.
- In case *not* $R(r,s)$, then $S(r,s)$ and both r and s are from h and h' respectively. Now the requirement follows immediately from the fact that S is a branching bisimulation.

(iii) Clearly, the root nodes are uniquely related by $(R \cup S)$. □

THEOREM 2.4 *Provided that there exists al least one action* $a \in \text{Act}$ *with* $a \neq \tau$, $\underline{\leftrightarrow}_{r*}$ *is the coarsest congruence on* $\boldsymbol{G}_{\text{BPA}}$ *with respect to* + *that is contained in* $\underline{\leftrightarrow}_{*}$, *for* $* \in \{\tau, b, \eta, d\}$. *Hence* $\underline{\leftrightarrow}_{r\tau}$ *coincides with observation congruence.*

PROOF The idea for this proof is due to J.W. Klop (personal communication). Let g and $h \in \boldsymbol{G}_{\text{BPA}}$ and suppose $g+k \;\underline{\leftrightarrow}_{*}\; h+k$ for any graph $k \in \boldsymbol{G}_{\text{BPA}}$. Suppose there is an action $a \in \text{Act}$ $(a \neq \tau)$ that does not occur in g and h. Then $g+a \;\underline{\leftrightarrow}_{*}\; h+a$. Let R be a $*$-bisimulation between g+a and h+a, then R must be rooted. Therefore the restriction of R to the nodes of g and h is a rooted $*$-bisimulation between g and h. If no 'fresh atom' $a \in \text{Act}$ can be found a variant of this method still works. First note that for each infinite cardinal κ there are at least κ $*$-bisimulation equivalence classes of graphs with less then κ nodes. (Choose an action $a \in \text{Act}$ $(a \neq \tau)$ and define for each ordinal $\lambda > 0$ the graphs g_λ as follows: $g_1 = a$, $g_{\lambda+1} = g_\lambda + ag_\lambda$ and for λ a limit ordinal g_λ is contructed from all graphs g_μ for $\mu < \lambda$ by identifying their roots. Then with transfinite induction it follows that no two different g_λ's are $*$-bisimilar. Furthermore, for infinite λ, the cardinality of the nodes of g_λ is the cardinality of λ.) Thus for any two graphs g and h there must be a graph $k \in \boldsymbol{G}_{\text{BPA}}$ with the same cardinality such that k is not bisimilar with any subgraph corresponding with a node in g or h. Now take a $*$-bisimulation between $g+\tau k$ and $h+\tau k$. □

The equivalence relations $\underline{\leftrightarrow}_{r*}$ are called *rooted $*$-bisimulation equivalence* or *$*$-bisimulation congruence*. As a consequence of Theorem 2.3, we find that all structures $(\boldsymbol{G}_{\text{BPA}}/\underline{\leftrightarrow}_{*}, +, \cdot, \text{Act})$ are well-defined algebras, every one of which may satisfy a different equational theory. In a slightly different setting, MILNER [92] found that the algebra $(\boldsymbol{G}_{\text{BPA}}/\underline{\leftrightarrow}_{r\tau}, +, \cdot, \text{Act})$ can be completely axiomatized by BPA together with the following three equations:

$x\tau = x$	T1
$\tau x = \tau x + x$	T2
$a(\tau x + y) = a(\tau x + y) + ax$	T3

Table 2. τ-laws $(a \in \text{Act})$.

THEOREM 2.5 BPA + T1-T3 *is a complete axiomatization of* $(\boldsymbol{G}_{\text{BPA}}/\underline{\leftrightarrow}_{r\tau}, +, \cdot, \text{Act})$.

In the setting of BPA and process graphs, this theorem was first established in BERGSTRA & KLOP [20]. Its proof will be given in Section 4, together with the proofs of the Theorems 2.6-2.8.

From Figure 1 one can observe that the constructions (b) and (c) are highly fundamental for the behaviour of τ in the graph model. For instance, by simplifying Figure 1 (b) one finds the second τ-law T2, whereas T3 can be easily found from Figure 1 (c). This shows us that the extra τ-laws T2 and T3 actually originate from the fact that observation equivalence does not preserve branching structures.

Since branching bisimulation equivalence distinguishes between all three graphs in Figure 1, we expect that the laws T2 and T3 will no longer hold in $(G_{BPA}/\underline{\leftrightarrow}_{rb},+,\cdot,Act)$. As it turns out, axiom T3 is completely dropped and T2 is considerably weakened to axiom H2 from the following table:

$x\tau = x$	H1 (T1)
$x(\tau(y + z) + y) = x(y + z)$	H2

Table 3. τ-laws for branching bisimulation.

H1 is the same axiom as T1 whereas H2 is derivable from T1 and T2 as one can check easily. Both axioms refer to the axiomatization of η, a constant for abstraction from BAETEN & VAN GLABBEEK [11] similar to τ. In fact they are a variation on the first two η-laws in the sense that in [11] the second law H2 was only introduced for atomic actions x, instead of taking x as a general variable ranging over all processes. On the domain of closed terms the two variants are equally powerful.

THEOREM 2.6 BPA + H1-H2 *is a complete axiomatization of* $(G_{BPA}/\underline{\leftrightarrow}_{rb},+,\cdot,Act)$.

Obviously, $\underline{\leftrightarrow}_{r\eta}$ is a coarser notion than $\underline{\leftrightarrow}_{rb}$ and it respects the axioms H1-H2. As it turns out we have the additional axiom H3 which was introduced earlier as T3.

$x\tau = x$	H1 (T1)
$x(\tau(y + z) + y) = x(y + z)$	H2
$a(\tau x + y) = a(\tau x + y) + ax$	H3 (T3)

Table 4. η-laws ($a \in Act$).

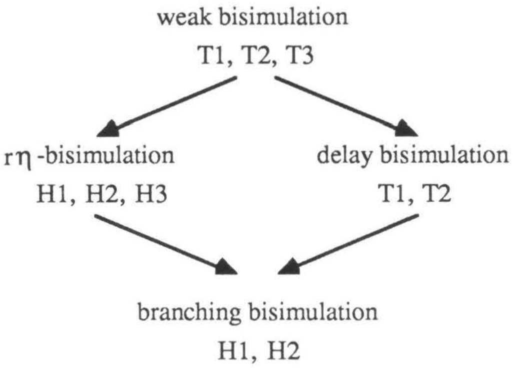
BAETEN & VAN GLABBEEK [11] established a completeness theorem for rooted η-bisimulation:

THEOREM 2.7 BPA + H1-H3 *is a complete axiomatization of* $(G_{BPA}/\underline{\leftrightarrow}_{r\eta},+,\cdot,Act)$.

So, on closed terms, the difference between H2 and T2 is precisely all the difference there is between the usual τ-laws and η. Finally a completeness theorem for delay bisimulation was (in the setting of CCS) established by WALKER [126].

THEOREM 2.8 BPA + T1-T2 *is a complete axiomatization of* $(G_{BPA}/\leftrightarrow_{r\Delta},+,\cdot,Act)$.

Resuming we have the following diagram (see Figure 7):



T1	$x\tau = x$	H1
	$x(\tau(y + z) + y) = x(y + z)$	H2
T2	$\tau x = \tau x + x$	
T3	$a(\tau x + y) = a(\tau x + y) + ax$	H3

FIGURE 7. Four notions of bisimulation with τ ($a \in Act$).

REMARK

In case we do not restrict to root unwound process graphs the definitions of the various bisimulations become a little more complicated. In particular the root conditions will have a different form (cf. BAETEN & VAN GLABBEEK [11]) and the definition of the operator + on process graphs has to be changed.

2.2 CCS

In the setting of CCS we extend the graph domain G_{BPA} to G_{CCS} consisting of the root unwound process graphs, thus no longer excluding the trivial graph (the one-node graph without edges) nor any of the graphs with divergences (i.e. infinite τ-paths). We obtain: $G_{BPA} \subseteq G_{CCS} \subseteq G$.

We introduce a constant 0 for inaction, a binary infix written operator + for alternative composition, and unary operators a. for prefixing ($a \in Act$).

DEFINITION 2.3 The constant 0 and the operators + and a. are defined on G_{CCS} as follows:

(i) The constant 0 is interpreted as the trivial graph
(ii) (g + h) can be constructed by identifying the root nodes of g and h
(iii) (a.g) is constructed from g by adding a new node which will be the root of a.g, and a new a-labelled edge from the root of a.g to the root of g.

We will often leave out brackets, assuming that + will be the weakest operator symbol. For agents p we will often write ap instead of a.p in order to avoid non-essential distinctions between CCS and ACP. Similarly, we write Act for the set of prefix operators {a.: a∈Act}. MILNER [92] proved that the operators from Act and + all are well-defined on $G_{\mathrm{CCS}}/\underline{\leftrightarrow}$:

PROPOSITION 2.9 *Bisimulation equivalence is a congruence with respect to the operators from* Act *and* +.

So again, the structure ($G_{\mathrm{CCS}}/\underline{\leftrightarrow}$,0,+,Act) is a well-defined algebra, and as in the case of ($G_{\mathrm{BPA}}/\underline{\leftrightarrow}$,+,·,Act) we can find a complete axiomatization of its equalities with respect to closed terms:

$x + y = y + x$	A1
$(x + y) + z = x + (y + z)$	A2
$x + x = x$	A3
$x + 0 = x$	A6

Table 5. Basic CCS.

Let us call the theory from Table 5 *Basic CCS*, and write BCCS := A1-A3,A6. Then the following theorem is due to HENNESSY & MILNER [72] and MILNER [92].

THEOREM 2.10 *BCCS is a complete axiomatization of* ($G_{\mathrm{CCS}}/\underline{\leftrightarrow}$,0,+,Act).

As before, we have four other equivalences $\underline{\leftrightarrow}_{r*}$ for $* \in \{\tau,b,\eta,d\}$ on G_{CCS} which can be considered. First we establish that they are congruences.

THEOREM 2.11 *For* $* \in \{\tau,b,\eta,d\}$, $\underline{\leftrightarrow}_{r*}$ *is a congruence on* G_{CCS} *with respect to + and* Act.

PROOF We prove it for $\underline{\leftrightarrow}_{rb}$. The other proofs proceed in the same way.
The proof that $\underline{\leftrightarrow}_{rb}$ is an equivalence and respects + can be copied from the proof of Theorem 2.3. So it is left to prove that it respects the operators in Act. So suppose that R: g $\underline{\leftrightarrow}_{rb}$ g' and p, p' are the root nodes of a.g and a.g'. Put R* := R∪{p,p'}. Then we prove R*: (a.g) $\underline{\leftrightarrow}_{rb}$ (a.g').

(i) Clearly, the roots of both graphs are related by R*.

(ii) Assume that in (a.g) we have an edge $r \rightarrow^{b} r'$ and suppose we have R*(r,s) then from the construction of (a.g) it follows that either r=p or this edge originates from g.

If r=p then by the definition of R* we have s=p'. Furthermore, b=a and r' is the root node of g and by the construction of prefixing we find that in g' there exists an edge $s \rightarrow^{a} s'$ to the root node s' of g'. Since R is a branching bisimulation we find R(r',s') and hence R*(r',s').

If $r \rightarrow^{b} r'$ originates from g then it follows from the definition of R* that R(r,s), from which the requirement follows immediately.

(iii) Clearly, the root nodes are uniquely related by R*. □

It follows from Theorem 2.4 that $\underline{\leftrightarrow}_{r*}$ is moreover the coarsest BCCS-congruence contained in $\underline{\leftrightarrow}_{*}$.

Now consider the axioms from the following table:

H1'	$a\tau x = ax$	T1'
H2'	$a(\tau(y + z) + y) = a(y + z)$	
	$\tau x = \tau x + x$	T2'
H3'	$a(\tau x + y) = a(\tau x + y) + ax$	T3'

Table 6. τ-laws in CCS ($a \in Act$).

The only difference between these axioms and the ones introduced in the previous section is the replacement of sequential composition by prefixing in the axioms T1 (H1) and H2. The prime accents (') refer to this replacement. Note that H1' is derivable from H2. We find the following completeness results:

THEOREM 2.12

(i) BCCS *is a complete axiomatization of* $(G_{CCS}/\underline{\leftrightarrow},0,+,Act)$

(ii) BCCS + T1'-T3' *is a complete axiomatization of* $(G_{CCS}/\underline{\leftrightarrow}_{r\tau},0,+,Act)$.

(iii) BCCS + H2' *is a complete axiomatization of* $(G_{CCS}/\underline{\leftrightarrow}_{rb},0,+,Act)$.

(iv) BCCS + H2'-H3' *is a complete axiomatization of* $(G_{CCS}/\underline{\leftrightarrow}_{r\eta},0,+,Act)$.

(v) BCCS + T1'-T2' *is a complete axiomatization of* $(G_{CCS}/\underline{\leftrightarrow}_{rd},0,+,Act)$.

For the proof of Theorem 2.12, we refer to Section 4.

2.3. TERMINATION

In the previous two subsections, we presented two models: the model $(G_{BPA}/\underline{\leftrightarrow}_{r*},+,\cdot,Act)$ for BPA with sequential composition, and $(G_{CCS}/\underline{\leftrightarrow}_{r*},0,+,Act)$ for BCCS with prefixing. As we argued before, including the trivial graph in

$\boldsymbol{G}_{\mathrm{BPA}}/\underline{\leftrightarrow}_{r*}$ would destroy the soundness of BPA in the corresponding model, i.e. of the axiom A4. Furthermore, from $\boldsymbol{G}_{\mathrm{BPA}}/\underline{\leftrightarrow}_{r*}$ we have to exclude graphs containing infinite τ-paths since otherwise sequential composition no longer respects the equivalences - i.e. the equivalences $\underline{\leftrightarrow}_{r*}$ are no longer congruences with respect to $\cdot$. For consider the following example:

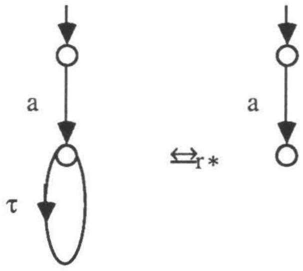


FIGURE 8. Equivalent graphs with and without divergence.

In Figure 8, we find two equivalent graphs, one with and one without divergence, which we informally denote by $a{\cdot}\tau^{\omega}$ and a. So: $a{\cdot}\tau^{\omega}$ $\underline{\leftrightarrow}_{r*}$ a, for $* \in \{\tau, b, \eta, d\}$. However, since $a{\cdot}\tau^{\omega}$ does not contain any endnodes we find that $(a{\cdot}\tau^{\omega}){\cdot}b = a{\cdot}\tau^{\omega}$ and $a{\cdot}\tau^{\omega}$ $\not\underline{\leftrightarrow}$ ab. So in the presence of divergence $\underline{\leftrightarrow}_{r*}$ no longer is a congruence with respect to $\cdot$.

The question arises whether the virtues of $(\boldsymbol{G}_{\mathrm{BPA}}/\underline{\leftrightarrow}_{r*},+,\cdot,\mathrm{Act})$ and $(\boldsymbol{G}_{\mathrm{CCS}}/\underline{\leftrightarrow}_{r*},0,+,\mathrm{Act})$ can be conbined, i.e. whether it is possible to define inaction and general sequential composition in one model (without destroying the intuitively plausable axiom A4) as well as to define general sequential composition on graphs with possible divergence paths, while respecting the equivalences. We will give a positive answer to this question by once again extending $\boldsymbol{G}_{\mathrm{CCS}}$ to a larger domain $\boldsymbol{G}_{\mathrm{ACP}}$ (so: $\boldsymbol{G}_{\mathrm{BPA}} \subseteq \boldsymbol{G}_{\mathrm{CCS}} \subseteq \boldsymbol{G}_{\mathrm{ACP}}$).

Let us extend the set Act with an additional label, written as $\surd$. Then, in $\boldsymbol{G}_{\mathrm{ACP}}$ we will distinguish between successful and unsuccessful termination of a process by adding a *termination edge* to the endnodes which are considered to terminate successfully. Such termination edges consist of an outgoing edge labelled with $\surd$ to a new endnode. Let $\boldsymbol{G}_{\mathrm{ACP}}$ consist of all graphs that can be obtained from non-trivial, root unwound graphs from $\boldsymbol{G}_{\mathrm{BPA}}$ by adding termination edges to some of their endnodes. Next we add the trivial graph to $\boldsymbol{G}_{\mathrm{ACP}}$ but assume that $\boldsymbol{G}_{\mathrm{ACP}}$ is without the graph consisting of a single termination edge, i.e. the graph representing instant termination.

Observe that in graphs from $\boldsymbol{G}_{\mathrm{ACP}}$ every node has at most one outgoing termination edge and if it has one, then it does not have any other outgoing edges. Furthermore, if a node has an incoming termination edge then it is an endnode and it does not have any other incoming edges. We immediately find that $\boldsymbol{G}_{\mathrm{CCS}} \subseteq \boldsymbol{G}_{\mathrm{ACP}}$ and $\boldsymbol{G}_{\mathrm{ACP}} \subseteq \boldsymbol{G}^{\surd}$, where $\boldsymbol{G}^{\surd}$ is the set of process graphs with $\surd$ as a possible edge-label. The difference between G_{CCS} and G_{ACP} is that the latter distinguishes between two kinds of termination.

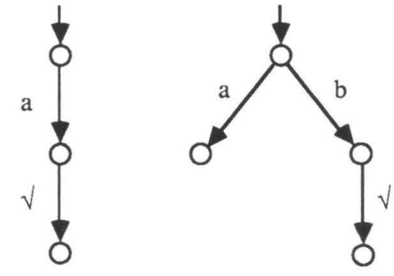


FIGURE 9. Process graphs with termination edges.

With respect to the algebraic operators, we simple combine the operators from BPA and ACP, but we adapt the definitions of action constants and sequential composition to the presence of √-labels. This is done in the following definition. The new operator for sequential composition will again be denoted by ·, and similarly for action constants. It will appear from the context (whether it is about G_{BPA} or G_{ACP}) which one of the Definitions 2.1 and 2.4 presents their current interpretation. In case of doubt we underline the BPA operators.

DEFINITION 2.4 On G_{ACP} the constants 0 and a (for a∈ Act) and the operators + and · are defined as follows:

(i) 0 is the trivial graph
(ii) Constants a∈ Act are interpreted by the left hand side of Figure 9
(iii) (g + h) can be constructed by identifying the root nodes of g and h
(iv) (g·h) is constructed by identifying every node in g with an outgoing termination edge with the root node of h while deleting its termination edge. The root node of (g·h) is that of g. If g is without termination edges, then (g·h) is just g.

The prefixing operator of CCS can now be defined by: a.g=a·g. In the subdomain G_{CCS} of G_{ACP} all processes end in deadlock (unsuccesful termination), so g·h=g. This explains the absence of sequential composition on G_{CCS}. Let G'_{BPA} be the subdomain of G_{ACP} consisting of all divergence free graphs from G_{ACP} only ending with succesful termination. Then $(G'_{\mathrm{BPA}},+,\cdot,\mathrm{Act})$ and $(G_{\mathrm{BPA}},+,\underline{\cdot},\underline{\mathrm{Act}})$ are isomorphic algebras and the latter can be interpreted as a notational abbreviation of the former, where all √-labels have been left out.

On the new graph domain G_{ACP} we can define the bisimulation relations from Definition 1.2-1.6 and 2.2, taking into account that √∈ Act. That is, termination edges are not treated anyhow different from other edges. The relations on G_{BPA}, inherited through the isomorphism from G'_{BPA}, coincide with the relations considered in Subsection 2.1. However, this is no longer true if divergent graphs would be added to G_{BPA}; in that case all relations need an additional clause:

- If R(r,s) and r is an endnode than there is a path s $\Rightarrow$ s' to an endnode s'.

In order to prevent this complication in Section 2.1, there we treated divergence free graphs only.

The fact that Definition 2.4 provides us with a proper algebraic structure on G_{ACP} follows from the following theorem:

THEOREM 2.13 *All equivalences* $\leftrightarrow$, $\leftrightarrow_{\mathrm{r}\tau}$, $\leftrightarrow_{\mathrm{rb}}$, $\leftrightarrow_{\mathrm{r}\eta}$ *and* $\leftrightarrow_{\mathrm{rd}}$ *are congruences with respect to the operators + and · on* G_{ACP}.

PROOF Again we prove the theorem for $\leftrightarrow_{\mathrm{rb}}$. The fact the on G_{ACP} they are conguences with respect to + can be found directly from the proof of Theorem 2.3. Considering the proof for ·, suppose that R: g $\leftrightarrow_{\mathrm{rb}}$ g' and S: h $\leftrightarrow_{\mathrm{rb}}$ h'. Let R' be the restriction of R to the nodes in g that also appear in g·h (i.e. the nodes without incoming $\surd$-edges). We prove that R'$\cup$S: (g·h) $\leftrightarrow_{\mathrm{rb}}$ (g'·h').

(i) Clearly the roots of (g·h) and (g'·h') are related by R'$\cup$S.

(ii) Assume that in (g·h) we have an edge r $\rightarrow^{a}$ r' and suppose (R'$\cup$S)(r,s), then from the construction of (g·h) it follows that this edge either originates from g or from h. If it is from g, then the proof proceeds as in the proof of Theorem 2.3. So assume r $\rightarrow^{a}$ r' is from h.

- In case R'(r,s), we find that in g the node r has an outgoing termination edge r $\rightarrow^{\surd}$ r" to an endnode r" (since those are the only nodes of g that are identified with nodes from h). Since R is a branching bisimulation, we find that in g' there exists a path s $\Rightarrow$ s' $\rightarrow^{\surd}$ s" such that R(r,s') and R(r",s"). By applying the definition of $\leftrightarrow_{\mathrm{rb}}$ we even find that all nodes in s $\Rightarrow$ s' are related with r. Furthermore, by construction of (g'·h') the node s' is identified with the root node of h', and since S is a rooted branching bisimulation between h and h', we find: either a=τ and S(r',s'), hence (R'$\cup$S)(r',s');
or there exists a path s' $\Rightarrow$ $s_1 \rightarrow s_2 \Rightarrow s_3$ such that S(r,s_1), S(r',s_2) and S(r',s_3) and hence (R'$\cup$S)(r,s_1), (R'$\cup$S)(r',s_2) and (R'$\cup$S)(r',s_3) as required.

- In case *not* R'(r,s), then S(r,s) and both r and s are from h and h' respectively. Now the requirement follows immediately from the fact that S is a branching bisimulation.

(iii) In G_{ACP} the root node cannot have an outgoing termination edge, and hence the root nodes of (g·h) and (g'·h') are only related by R' (they are not identified with nodes from h or h'). Hence (R'$\cup$S) is rooted since R is. □

As a consequence, we find a well-defined algebra ($G_{\mathrm{ACP}}/\leftrightarrow$,0,+,·,Act), and four others with domain $G_{\mathrm{ACP}}/\leftrightarrow_{\mathrm{r}*}$ ($* \in \{\tau,\mathrm{b},\eta,\mathrm{d}\}$). To start with, we find that the following basic theory is valid in all five algebras (see Table 7):

$x + y = y + x$	A1
$(x + y) + z = x + (y + z)$	A2
$x + x = x$	A3
$(x + y)z = xz + yz$	A4
$(xy)z = x(yz)$	A5
$x + 0 = x$	A6
$0 \cdot x = 0$	A7

Table 7. BPA_0.

The theory BPA_0 is the kernel of the axiom system ACP, introduced in BERGSTRA & KLOP [19], where 0 was called δ. As before, we have the following completeness theorem for the five respective algebras:

THEOREM 2.14
(i) BPA_0 *is a complete axiomatization of* $(G_{ACP}/\underline{\leftrightarrow},0,+,\cdot,Act)$
(ii) BPA_0 + T1-T3 *is a complete axiomatization of* $(G_{ACP}/\underline{\leftrightarrow}_{r\tau},0,+,\cdot,Act)$
(iii) BPA_0 + H1-H2 *is a complete axiomatization of* $(G_{ACP}/\underline{\leftrightarrow}_{rb},0,+,\cdot,Act)$
(iv) BPA_0 + H1-H3 *is a complete axiomatization of* $(G_{ACP}/\underline{\leftrightarrow}_{r\eta},0,+,\cdot,Act)$
(v) BPA_0 + T1-T2 *is a complete axiomatization of* $(G_{ACP}/\underline{\leftrightarrow}_{rd},0,+,\cdot,Act)$.

Again for the proof of this completeness theorem we refer to Section 4.

3. BRANCHES AND TRACES

As we saw in Figure 1, while preserving observation equivalence we are able to introduce new 'paths' in a graph. To be more precise: in these new paths alternative options may branch off at different places than in any of the old paths. So far, we claimed to have solved this problem by defining a new kind of bisimulation, but as of yet we still have to prove that our solution solves the problem in a fundamental way. In this section we will establish an alternative characterization of branching bisimulation. In fact, we will show the way in which branching bisimulation preserves the branching structure of graphs. Let us first consider ordinary bisimulation.

DEFINITION 3.1 A *concrete trace* of a process graph is a finite sequence $(a_1, a_2, a_3,..., a_k)$ of actions from Act, such that there exists a path $r_0 \rightarrow^{a_1} r_1 \rightarrow^{a_2} r_2 \rightarrow \cdots \rightarrow^{a_k} r_k$ from the root node r_0.

Two graphs g and h are said to be *concrete trace equivalent*, notation g $\equiv_t$ h, if their *concrete trace sets* (i.e. the sets of their concrete traces) are equal. It is easily checked that $\equiv_t$ is a congruence on G_{BPA} and g $\leftrightarrows$ h $\Rightarrow$ g $\equiv_t$ h. Consequently we find that $G_{BPA}/\equiv_t$ is a model for BPA. Compared with bisimulation, concrete trace equivalence makes much more identifications. For example, we find that $G_{BPA}/\equiv_t$ satisfies the equation x(y + z) = xy + xz which cannot be proved from BPA.
The main reason for this is that in a concrete trace we lose information about the potentials in the intermediate nodes. Therefore we cannot distinguish between processes a(b + c) and (ab + ac). In the following we will use *colours* at the nodes to indicate these potentials.

DEFINITION 3.2 A *coloured graph* is a process graph with colours C$\in$ C as labels at the nodes.

Obviously, in a coloured graph we have traces which have colours in the nodes:

DEFINITION 3.3 A *concrete coloured trace* of a coloured graph g is a sequence (C_0, a_1, C_1, a_2, C_2,..., a_k, C_k) for which there exists a path $r_0 \rightarrow^{a_1} r_1 \rightarrow^{a_2} r_2 \rightarrow \cdots \rightarrow^{a_k} r_k$ in g, starting from the root node r_0, such that r_i has colour C_i.

The concrete coloured traces of a node r in a graph g are the concrete coloured traces of the subgraph $(g)_r$ of g that has r as its root node. This graph is obtained from g by deleting all nodes and edges which are inaccessible from r.
The question remains how to detect the colour of a node in a graph, or - to put it differently - how to define the concept of 'potential in a node' properly. There are several ways to do this. Probably the shortest definition is the following:

DEFINITION 3.4 A *concrete consistent colouring* of a set of graphs is a colouring of their nodes with the property that two nodes have the same colour only if they have the same concrete coloured trace set.

Obviously, the *trivial* colouring - in which every node has a different colour - is consistent on any set of graphs. Note that - even apart from the choice of the colours - a set of graphs can have more than one consistent colouring. For instance, consider a set containing only an infinite graph representing a^ω or a·a·a··· then obviously the *homogeneous* colouring - in which every node has the same colour - is a consistent one, as well as the *alternating* or the trivial colouring.
Let us say two graphs g and h are *concrete coloured trace equivalent* - notation: g $\equiv_{cc}$ h - if for some concrete consistent colouring on {g,h} they have the same concrete coloured trace set, or equivalently the root nodes have the same colour. Then we have the following important characterization:

THEOREM 3.1 $g \leftrightarrow h$ *if and only if* $g \equiv_{cc} h$.

PROOF $\Rightarrow$: Suppose R is the largest bisimulation relation between g and h. Let $\underline{R}$ be the transitive closure of R, then $\underline{R}$ is an equivalence relation on the set of nodes from g and h. Let C be the set of equivalence classes induced by $\underline{R}$ and label every node with its own equivalence class. Then this colouring is consistent on g and h. To see this let r_0 be a node in g say, and $(C_0, a_1, C_1, a_2, C_2,..., a_k, C_k)$ be a concrete coloured trace which corresponds to a path $r_0 \rightarrow^{a_1} r_1 \rightarrow^{a_2} r_2 \rightarrow \cdots \rightarrow^{a_k} r_k$ starting from r_0. Now suppose for some node s_0 in h we have $R(r_0,s_0)$, then we find from Definition 1.2 that $s_0 \rightarrow^{a_1} s_1$ for some s_1 such that $R(r_1,s_1)$. Thus r_1 and s_1 have the same colour C_1. By induction we find that s_0 has the same concrete coloured trace $(C_0, a_1, C_1, a_2, C_2,..., a_k, C_k)$. So R preserves concrete coloured trace sets, hence so does $\underline{R}$. Since the roots of g and h are related we find $g \equiv_{cc} h$.
$\Leftarrow$: Suppose that g and h have the same concrete coloured trace sets. Then consider the relation R which relates two nodes of g and h iff they are labelled with the same colour. It is easy to prove that R is a bisimulation between g and h. □

So far we did not have any notion of abstraction in the definition of coloured traces, so if a coloured graph has τ-labels then these are treated as if they were ordinary actions. In the following definition we find how to abstract from these τ-steps. The idea is simple: τ-steps can only be left out if they are *inert*, which says that they are between two nodes that have the same colour (potentials). Thus it is not only that the inert steps are not observable, but even more, they do not cause any change in the overall state of the machine.

DEFINITION 3.5 A *coloured trace* of a coloured graph is a sequence of the form $(C_0, a_1, C_1, a_2, C_2,..., a_k, C_k)$ which is obtained from a concrete coloured trace of this graph by replacing all subsequences $(C, \tau, C, \tau, ..., \tau, C)$ by C.

DEFINITION 3.6 A *consistent colouring* of a set of graphs is a colouring of their nodes with the property that two nodes have the same colour only if they have the same coloured trace set. Furthermore such a colouring is *rooted* if no root-node has the same colour as a non-root node.

For two root unwound graphs g and h let us write $g \equiv_c h$ if for some consistent colouring on {g,h} they have the same coloured trace set, and $g \equiv_{rc} h$ if moreover this colouring is rooted. Then we find the following characterization for (rooted) branching bisimulation:

THEOREM 3.2

i. $g \leftrightarrow_b h$ *if and only if* $g \equiv_c h$

ii. $g \leftrightarrow_{rb} h$ *if and only if* $g \equiv_{rc} h$.

PROOF ⇒: Suppose R is the largest (rooted) branching bisimulation between g and h. Let $\underline{R}$ be its transitive closure and C the set of equivalence classes induced by $\underline{R}$. Then the colouring in which every node is labelled with its own equivalence class is consistent (and rooted) on g and h.

To see this, let us write C(r) for the colour of the node r and assume that, for certain nodes r_0 and s_0, $R(r_0,s_0)$ and r_0 has an coloured trace $(C_0, a_1, C_1, a_2, C_2,..., a_k, C_k)$. Then there exists a path of the form $r_0 \rightarrow^{\tau} u_1 \rightarrow^{\tau} ... \rightarrow^{\tau} u_m \rightarrow^{a_1} r_1$ $(m \geq 0)$ such that $C(r_1) = C_1$ and for all i: $C(u_i) = C(r_0) = C_0$. For every edge $u_i \rightarrow^{\tau} u_{i+1}$ $(0 \leq i < m, u_0=r_0)$ there exists a path $v_i \Rightarrow v_{i+1}$ $(v_0=s_0)$ such that $R(u_i,v_i)$, and all intermediate nodes are related to either u_i or u_{i+1} (by Lemma 1.1), hence all v_i have the same colour C_0. So we find a path $s_0 \Rightarrow v_m$ with only one colour in the nodes such that $R(u_m,v_m)$.

Next, since $u_m \rightarrow^{a_1} r_1$ and $R(u_m,v_m)$ we find that either $a_1=\tau$ and $R(r_1,v_m)$ - in which case $C_1=C_0$ in contradiction with $(C_0, a_1, C_1, a_2, C_2,..., a_k, C_k)$ being a coloured trace - or there is a path $v_m \Rightarrow t_1 \rightarrow^{a_1} s_1$ such that $R(u_m,t_1)$ and $R(r_1,s_1)$. Again by Lemma 1.1 we find that t_1 and all the intermediate nodes in ⇒ have the same colour as v_m and so we find a coloured trace (C_0, a_1, C_1) of s_0. By repeating this argument k times, we find that s_0 has a coloured trace $(C_0, a_1, C_1, a_2, C_2,..., a_k, C_k)$ and so R preserves coloured trace sets. Thus $\underline{R}$ induces a consistent colouring and since the roots are related we find $g \equiv_c h$. If moreover R is rooted, then so is the induced colouring.

⇐: Consider a (rooted) consistent colouring such that the coloured trace sets of g and h are equal with respect to that colouring. Let R be the relation between nodes of g and h relating two nodes iff they have the same colour, then it is easy to see that R is a (rooted) branching bisimulation. □

This characterization provides us with a clear intuition about what branching bisimulation actually is, since the difference between *inert* steps - not changing the state of the machine - and relevant τ-steps - that behave as common actions - is visualized immediately by the (change of) colours at the nodes. It follows that branching bisimulation equivalence preserves computations together with the potentials in all intermediate states that are passed through.

Another way of looking at the canonical colouring of a graph is the following. Since trace-equivalence is too weak to characterize branching bisimilarity we can add more information to traces in order to distinguish between processes. Consider the following definition:

DEFINITION 3.7 For ordinals α the *α-trace set* of a graph g is defined as follows:

1. The α-trace set of a node r of g is the set of all γ-traces of r, for $\gamma<\alpha$.
2. An α-trace of r is made of a sequence $(T_0, a_1, T_1, a_2,..., a_k, T_k)$, where a_i are actions from Act and T_i are α-trace sets such that g has a path of the form

$r_0 \rightarrow^{a_1} r_1 \rightarrow^{a_2} \cdots \rightarrow^{a_k} r_k$ for which r_i has α-trace set T_i, by replacing all subsequences $(T, \tau, T, \tau, ..., \tau, T)$ by T.

3. The α-trace set of g is the α-trace set of its root.

Note that the 1-trace set of g is just the set of its concrete traces from which τ's have been left out. Two graphs g and h are *α-trace equivalent* - notation $g \approx_\alpha h$ - if they have the same α-trace set. Let us say that they are *hypertrace equivalent* - notation $g \approx h$ - if $g \approx_\alpha h$ for all ordinals α. Note that if $\lambda<\alpha$ then $g \approx_\alpha h$ implies $g \approx_\lambda h$. From this it immediately follows that if $\boldsymbol{G'} \subseteq \boldsymbol{G}$ is a *set* of process graphs then on $\boldsymbol{G'}$ the notion of α-trace equivalence stabilizes for some ordinal - i.e. there exists a *closure ordinal* α such that, for $g,h \in \boldsymbol{G'}$, $g \approx h$ iff $g \approx_\alpha h$. It will follow from the proof of Theorem 3.3 that the smallest ordinal with $(g \approx_\alpha h \Leftrightarrow g \approx_{\alpha+1} h)$ is a closure ordinal. Furthermore if $\boldsymbol{G}$ has cardinality β then β must be a closure ordinal. Next we prove that hypertrace equivalence coincides with coloured trace equivalence:

THEOREM 3.3 $g \approx h$ *if and only if* $g \equiv_c h$.

PROOF $\Rightarrow$: Let $\boldsymbol{G'}$ be a set of process graphs containing g,h and all their subgraphs and let α be the smallest ordinal such that, for $g',h' \in \boldsymbol{G'}$, $g' \approx_\alpha h'$ iff $g' \approx_{\alpha+1} h'$. If $g' \approx_{\alpha+1} h'$ then by definition g' and h' have the same γ-traces, for $\gamma \leq \alpha$. Since α-traces are recognizable from there form, this implies that g' and h' must have the same α-traces. Consider the colouring on g and h in which every node is coloured with its own α-trace set. Now a coloured trace $(C_0, a_1, C_1, a_2,..., a_k, C_k)$ of a node r with α-trace sets C_i is just an α-trace and by definition of α we have that r and r' have the same α-trace set only if they have the same α-traces, i.e. they have the same colour only if they have the same coloured traces. Hence the colouring is consistent.

Now $g \approx h \Rightarrow g \approx_{\alpha+1} h \Rightarrow$ g and h have the same coloured traces $\Rightarrow g \equiv_c h$.

$\Leftarrow$: Assume a consistent colouring on g and h such that the roots of g and h have the same colour. Then with transfinite induction on γ it is easy to prove that equally coloured nodes have the same γ-traces. □

Hence we find that $\approx$ is equivalent to $\equiv_c$, and hence to $\leftrightarrow_b$ (Theorem 3.2). Note that compared to *readiness semantics* (cf. OLDEROG & HOARE [102]), *possible-futures semantics* (cf. ROUNDS & BROOKES [112]) and *ready trace semantics* (cf. BAETEN, BERGSTRA & KLOP [10]) in an α-trace ($\alpha \geq 1$) we keep track of a lot more information. Apart from just all one-step exits from the endstate of a partial execution we are now able to see all traces (and higher traces) that can be chosen at every state during the execution.

The notion of hypertrace equivalence gives us an indication of the amount of extra information that is needed to turn trace equivalence into branching bisimulation

equivalence. Furthermore, it provides us with an idea of how to build a consistent colouring on a set of graphs by distinguishing more and more between nodes.
A construction similar to Definition 3.7 was used by MILNER [95] to characterize observation equivalence in the spirit of Definition 1.7.

As a tool for further analysis we have the following proposition:

PROPOSITION 3.4 *It is possible to colour the nodes of a root unwound process graph g in such a way that two nodes have the same colour iff they can be related by a rooted branching autobisimulation on g (relating* g *with itself). This colouring is rooted and consistent.*

PROOF For every root unwound process graph g the largest rooted branching autobisimulation on g is an equivalence relation on the nodes. It follows from the proof of Theorem 3.2 that every node can be labelled with its equivalence class as a colour, in order to obtain a rooted consistent colouring.

This colouring of a graph is called its *canonical colouring*. Note that two nodes r and s of a root unwound process graph g have the same colour with respect to its canonical colouring if and only if r,s ≠ root(g) and $(g)_r \underline{\leftrightarrow}_b (g)_s$ (the subgraph $(g)_r$ of g with root r is defined in the beginning of this section). In this case we say that r and s are *rooted branching bisimilar*. A root unwound graph is said to be in *normal form* if it has no τ-loops $r \rightarrow^{\tau} r$ and each node has a different colour with respect to its canonical colouring. Next we show that each root unwound process graph is rooted branching bisimilar with exactly one normal form (up to isomorphism).

DEFINITION 3.8 Let g be a root unwound process graph and consider its canonical colouring with colour set C. Let N(g) - the *normal form* of g - be the graph which can be found from g by contracting all nodes with the same colour and removing τ-loops. To be precise:

1. N(g) has colours C∈ C as its nodes.
2. N(g) has an edge $C \rightarrow^{a} C'$ (a≠τ) iff g has an edge $r \rightarrow^{a} r'$ such that C(r)=C and C(r')=C', where C(r) denotes the colour of the node r.
3. N(g) has an edge $C \rightarrow^{\tau} C'$ iff C≠C' and g has an edge $r \rightarrow^{\tau} r'$ with C(r)=C and C(r')=C'.

PROPOSITION 3.5 *For all root unwound process graphs* g: $g \underline{\leftrightarrow}_{rb} N(g)$.

PROOF Consider the canonical colouring on g, and the trivial colouring on N(g) in which each node (being a colour from **C**) is labelled by itself. Let R be the relation relating nodes from g and N(g) iff they have the same colour. Now it follows directly from the construction above that R is a rooted branching bisimulation between g and N(g). □

So in every rooted branching bisimulation equivalence class of root unwound process graphs there is a normal form. We proceed by showing that up to isomorphism there is only one.

DEFINITION 3.9 A *graph isomorphism* is a bijective relation R between the nodes of two process graphs g and h such that:

1. The roots of g and h are related by R
2. If R(r,s) and R(r',s') then $r \rightarrow^a r'$ is an edge in g iff $s \rightarrow^a s'$ is an edge in h ($a \in Act$).

Note that a graph isomorphism is just a bijective bisimulation, or a bijective branching bisimulation for that matter. Two graphs are isomorphic - notation $g \cong h$ - iff there exists an isomorphism between them. In that case g and h only differ with respect to the identity of the nodes. Note that $\cong$ is a congruence relation on process graphs.

THEOREM 3.6 (normal form theorem)
Let g *and* h *be root unwound graphs that are in normal form.*
Then $g \leftrightarrow_{rb} h$ *if and only if* $g \cong h$.

PROOF This follows since any bisimulation R: $g \leftrightarrow_{rb} h$ must be bijective:
(i) it is surjective because every node in g or h can be reached from the root; hence by Definition 1.4 every node is related to some node in the other graph;
(ii) it is injective since every node is related with at most one other node: if two different nodes in g are related to the same node in h, then these two are also related by a branching autobisimulation on g, and so with respect to the canonical colouring they have the same colour. But then by Definition 3.8 the nodes are identical, which is a contradiction. □

Theorem 3.6 says that each equivalence class in $G/\leftrightarrow_{rb}$ can be represented by one special element: its normal form. It follows that $g \leftrightarrow_{rb} h$ if and only if $N(g) \cong N(h)$.

4. COMPLETENESS PROOFS

In this section we will present the proofs of the completeness theorems 2.6, 2.7, 2.8 and 2.5. By means of a rather trivial adaption of the contents of this section one obtains the completeness theorems for CCS and ACP_τ (Theorems 2.12 and 2.14). The basic idea in these proofs is to establish a *graph rewriting system* on finite process graphs, which is confluent and terminating. Next we prove that (i) two normal forms (with respect to the graph rewriting system) are bisimilar iff they are equal (i.e. isomorphic), and furthermore that every rewriting step in the system (ii) corresponds to a proof step in the theory, and (iii) preserves bisimulation. Then we conclude:

- two finite graphs are bisimilar iff they have the same normal form
- if two graphs have the same normal form then the corresponding terms can be proved equal.

To start with, let us consider some definitions.

DEFINITION 4.1 Let $\boldsymbol{H} \subseteq \boldsymbol{G}$ be the set of *finite* process graphs and $\boldsymbol{H}^+ \subseteq \boldsymbol{G}_{\mathrm{BPA}}$ the set of finite, non-trivial process graphs. Here a process graph is *finite* if it has only finitely many paths.

Note that finite process graphs are acyclic and thus certainly root-unwound, and contain only finitely many nodes and edges. Later on, we will establish a correspondence between graphs from $\boldsymbol{H}^+$ and closed terms in BPA_τ, i.e. the signature of BPA together with the extra constant τ. Below we will use the results form the previous section, starting from Proposition 3.4.

DEFINITION 4.2 A pair (r,s) of nodes in a process graph g is called a pair of *double nodes* if $r \neq s$, $r,s \neq \mathrm{root}(g)$ and for all nodes t and labels $a \in \mathrm{Act}$: $r \rightarrow^a t \Leftrightarrow s \rightarrow^a t$.

DEFINITION 4.3 An edge $r \rightarrow^\tau s$ in a process graph g is called *manifestly inert* if $r \neq \mathrm{root}(g)$ and for all nodes t and labels $a \in \mathrm{Act}$ such that $(a,t) \neq (\tau,s)$: $r \rightarrow^a t \Rightarrow s \rightarrow^a t$.

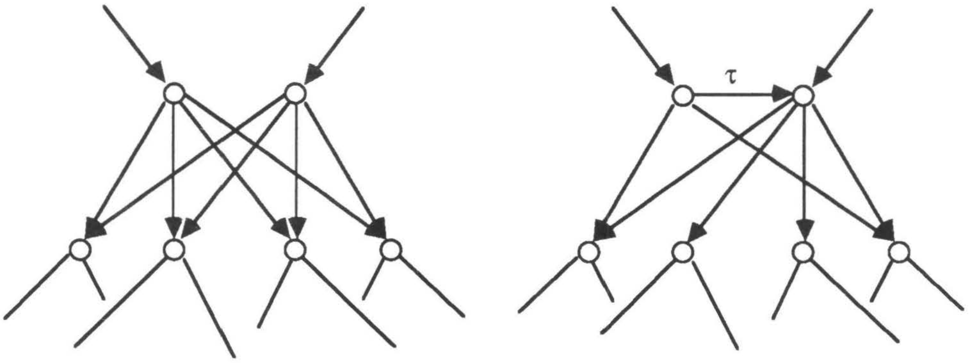


Figure 10. A pair of double nodes (left) and a manifestly inert τ-step.

Note that for finite process graphs g, the requirement r,s ≠ root(g) in Definition 4.2 is redundant. A τ-edge in a root unwound graph g is *inert* if it is between two rooted branching bisimilar nodes (i.e. nodes that have the same colour in the canonical colouring of g). For root unwound graphs it is easily checked that if (r,s) is a pair of double nodes or if $r \xrightarrow{\tau} s$ is manifestly inert, then r and s are rooted branching bisimilar. As one can see from Figure 10, it is essential in the Definitions 4.2 and 4.3 that this can be found by investigating the outgoing edges only up to one level. For this reason, in Definition 4.3 the τ-step is called *manifestly* inert, since it can be recognized as such. On ***H***, sharing of double nodes and contraction of manifestly inert τ-steps turns out to be strong enough to reduce a graph to its normal form. This means that in the reduction process all rooted branching bisimilar nodes become *manifestly* rooted branching bisimilar.

THEOREM 4.1 *A graph* g∈***H*** *without double nodes or manifestly inert edges is in normal form.*

PROOF Let g∈***H*** be a finite graph which is not in normal form. Then with respect to its canonical colouring (Proposition 3.4) it has at least one pair of different nodes with the same colour. Now define the *depth* d(s) of a node s as the number of edges in the longest path starting from s, and the *combined depth* of two nodes as the sum of their depths. Choose a pair (r,s) of different equally coloured nodes in g with minimal combined depth. Consequently we have:

(*) if r' and s' have the same colour and d(r') + d(s') < d(r) + d(s) then r'=s'.

Without loss of generality assume d(s)≤d(r). Then we prove the following two statements:

1. *if* $r \xrightarrow{a} t$ (a∈Act) *is an edge in* g *and* $(a,t)\neq(\tau,s)$, *then* $s \xrightarrow{a} t$ *is an edge in* g
2. *if* $s \xrightarrow{a} t$ (a∈Act) *is an edge in* g, *then either* $r \xrightarrow{\tau} s$ *or* $r \xrightarrow{a} t$ *is an edge in* g.

From these two statements we find that if $r \xrightarrow{\tau} s$ is an edge in g then it is manifestly inert, and if $r \xrightarrow{\tau} s$ is not an edge in g, then (r,s) is a pair of double nodes, which proves our theorem. Note that since r and s are different equally coloured nodes, they both must be different from the root.

ad 1: Let $r \xrightarrow{a} t$ be an edge in g and $(a,t)\neq(\tau,s)$. Since r and s have the same colour (hence the same coloured traces) we find that either $a=\tau$ and t has the same colour as r and s, or s has the coloured trace (C(r), a, C(t)). In the first case it follows from d(t) < d(r) and (*) that t=s, which is in contradiction with our assumption $(a,t)\neq(\tau,s)$. So s has a coloured trace (C(r), a, C(t)). Suppose that $s \xrightarrow{\tau} u$ for a node u with colour C(u)=C(s)=C(r), then it follows from d(u)<d(s) and (*) that u=r, contradicting d(u)<d(s)≤d(r). Hence there is a node u such that $s \xrightarrow{a} u$ and C(t)=C(u), and since d(t) + d(u) < d(r) + d(s) we conclude from (*) that t=u. Hence $s \xrightarrow{a} t$ is an edge in g.

ad 2: Let $s \xrightarrow{a} t$ be an edge in g. If C(t)=C(s)=C(r) then it follows from (*) and d(t)<d(s) that r=t, in contradiction with d(t)<d(s)≤d(r). So (C(s), a, C(t)) is a

coloured trace of s, and since r and s have the same colour (C(s), a, C(t)) is a coloured trace of r as well. Now if r has an outgoing τ-edge r $\rightarrow^{\tau}$ u to a node with the same colour C(r), then it follows from d(u) + d(s) < d(r) + d(s) and (*) that u=s. If r has no such edge, then it has an edge r $\rightarrow^{a}$ u with C(u)=C(t), and since d(u) + d(t) < d(r) + d(s) we find that u=t. Thus we proved that either r $\rightarrow^{\tau}$ s or r $\rightarrow^{a}$ t, which proves (2). □

Theorem 4.1 tells us that all we need to do to turn a finite graph g into its normal form is to repeatedly unify its pairs of double nodes and contract its manifestly inert edges. In the case of finite graphs this can be done in finitely many steps as follows:

DEFINITION 4.4 For any graph g$\in$ ***H*** the rewriting relation $\rightarrow_{H}$ is defined by the following two one-step reductions:
1. *sharing* a pair of double nodes (r,s): replace all edges t $\rightarrow^{a}$ r by t $\rightarrow^{a}$ s (if not already there, otherwise just remove t $\rightarrow^{a}$ r) and remove r together with all its outgoing edges from g;
2. *contracting* a manifestly inert step r $\rightarrow^{\tau}$ s: replace all edges t $\rightarrow^{a}$ r by t $\rightarrow^{a}$ s (if not already there, otherwise just remove t $\rightarrow^{a}$ r) and remove r together with all its outgoing edges from g.

PROPOSITION 4.2 *The rewriting relation* $\rightarrow_{H}$ *has the following properties:*
i. ***H*** *as well as* H^{+} *are closed under applications of* $\rightarrow_{H}$
ii. if g $\rightarrow_{H}$ h *then* g $\leftrightarrows_{rb}$ h
iii. $\rightarrow_{H}$ *is confluent and terminating.*

PROOF (i) In applications of $\rightarrow_{H}$ the root is never removed and in the resulting graph all nodes remain reachable from the root. Never two edges with the same label appear between the same two nodes. The graph also remains finite (and non-trivial).
(ii) Suppose (r,s) is a pair of double nodes or r $\rightarrow^{\tau}$ s is a manifestly inert edge in g, and g $\rightarrow_{H}$ h identifies the nodes r and s (= removes the node r). Let I be the identity relation on the nodes of h then I$\cup$\{\{r,s\}\} is a rooted branching bisimulation between g and h. This is easy to prove from the Definitions 4.2 and 4.3.
(iii) $\rightarrow_{H}$ is terminating since it decreases the number of nodes, and every finite process graph has finitely many nodes. Next, suppose g has two normal forms n and n', then by the definition of $\rightarrow_{H}$ n and n' are without pairs of double nodes and without manifestly inert edges. Thus by Theorem 4.1 n and n' are in normal form. By (ii) it follows that n $\leftrightarrows_{rb}$ n' and hence by Theorem 3.6 (normal form theorem) we have n $\cong$ n'. □

Next we will establish a correspondence between finite non-trivial graphs and closed BPA_τ-terms, such that the graph reductions of Definition 4.4 correspond to proof steps in BPA + H1,2.
Write $s \equiv_\Gamma t$ for $\Gamma \vdash s=t$ saying that s and t are equal *modulo* applications of axioms from Γ and the standard axioms for equality (reflexivity, commutativity, transitivity and replacement). It is quite easy to turn finite non-trivial graphs into BPA_τ-terms as follows. Let $T(BPA_\tau)$ be the set of closed BPA_τ terms.

DEFINITION 4.5 Let $<\cdot>: \boldsymbol{H}^+ \rightarrow T(BPA_\tau)$ be a mapping that satisfies

$$<g> = \sum_{\substack{r(g) \rightarrow^a s \text{ is an edge in } g;\\ s \text{ not an endnode}}} a\cdot<(g)_s> + \sum_{\substack{r(g) \rightarrow^b s \text{ is an edge in } g;\\ s \text{ is an endnode}}} b.$$

Here r(g) denotes the root node of $g \in \boldsymbol{H}^+$ and if p_i is a BPA_τ-term for $i \in I$, with $I=\{i_1,...,i_n\}$ a finite non-empty set of indices, then $\sum_{i \in I} p_i$ denotes a BPA_τ-term $p_{i_1} + ... + p_{i_n}$. Note that the notation $\sum_{i \in I} p_i$ does not determine the order and association of te terms p_i.

If $g \in \boldsymbol{H}^+$, $r(g) \rightarrow^a s$ is an edge in g, and s is not an endnode, then $(g)_s \in \boldsymbol{H}^+$. Furthermore, since $g \in \boldsymbol{H}^+$ is finite, r(g) has only finitely many outgoing edges, so the requirement of Definition 4.5 is well-defined. Moreover, with induction to the depth of its arguments, one easily proves that a mapping that meets this requirement exists. However, for $g \in \boldsymbol{H}^+$, this requirement determines <g> only modulo A1-A2.

PROPOSITION 4.3 *If* $g,h \in \boldsymbol{H}^+$ *and* $g \cong h$, *then* A1-A2 $\vdash$ <g> = <h>.
PROOF Trivial.

DEFINITION 4.6 The denotation [p] of a BPA_τ-term p in the graph domain $\boldsymbol{G}$, is defined by:

$[a] = a_{\boldsymbol{G}}$ for $a \in A_\tau$
$[x + y] = [x] +_{\boldsymbol{G}} [y]$
$[x\cdot y] = [x] \cdot_{\boldsymbol{G}} [y]$
where $a_{\boldsymbol{G}}$, $+_{\boldsymbol{G}}$ and $\cdot_{\boldsymbol{G}}$ are the interpretations in $\boldsymbol{G}$, of the constants and operators a, + and · of BPA_τ, as defined in Definition 2.1.

Now it turns out that terms of the form <g> (for $g \in \boldsymbol{H}^+$) are all of a specific shape, and for terms of this shape, <·> is a left-inverse of [·], modulo A1-A2. Consider the following definition:

DEFINITION 4.7 The set BT of *basic terms* over BPA_τ is inductively defined by:

1. For all $a \in Act$ we have $a \in BT$;
2. If $p,q \in BT$ then $(p + q) \in BT$ and for all $a \in Act$: $a\cdot p \in BT$.

LEMMA 4.4 *For* $g \in \boldsymbol{H}^+$, $<g>$ is a basic term and if $p \in BT$, then $<[p]> \equiv_{A1,2} p$.

PROOF With induction to the structure of terms:

- If $p = a$ ($a \in Act$) then $[p]$ is the one-edge graph labelled with a, and $< [p] > = p$.
- If $p = a \cdot u$ for some basic term u, then $[p]$ is the graph with an edge labelled a, followed by $[u]$.
  Then, $< [p] > = a \cdot < [u] >$ and so by induction we find that $< [p] > \equiv_{A1,2} a \cdot u$.
- Suppose $p = u + v$. One can easily see that for graphs g and h: $< g +_G h > \equiv_{A1,2} <g> + <h>$.
  Then: $A1\text{-}A2 \vdash < [u + v] > = < [u] > + < [v] > = u + v$ (by induction). □

LEMMA 4.5 (elimination)

For every closed BPA_τ*-term* p *there exists a basic term* q *such that* $A4\text{-}A5 \vdash p = q$.

PROOF By induction on the structure of p.

- If $p = a$ ($a \in Act$) then p is a basic term.
- If $p = u \cdot v$ and Lemma 4.5 can be proved for all terms smaller than p, then there exist basic terms u' and v' such that $A4\text{-}A5 \models u = u', v = v'$. Now suppose u' has the form $(\Sigma_j a_i \cdot w_i + \Sigma_j b_j)$, then we find:
  $$\begin{aligned} A4\text{-}A5 \models p = u' \cdot v' &= (\Sigma_i a_i \cdot w_i + \Sigma_j b_j) \cdot v' = \\ &= \Sigma_i (a_i \cdot w_i) \cdot v' + \Sigma_j b_j \cdot v' \text{ (by axiom A4)} \\ &= \Sigma_i a_i \cdot (w_i \cdot v') + \Sigma_j b_j \cdot v' \text{ (by axiom A5)} \\ &= \Sigma_i a_i \cdot q_i + \Sigma_j b_j \cdot v' \text{ for some basic terms } q_i \text{ (by induction)} \end{aligned}$$
  which is a basic term again.
- If $p = u + v$ then $A4\text{-}A5 \models p = u' + v'$ for basic terms u' and v', and the sum of two basic terms is again a basic term. □

PROPOSITION 4.6 *For all closed* BPA_τ*-terms* p *we have:* $A1\text{-}A2\text{+}A4\text{-}A5 \vdash <[p]> = p$.

PROOF If '$p=q$' is an instantiation of A4 or A5 (possibly in a context) then $< [p] > \equiv_{A1,2} < [q] >$. Now the proposition follows immediately from the Lemma's 4.4 and 4.5. □

This concludes the establisment of a correspondence between $\boldsymbol{H}^+$ and $T(BPA_\tau)$. Next we will show that every rewriting step on $\boldsymbol{H}^+$ corresponds to a proof step in BPA + H1-H2.

LEMMA 4.7 *Let* (r,s) *be a pair of dubble nodes or* $r \rightarrow^\tau s$ *be a manifestly inert* τ*-step in a process graph* g, *such that neither r nor s are endnodes, and let* $a \in Act$. *Then we have:* $BPA + H1\text{-}H2 \vdash a \cdot <(g)_r> = a \cdot <(g)_s>$.

PROOF In case (r,s) is a pair of dubble nodes r has an edge r $\rightarrow^a$ t iff s has an edge s $\rightarrow^a$ t and so $\langle(g)_r\rangle \equiv_{A1,2} \langle(g)_s\rangle$, hence $a\cdot\langle(g)_r\rangle = a\cdot\langle(g)_s\rangle$.

In case r $\rightarrow^\tau$ s is a manifestly inert τ-step we distinguish two subcases: First assume that r has more outgoing edges than only r $\rightarrow^\tau$ s. Then there must be basic terms p and q such that

(1) $\langle(g)_r\rangle \equiv_{A1,2} \tau\cdot\langle(g)_s\rangle + p$

(2) $\langle(g)_s\rangle \equiv_{A1,2} p + q$.

So we derive:

$$\begin{aligned} A1,2 + H2 \vdash a\cdot\langle(g)_r\rangle &= a\cdot(\tau\cdot\langle(g)_s\rangle + p) \text{ (by (1))} = \\ &= a\cdot(\tau\cdot(p + q) + p) \text{ (by (2))} = \\ &= a\cdot(p + q) \text{ (by applying H2)} \\ &= a\cdot\langle(g)_s\rangle \text{ (by (2)).} \end{aligned}$$

In case r has no more outgoing edges than r $\rightarrow^\tau$ s we have $\langle(g)_r\rangle = \tau\cdot\langle(g)_s\rangle$, hence

$$A5 + H1 \vdash a\cdot\langle(g)_r\rangle = a\cdot(\tau\cdot\langle(g)_s\rangle) = (a\cdot\tau)\cdot\langle(g)_s\rangle = a\cdot\langle(g)_s\rangle. \qquad \square$$

PROPOSITION 4.8 *If* g $\rightarrow_{\boldsymbol{H}}$ h *then* BPA + H1-H2 $\vdash \langle g\rangle = \langle h\rangle$.

PROOF On $\boldsymbol{H}$ the rewriting relation $\rightarrow_{\boldsymbol{H}}$ can be decomposed in the following elementary reductions:

Take a pair of double nodes (r,s) or a manifestly inert τ-step r $\rightarrow^\tau$ s and replace one edge t $\rightarrow^a$ r by t $\rightarrow^a$ s (if not already there, otherwise just remove t $\rightarrow^a$ r) and if r has no more incoming edges remove r together with all its outgoing edges from g. So it suffices to proof that if h is obtained from g by means of such an elementary reduction, we have $\langle g\rangle \equiv_\Gamma \langle h\rangle$, where Γ = BPA + H1-H2. From Definition 4.5 it follows that it even suffices to proof $\langle(g)_t\rangle \equiv_\Gamma \langle(h)_t\rangle$.

- First consider the case that neither r nor s are endnodes and there is no edge t $\rightarrow^a$ s in g. Then $\langle(g)_t\rangle \equiv_{A1,2} a\cdot\langle(g)_r\rangle + p$ for certain basic term p. Lemma 4.7 says $a\cdot\langle(g)_r\rangle \equiv_\Gamma a\cdot\langle(g)_s\rangle$, hence $\langle(g)_t\rangle \equiv_\Gamma a\cdot\langle(g)_s\rangle + p \equiv_{A1,2} \langle(h)_t\rangle$.
- In case t $\rightarrow^a$ s is an edge in g, and r,s are still assumed not to be endnodes we have $\langle(g)_t\rangle \equiv_{A1,2} a\cdot\langle(g)_r\rangle + a\cdot\langle(g)_s\rangle + p \equiv_\Gamma a\cdot\langle(g)_s\rangle + a\cdot\langle(g)_s\rangle + p \equiv_{A2,3} a\cdot\langle(g)_s\rangle + p \equiv_{A1,2} \langle(h)_t\rangle$.
- If (r,s) is a pair of double nodes than r is an endnode iff s is. In this case we have $\langle(g)_t\rangle \equiv_{A1,2} a + p \equiv_{A1,2} \langle(h)_t\rangle$ if t $\rightarrow^a$ s is not an edge in g
and $\langle(g)_t\rangle \equiv_{A1,2} a + a + p \equiv_{A2,3} a + p \equiv_{A1,2} \langle(h)_t\rangle$ otherwise.
- Finally if t $\rightarrow^a$ s is a manifestly inert τ-edge and s is an endnode in g, we have $\langle(g)_t\rangle \equiv_{A1,2} a\cdot\tau + p \equiv_{H1} a + p \equiv_{A1,2} \langle(h)_t\rangle$ if t $\rightarrow^a$ s is not an edge in g
and $\langle(g)_t\rangle \equiv_{A1,2} a\cdot\tau + a + p \equiv_{H1} a + a + p \equiv_{A2,3} a + p \equiv_{A1,2} \langle(h)_t\rangle$ otherwise. □

Now we are in the position to prove the completeness of BPA + H1-H2 with respect to $G_{\mathrm{BPA}}/{\leftrightarrow}_{rb}$:

PROOF OF THEOREM 2.6: (soundness) The fact that $(\mathcal{G}_{BPA}/\leftrightarroweq_{rb},+,\cdot,Act)$ is a model for BPA + H1-H2 follows easily by inspection of the axioms of BPA + H1-H2.
(completeness) Let $(\mathcal{G}_{BPA}/\leftrightarroweq_{rb},+,\cdot,Act) \models p=q$ for two closed BPA$_\tau$-terms p,q, then by definition $[p] \leftrightarroweq_{rb} [q]$. Let g and h be the unique normal forms of [p] and [q] with respect to $\rightarrow_{\mathcal{H}}$. By Proposition 4.2 we find $g \leftrightarroweq_{rb} [p] \leftrightarroweq_{rb} [q] \leftrightarroweq_{rb} h$. From Theorem 4.1 it follows that g and h must be in normal form in the sense of Section 3 and by the normal form theorem (Theorem 3.6) it then follows that $g \cong h$. Thus we find BPA + H1-H2 $\vdash p = <[p]> = <g> = <h> = <[q]> = q$ using Propositions 4.3, 4.6 and 4.8. So BPA + H1-H2 is a complete axiomatization of $\mathcal{G}_{BPA}/\leftrightarroweq_{rb}$. □

Next we will prove the other completeness theorems, using the earlier results in this section. In fact we will extend the graph rewriting system to one which is 'typical' for the corresponding bisimulation relation. The rewrite rules which are added to the system are derived from Figure 1: in case of η-bisimulation we will *saturate* the graph by exhaustively adding edges of the kind of Figure 1 (c), whereas in the case of delay bisimulation we add edges as in Figure 1 (b). For τ-bisimulation we do both. This way we obtain normal forms which are saturated and which turn out to be unique modulo rooted branching bisimulation. From there we establish the completeness result precisely in the same way as before.

DEFINITION 4.8 Let $a \in Act$, then:

1. The rewriting relation $\rightarrow_\eta$ is defined on $\mathcal{H}$ by the rule:
   if a graph has a path $s \rightarrow^a s_1 \rightarrow^\tau s'$ without an edge $s \rightarrow^a s'$ then add $s \rightarrow^a s'$.
2. The rewriting relation $\rightarrow_d$ is defined on $\mathcal{H}$ by the rule:
   if a graph has a path $s \rightarrow^\tau s_1 \rightarrow^a s'$ without an edge $s \rightarrow^a s'$ then add $s \rightarrow^a s'$.
3. Furthermore, we set: $\rightarrow_\tau = \rightarrow_\eta \cup \rightarrow_d$.

Applications of $\rightarrow_\eta$, $\rightarrow_d$ or $\rightarrow_\tau$ are referred to as *saturation steps* (cf. BERGSTRA & KLOP [21]).

PROPOSITION 4.9 *The relations $\rightarrow_\eta$, $\rightarrow_d$ and $\rightarrow_\tau$ satisfy the following properties:*

i. $\mathcal{H}$ as well as $\mathcal{H}^+$ are closed under applications of $\rightarrow_\eta$, $\rightarrow_d$ and $\rightarrow_\tau$

ii. $\rightarrow_\eta$, $\rightarrow_d$ and $\rightarrow_\tau$ are confluent and terminating.

PROOF (i) Directly from Definition 4.8.
(ii) (termination) Let $g \in \mathcal{H}$. Let n(g) be the (finite) number of nodes in g, l(g) be the number of labels and e(g) be the number of edges in g. Note that n(g) and l(g) are not changed by $\rightarrow_\eta$, $\rightarrow_d$ and $\rightarrow_\tau$ whereas e(g) increases with every saturation step. Since g is finite we find that $e(g) < n(g)\times l(g)\times n(g)$ and so $n(g)\times l(g)\times n(g) - e(g)$ is positive and decreasing with the number of saturation steps.
(confluence) $\rightarrow_\eta$, $\rightarrow_d$ and $\rightarrow_\tau$ do not eliminate redexes. □

So from Proposition 4.9 we find that any graph $g \in \boldsymbol{H}$ has unique normal forms with respect $\rightarrow_\eta$, $\rightarrow_d$ and $\rightarrow_\tau$. These are written as H(g), D(g) and T(g) and (in that order) are called η-, d- and τ-*saturated*. The latter is also often referred to as the *transitive closure* of τ-steps. Furthermore, saturation preserves the corresponding bisimulation:

PROPOSITION 4.10 *For all* $g,h \in \boldsymbol{H}$:

i. *if* $g \rightarrow_\eta h$ *then* $g \leftrightarrows_{r\eta} h$

ii. *if* $g \rightarrow_d h$ *then* $g \leftrightarrows_{rd} h$

iii. *if* $g \rightarrow_\tau h$ *then* $g \leftrightarrows_{r\tau} h$.

The proof of the Proposition 4.10 is straightforward.

THEOREM 4.11 (normal form theorem) *Let* $g,h \in \boldsymbol{H}$, *then*

i. *if g and h are η-saturated process graphs, then* $g \leftrightarrows_{r\eta} h$ *if and only if* $g \leftrightarrows_{rb} h$

ii. *if g and h are d-saturated process graphs, then* $g \leftrightarrows_{rd} h$ *if and only if* $g \leftrightarrows_{rb} h$

iii. *if g and h are τ-saturated process graphs, then* $g \leftrightarrows_{r\tau} h$ *if and only if* $g \leftrightarrows_{rb} h$.

PROOF We will only prove (i). The other cases proceed in the same way.

Suppose that R: $g \leftrightarrows_{r\eta} h$ then it is sufficient to prove that R is a rooted branching bisimulation:

(i) The roots of H(g) and H(h) are related and (iii) R satisfies the root condition.

(ii) If R(r,s) and $r \rightarrow^a r'$ then either $a=\tau$ and R(r',s), or $s \Rightarrow s_1 \rightarrow^a s_2 \Rightarrow s'$ such that $R(r,s_1)$ and R(r',s'). Let $t_1,...,t_k$ be such that $s_2 = t_0 \rightarrow^\tau t_1 \rightarrow^\tau \cdots \rightarrow^\tau t_k = s'$ ($k \geq 0$) then since g and h are η-saturated there are edges $s_1 \rightarrow^a t_i$ and so there is a path $s \Rightarrow s_1 \rightarrow^a s'$. □

COROLLARY

i. $g \leftrightarrows_{r\eta} h$ *if and only if* $H(g) \leftrightarrows_{rb} H(h)$ *if and only if* $N(H(g)) \cong N(H(h))$

ii. $g \leftrightarrows_{rd} h$ *if and only if* $D(g) \leftrightarrows_{rb} D(h)$ *if and only if* $N(D(g)) \cong N(D(h))$

iii. $g \leftrightarrows_{r\tau} h$ *if and only if* $T(g) \leftrightarrows_{rb} T(h)$ *if and only if* $N(T(g)) \cong N(T(h))$.

PROOF It follows by Proposition 4.10 that $H(g) \leftrightarrows_{r\eta} g$, $D(g) \leftrightarrows_{rd} g$ and $T(g) \leftrightarrows_{r\tau} g$. Now apply the normal form theorems 4.11 and 3.6. □

So we find that in each r$*$-bisimulation equivalence class of finite process graphs for $* \in \{\tau,\eta,d\}$ there is exactly one $*$-saturated process graph up to rooted branching bisimulation and exactly one $*$-saturated normal form up to isomorphism. In order to prove the completeness theorems we still need to prove that rewriting steps correspond to proof steps.

PROPOSITION 4.12 *For finite graphs g and h:*

i. *If* $g \rightarrow_\eta h$ *then* A1-A3 + H1,3 $\vdash$ <g> = <h>

ii. If $g \rightarrow_d h$ *then* A1-A3 + T2 $\vdash \langle g \rangle = \langle h \rangle$

iii. If $g \rightarrow_\tau h$ *then* A1-A3 + T1-3 $\vdash \langle g \rangle = \langle h \rangle$.

PROOF (i) If $r \rightarrow^a r' \rightarrow^\tau r'' \rightarrow$ is a path is g and $r \rightarrow^a r''$ is added in g to obtain h, then we find that $\langle (g)_r \rangle \equiv_{A1\text{-}3} \langle (g)_r \rangle + a \cdot \langle (g)_{r'} \rangle$
and $\langle (g)_{r'} \rangle \equiv_{A1\text{-}3} \tau \cdot \langle (g)_{r''} \rangle + \langle (g)_{r'} \rangle$ and hence:

$$\text{A1-A3 + H3} \vdash \langle (g)_r \rangle = \langle (g)_r \rangle + a \cdot (\tau \cdot \langle (g)_{r''} \rangle + \langle (g)_{r'} \rangle) =$$
$$= \langle (g)_r \rangle + a \cdot (\tau \cdot \langle (g)_{r''} \rangle + \langle (g)_{r'} \rangle) + a \cdot \langle (g)_{r''} \rangle \text{ (by H3)} =$$
$$= \langle (g)_r \rangle + a \cdot \langle (g)_{r''} \rangle = \langle (h)_r \rangle.$$

In case $r \rightarrow^a r' \rightarrow^\tau r''$ and r'' is an endnode we find:

$$\text{A1-A3 + H1,3} \vdash \langle (g)_r \rangle = \langle (g)_r \rangle + a \cdot (\tau + \langle (g)_{r'} \rangle) =$$
$$= \langle (g)_r \rangle + a \cdot (\tau \cdot \tau + \langle (g)_{r'} \rangle) \text{ (by H1)} =$$
$$= \langle (g)_r \rangle + a \cdot (\tau \cdot \tau + \langle (g)_{r'} \rangle) + a \cdot \tau \text{ (by H3)} =$$
$$= \langle (g)_r \rangle + a = \langle (h)_r \rangle.$$

From A1-A3 + H3 $\vdash \langle (g)_r \rangle = \langle (h)_r \rangle$ it easily follows that

A1-A3 + H3 $\vdash \langle g \rangle = \langle h \rangle$.

(ii) If $r \rightarrow^\tau r' \rightarrow^a r'' \rightarrow$ is a path is g and $r \rightarrow^a r''$ is added in g to obtain h, then:

$\langle (g)_r \rangle \equiv_{A1\text{-}3} \langle (g)_r \rangle + \tau \cdot \langle (g)_{r'} \rangle$ and
$\langle (g)_{r'} \rangle \equiv_{A1\text{-}3} a \cdot \langle (g)_{r''} \rangle + \langle (g)_{r'} \rangle$ and hence:

$$\text{A1-A3 + T2} \vdash \langle (g)_r \rangle = \langle (g)_r \rangle + \tau \cdot (a \cdot \langle (g)_{r''} \rangle + \langle (g)_{r'} \rangle) =$$
$$= \langle (g)_r \rangle + a \cdot \langle (g)_{r''} \rangle \text{ (by T2 and A3)} = \langle (h)_r \rangle.$$

In case $r \rightarrow^a r' \rightarrow^\tau r''$ and r'' is an endnode we simply leave out $\cdot \langle (g)_{r''} \rangle$ in the argument above. Hence A1-A3 + T2 $\vdash \langle g \rangle = \langle h \rangle$.

(iii) Immediately from (i) and (ii). Note that H1 = T1 and H3 = T3. □

PROOFS OF THE THEOREMS 2.5, 2.7 AND 2.8

The soundness theorems follow easily after inspection of the axioms. Of the completeness theorems we only prove Theorem 2.7. The others proceed in the same way.

Let $(\mathcal{G}_{BPA}/\leftrightarrows_{r\eta},+,\cdot,Act) \models p=q$ for two closed BPA$_\tau$-terms p,q, then by definition $[p] \leftrightarrows_{r\eta} [q]$. Let g and h be the unique normal forms of [p] and [q] with respect to $\rightarrow_\eta$. By Proposition 4.10 we find $g \leftrightarrows_{r\eta} [p] \leftrightarrows_{r\eta} [q] \leftrightarrows_{r\eta} h$. The graphs g and h must be η-saturated and by the normal form theorem (4.11) it then follows that g $\leftrightarrows_{rb}$ h. Thus we find BPA + H1-H3 $\vdash p = \langle [p] \rangle = \langle g \rangle = \langle h \rangle = \langle [q] \rangle = q$ using Propositions 4.6 and 4.12 and Theorem 2.6. So BPA + H1-H3 is a complete axiomatization of $\mathcal{G}_{BPA}/\leftrightarrows_{r\eta}$. □

5. FEATURES

In this section we list the main features of branching bisimulation semantics that occurred to us. We concentrate on the differences and similarities with τ-bisimulation semantics.

5.1. BRANCHING TIME

The main difference between branching and τ-bisimulation semantics is that the former notion preserves the branching structure of processes whereas the latter does not. This has been elaborated in the Sections 1 and 3. If one argues that branching equivalence is too fine, since it does not correspond to a natural testing scenario, the same argument can be used to move from τ-bisimulation to one of the decorated trace equivalences, which are even coarser. On the other hand, if one favours τ-bisimulation over the decorated trace semantics since it preserves the internal structure of processes and is therefore independent of a particular testing scenario, a systematic application of this argument points in the direction of the finer notion of branching bisimulation semantics.

5.2. EQUIVALENCE VERSUS CONGRUENCE

τ-bisimulation equivalence is not a congruence for +, and therefore τ-bisimulation congruence is defined as the closure of τ-bisimulation equivalence under contexts, or by means of the root condition. In this respect η-, delay and branching bisimulation behave exactly the same. However, each τ-bisimulation equivalence class consists of at most two τ-bisimulation congruence classes (this follows from Exersice 7.6 of HENNESSY in MILNER [92]), as is the case for delay bisimulation, whereas η- and branching bisimulation equivalence classes may contain many congruence classes. Nevertheless, for all four bisimulations there exists a close relationship between rooted and non-rooted bisimulation, since the root condition (Definition 2.2) only works on the root nodes:

THEOREM 5.1 *For all root unwound graphs g and h and* $* \in \{\tau, b, \eta, d\}$ *we have:*
$g \underline{\leftrightarrow}_* h$ *if and only if* $\tau{\cdot}g \underline{\leftrightarrow}_{r*} \tau{\cdot}h$.

PROOF If R is a ∗-bisimulation between g and h and r,s are the roots of τ·g and τ·h then $R \cup \{r,s\}$ is a rooted ∗-bisimulation between τ·g and τ·h. On the other hand, if R is a rooted ∗-bisimulation between τ·g and τ·h, then the roots of g and h are related by R, so R restricted to the nodes of g and h is a ∗-bisimulation between g and h. □

This theorem provides us with a tool to decide upon ∗-bisimulation equivalence, using the axiom systems of ∗-bisimulation congruence.

5.3. DIVERGENCE

In the literature on bisimulation semantics roughly three ways are suggested for treating divergence (= infinite τ-paths). The original notion of τ-bisimulation equivalence (HENNESSY & MILNER [72], MILNER [92] and PARK [103]) abstracted from all divergencies; the first two graphs of Figure 11 are equivalent, as well as the two graphs of Figure 8.

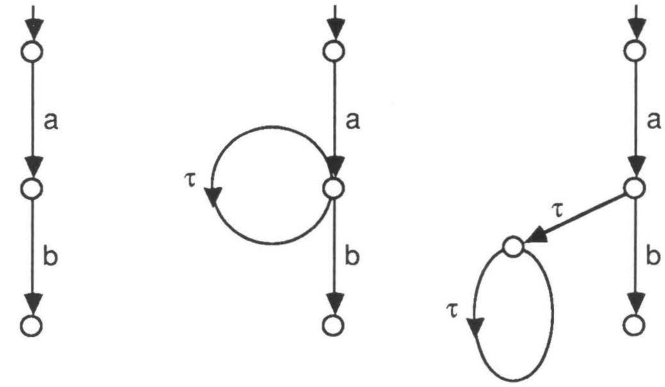


Figure 11. Three ways of modeling divergence.

These identifications can be justified by an appeal to fairness (MILNER [92], BAETEN, BERGSTRA & KLOP [9]), and play a crucial role in many protocol verifications. In BERGSTRA, KLOP & OLDEROG [23] the corresponding semantics is refered to as *bisimulation semantics with fair abstraction.* A variant were divergence is taken into account, in the sence that the first two graphs of Figure 11 are distinguished, as well as the two graphs of Figure 8, was proposed in HENNESSY & PLOTKIN [74] for τ-bisimulation and in MILNER [93] for delay bisimulation. In both cases a complete axiomatization is provided in WALKER [126]. In these semantics the basic notion is a *preorder* rather then an equivalence, and divergence is identified with underspecification. The induced equivalences identify the last two graphs of Figure 11, which are distinghuished in τ-bisimulation semantics with fair abstraction. Hence the two notions are incomparable. A semantics that refines both notions was proposed in BERGSTRA, KLOP & OLDEROG [23] under the name *bisimulation semantics with explicite divergence.*

η-, delay and branching bisimulation as presented in this chapter are all based on the variant of τ-bisimulation with fair abstraction. However it is completely straightforward to generalize the *τ-bisimulation preorder* of HENNESSY & PLOTKIN [74] to a *η-bisimulation preorder*, and the *delay bisimulation preorder* of MILNER [93] to a *branching bisimulation preorder*. Also it is not difficult to define *η-*, *delay* and *branching bisimulation with explicit divergence* in the spirit of BERGSTRA, KLOP & OLDEROG [23]. For branching bisimulation the definition can conveniently be given in terms of coloured traces.

DEFINITION 5.1 A node in a coloured graph is *divergent* if it is the starting point of an infinite path of which all nodes have the same colour. A colouring *preserves divergence* if no divergent node has the same colour as a non-divergent node. Two graphs g and h are *(rooted) branching bisimulation equivalent with explicit divergence* if there exists a (rooted) consistent divergence preserving colouring on g and h for which they have the same coloured trace set.

5.4. ADEQUACY FOR MODAL LOGICS

As mentioned in the introduction, τ-bisimulation semantics is not adequate for a modal logic with 'eventually' operator. From the example in the introduction of this chapter one can see that the problem originates from the circumstance that τ-bisimulation equivalence does not preserve the branching structure of processes, and indeed one can easily prove that such an operator would cause no problems in branching bisimulation semantics, at least not in the variant with explicit divergence. In fact, a much stronger result has been proved in DE NICOLA & VAANDRAGER [45].

The Computation Tree Logic CTL* (EMERSON & HALPERN [49]) is a very powerful logic, combining both branching time and linear time operators. It is a generalization of CTL (CLARKE & EMERSON [37]), that contains only branching time operators. CTL* is interpreted on Kripke structures (directed graphs of which the nodes are labelled with sets of atomic propositions). DE NICOLA & VAANDRAGER [45] established a translation from process graphs to Kipke structures, so that CTL* can also be regarded as a logic on process graphs. One of the operators of CTL/CTL*, the nexttime operator X, makes it possible to see when an (invisible) action takes place, and is therefore incompatible with abstraction. This operator was also criticized by LAMPORT [82]. BROWNE, CLARKE & GRÜMBERG [34] found that CTL-X and CTL*-X induce the same equivalence on Kripke structures, which they characterized as *stuttering equivalence*. In DE NICOLA & VAANDRAGER [45] branching bisimulation, after being translated to Kripke structures, is shown to coincide with stuttering equivalence. (To be precise, they consider two variants of CTL*, that correspond to two variants of stuttering equivalence and two variants of branching bisimulation, namely *divergence blind branching bisimulation* (our notion with fair abstraction) and *divergence sensitive branching bisimulation* (defined as branching bisimulation with explicit divergence above, but also considering endnodes to be divergent). The stuttering equivalence of BROWNE, CLARKE & GRÜMBERG [34] is the divergence sensitive variant.) Hence (divergence sensitive) branching bisimulation is adequate for CTL*-X. Since the eventually operator of GRAF & SIFAKIS [66] can be expressed in CTL*-X, this implies that it causes no problems in branching bisimulation semantics.

5.5. MODAL CHARACTERIZATIONS

It is well known (cf. HENNESSY & MILNER [73]) that observation equivalence can be characterized by means of a simple modal langage, called Hennessy-Milner logic (HML). The question arises if such a result can also be obtained for branching equivalence. As pointed out above, CTL-X characterizes branching equivalence, but this language is rather strong. Another possibility is adding the eventually operator to

HML. It remains to be determined for which classes of process graphs HML + 'eventually' is adequate. In DE NICOLA & VAANDRAGER [45] it has been shown that adding an 'until' operator to HML is sufficient.

5.6. BACK AND FORTH BISIMULATIONS

In DE NICOLA, MONTANARI & VAANDRAGER [44] it has been established that if in the definition of $*$-bisimulation, for $* \in \{\tau,b,\eta,d\}$, it is required that moves in the one process can be simulated by the other process, not only when going forward but also when going back in history, these modified notions all coincide with branching bisimulation. This also yields another modal characterization of branching bisimulation, namely HML with backward modalities.

5.7. PRACTICAL APPLICATIONS OF BRANCHING TIME

The extra identifications made in τ-bisimulation semantics on top of branching bisimulation semantics can be cumbersome in certain applications of the theory. See the remark in the introduction.

5.8. REFINEMENT OF ACTIONS

For sequential processes branching bisimulation is preserved under refinement of actions, whereas τ-bisimulation is not. This was established in VAN GLABBEEK & WEIJLAND [63], see the next section of this chapter. A proof can also be found in DARONDEAU & DEGANO [39].

5.9. AXIOMATIZATIONS AND REWRITE SYSTEMS

All $*$-bisimulations ($* \in \{\tau,b,\eta,d\}$) have relatively simple equitional characterizations (see Section 2), but the axiom system for branching bisimulation can easily be turned in a complete term rewriting system, which is not the case for the other notions.

5.10. COMPLEXITY

In GROOTE & VAANDRAGER [68] an algorithm is presented for deciding branching bisimulation equivalence between finite-state processes, with (time) complexity $O(k+n\cdot m)$. Here k is the size of Act, n is the number of nodes in the investigated process graphs and m the number of edges. The fastest algorithm for τ-bisimulation equivalence up till now has complexity $O(k\cdot n^{2.376})$. In general $n \leq m \leq k\cdot n^2$, so it depends on the density of edges in a graph which algorithm is faster. In a trial implementation of the scheduler of MILNER [92], reported in GROOTE & VAANDRAGER [68], branching bisimulation turned out to be much faster. Furthermore, it turned out that in such automatic verifications the space complexity was a much more serious handicap then the time complexity (the τ-bisimulation tools suffered from lack of memory already by processes with 15.000 states). The space complexity of the algorithm of GROOTE & VAANDRAGER [68] is $O(n+m)$, which is less than the space complexity of τ-bisimulation.

5.11. CORRESPONDENCE

Finally we present a theorem which tells us that in quite a number of cases observation and branching bisimulation equivalence are the same. For instance, consider the practical applications where implementations are verified by proving them equal to some specification (after having abstracted from a set of unobservable actions of course). In many such cases, the *specification* does not involve any τ-steps at all: in fact all τ-steps that occur in the verification process originate from the abstraction procedure which is carried out on the implementation.

As it turns out, in all such cases there is no difference between observation and branching bisimulation equivalence. For this reason we may expect many verifications involving observation equivalence to be valid in the stronger setting of branching bisimulation as well. In particular this is the case for all protocol verifications in τ-bisimulation semantics known to the authors.

THEOREM 5.2 *Suppose* g *and* h *are two graphs, and* g *is without τ-labelled edges. Then:*

i. $g \mathrel{\underline{\leftrightarrow}}_{\tau} h$ *if and only if* $g \mathrel{\underline{\leftrightarrow}}_{b} h$

ii. $g \mathrel{\underline{\leftrightarrow}}_{r\tau} h$ *if and only if* $g \mathrel{\underline{\leftrightarrow}}_{rb} h$.

PROOF Let R be the *largest* (rooted) τ-bisimulation between g and h. We show that R is even a (rooted) branching bisimulation. Assume that $R(r,s)$ and $r \rightarrow^{a} r'$ is an edge in g, then either $a=\tau$ and $R(r',s)$ - contradicting the absence of τ-edges in g - or in h there is a path $s \Rightarrow s_1 \rightarrow^{a} s_2 \Rightarrow s'$ and $R(r',s')$. Assume $s \Rightarrow s_1$ has the form $s = v_0 \rightarrow^{\tau} v_1 \rightarrow^{\tau} \cdots \rightarrow^{\tau} v_m = s_1$ $(m \geq 0)$ then it follows from $s \rightarrow^{\tau} v_1$ and $R(r,s)$ that for some r_1: $r \Rightarrow r_1$ and $R(r_1,v_1)$. Since g has no τ-edges we find that $r=r_1$. Repeating this argument m times we find that $R(r,v_i)$ and $R(r,s_1)$.

Furthermore, since $R(r,s_1)$ and $s_1 \rightarrow^{a} s_2$ we find that $r \rightarrow^{a} r''$ (g has no τ-steps) such that $R(r'',s_2)$. Since $s_2 = w_0 \rightarrow^{\tau} w_1 \rightarrow^{\tau} \cdots \rightarrow^{\tau} w_n = s'$ it follows from the same argument as before that $R(r'',w_i)$ and $R(r'',s')$. Thus we find $R(r',s')$, $R(s',r'')$ and $R(r'',s_2)$ and since R is the largest rooted τ-bisimulation we have $R(r',s_2)$.

On the other hand, if $R(r,s)$ and $r \rightarrow^{a} r'$ is an edge in h, then either $a=\tau$ and $R(r',s)$ or directly $s \rightarrow^{a} s'$ such that $R(r',s')$, since g contains no τ-edges. □

For η- instead of branching bisimulation equivalence this theorem was already proven in BAETEN & VAN GLABBEEK [11]. From Theorem 5.1 we easily find that for graphs g and h:

g is without τ-edges $\Rightarrow$ ($\tau \cdot g \mathrel{\underline{\leftrightarrow}}_{r\tau} \tau \cdot h \Rightarrow \tau \cdot g \mathrel{\underline{\leftrightarrow}}_{rb} \tau \cdot h$).

6. REFINEMENT

Virtually all semantic equivalences employed in theories of concurrency are - as in this thesis - defined in terms of *actions* that concurrent systems may perform. Mostly, these actions are taken to be *atomic*, meaning that they are considered not to be divisible into smaller parts. In this case, the defined equivalences are said to be based on *action atomicity*.

However, in the top-down design of distributed systems it might be fruitful to model processes at different levels of abstraction. The actions on an abstract level then turn out to represent complex processes on a more concrete level. This methodology does not seem compatible with non-divisibility of actions and for this reason PRATT [108], LAMPORT [83] and others plead for the use of semantic equivalences that are not based on action atomicity.

As indicated in CASTELLANO, DE MICHELIS & POMELLO [36], the concept of action atomicity can be formalized by means of the notion of *refinement of actions*. A semantic equivalence is *preserved under action refinement* if two equivalent processes remain equivalent after replacing all occurrences of an action a by a more complex process r(a). In particular, r(a) may be a sequence of two actions a_1 and a_2. An equivalence is strictly based on action atomicity if it is not preserved under action refinement.

In the previous sections in this chapter we argued that Milner's notion of observation equivalence does not respect the branching structure of processes, and proposed the finer notion of *branching bisimulation equivalence* which does. In this section we moreover find, that observation equivalence is not preserved under action refinement, whereas branching bisimulation equivalence is.

From the axioms T3 (see Table 2), it is easy to show why the notion of observation congruence is not preserved under refinement of actions: replacing the action a by the term bc, we obtain $bc(\tau x + y) = bc(\tau x + y) + bcx$, which obviously is not valid in $G/\leftrightarroweq_{r\tau}$. Applying T3, we *do* find $bc(\tau x + y) = b(c(\tau x + y) + cx)$, unfortunately denoting a different process however.

In this section we will prove that branching equivalence *is* preserved under refinement of actions, and so it allows us to look at actions as abstractions of much larger structures. We will present our result in the style of BPA, and indicate afterwards how our construction can be adapted to obtain refinement theorems in the style of CCS and ACP. Put $A=Act\backslash\{0\}$ (or $A=Act\backslash\{0,\surd\}$ if there are $\surd$-labels around). Consider the following definitions.

DEFINITION 6.1 (substitution) Let $r: A \rightarrow \mathbf{G}_{BPA}$ be a mapping from observable actions to graphs, and suppose $g \in \mathbf{G}_{BPA}$. Then, the graph r(g) can be found as follows.

For every edge $r \rightarrow^{a} r'$ ($a \in A$) in g, take a copy $\underline{r(a)}$ of r(a) ($\in G_{BPA}$). Next, identify r with the root node of $\underline{r(a)}$, and r' with all endnodes of $\underline{r(a)}$, and remove the edge $r \rightarrow^{a} r'$.

Note that in this definition it is never needed to identify r and r', since r(g) is non-trivial. This way, the mapping r is extended to the domain G_{BPA}. Note that since $\tau \notin A$, τ-edges cannot be substituted by graphs. Finally, observe that every node in g is a node in r(g).

DEFINITION 6.2 (preservation under action refinement) An equivalence $\approx$ on G_{BPA} is said to be *preserved under refinement of actions* if for every mapping r: $A \rightarrow G_{BPA}$, we have: $g \approx h \Rightarrow r(g) \approx r(h)$.

In other words, an equivalence $\approx$ is preserved under refinement if it is a congruence with respect to every substitution operator r.
Starting from a relation R: $g \leftrightarrows_{rb} h$, we construct a branching bisimulation r(R): $r(g) \leftrightarrows_{rb} r(h)$, proving that preserving branching congruence, every edge with a label from A can be replaced by a root unwound non-trivial graph.

DEFINITION 6.3 Let r: $A \rightarrow G_{BPA}$ be a mapping from observable actions to graphs, $g,h \in G_{BPA}$ and R: $g \leftrightarrows_{rb} h$. Now r(R) is the smallest relation between nodes of r(g) and r(h), such that:

1. $R \subseteq r(R)$.
2. If $r \rightarrow^{a} r'$ and $s \rightarrow^{a} s'$ ($a \in A$) are edges in g and h such that R(r,s) and R(r',s'), and both edges are replaced by copies $\underline{r(a)}$ and $\overline{r(a)}$ of r(a) respectively, then nodes from $\underline{r(a)}$ and $\overline{r(a)}$ are related by r(R) iff they are copies of the same node in r(a).

Edges $r \rightarrow^{a} r'$ and $s \rightarrow^{a} s'$ ($a \in A$) such that R(r,s) and R(r',s'), will be called *related* by R, as well as the copies $\underline{r(a)}$ and $\overline{r(a)}$ that are substituted for them. Observe, that on nodes from g and h the relation r(R) is equal to R. Note that if r(R)(r,s), then r is a node in g iff s is a node in h.

THEOREM 6.1 (refinement) *Branching congruence is preserved under refinement of actions.*

PROOF We prove that R: $g \leftrightarrows_{rb} h \Rightarrow$ r(R): $r(g) \leftrightarrows_{rb} r(h)$ by checking the requirements. For convenience, in the definition of branching equivalence (Definition 1.4), we omit the requirement of the existence of a path $s_2 \Rightarrow s'$, as it is redundant (see the remark just after Definition 1.6). Then we find:

i. The root nodes of r(g) and r(h) are related by r(R).

ii. Assume r(R)(r,s) and in r(g) there is an edge r $\rightarrow^{a}$ r'. Then there are two possibilities (similarly in case r $\rightarrow^{a}$ r' stems from r(h)):

(1) The nodes r and s originate from g and h. Then R(r,s), and by the construction of r(g) we find that either a=τ and r $\rightarrow^{\tau}$ r' was already an edge in g, or g has an edge r $\rightarrow^{b}$ r* and r $\rightarrow^{a}$ r' is a copy of an initial edge from r(b).

In the first case it follows from R: g $\leftrightarrow_{rb}$ h that either R(r',s) - hence r(R)(r',s) - or in h there is a path s $\Rightarrow$ s_1 $\rightarrow^{\tau}$ s' such that R(r,s_1) and R(r',s'). By definition of refinement, the same path also exists in r(h), and thus we have r(R)(r,s_1) and r(R)(r',s').

In the second case there must be a corresponding path s $\Rightarrow$ s_1 $\rightarrow^{b}$ s* in h such that R(r,s_1) and R(r*,s*). Then, in r(h) we find a path s $\Rightarrow$ s_1 $\rightarrow^{a}$ s' (by replacing $\rightarrow^{b}$ by r(b)) such that r(R)(r,s_1) and r(R)(r',s').

(2) The nodes r and s originate from related copies $\underline{r(b)}$ and $\overline{r(b)}$ of a substituted graph r(b) (for some b$\in$A), and are no copies of root or endnodes in r(b). Then r $\rightarrow^{a}$ r' is an edge in $\underline{r(b)}$. From r(R)(r,s) we find that r and s are copies of the same node from r(b). So, there is an edge s $\rightarrow^{a}$ s' in $\overline{r(b)}$ where s' is a copy of the node in r(b), corresponding with r'. Clearly r(R)(r',s').

iii. Since for nodes from g and h we have r(R)(r,s) iff R(r,s), the root condition is satisfied. □

With respect to closed BPA$_\tau$-terms, the refinement theorem can be proved much easier by syntactic analysis of proofs, instead of working with equivalences between graphs. For observe that the axioms A1-A5 + H1-H2, that form a complete axiomatization of branching congruence for closed terms, do *not* contain any occurrences of (atomic) actions from A. Now assume we have a proof of some equality s=t between closed terms, then this proof consists of a sequence of applications of axioms from A1-A5 + H1-H2. Since all these axioms are universal equations without actions from A, the actions from s and t can be replaced by general variables, and the proof will still hold. Hence, every equation is an instance of a universal equation *without* any actions. Immediately we find that we can substitute arbitrary closed terms for these variables, obtaining refinement for closed terms.

Nevertheless, the semantic proof of the refinement theorem is important since it also holds for larger graphs from $\mathcal{G}_{BPA}$that are not representable by closed BPA$_\tau$-terms.

In the setting of BCCS, a substitution should be a mapping r: A $\rightarrow$ $\mathcal{G}_{CCS}\backslash\{0\}$, where 0 denotes the trivial graph. Then the semantic proof of the refinement theorem goes exactly as in the setting of BPA. However the syntactic proof breaks down on the absence of general sequential composition and on the presence of actions in the axioms for branching congruence. In the setting of basic ACP, Definition 6.1 should be adapted such that r' is identified not with all endnodes of $\underline{r(a)}$, but with all nodes of $\underline{r(a)}$ that have an outgoing termination edge. These termination edges should then be deleted. Furthermore if certain parts in the resulting graph have become disconnected from the root, they should be deleted as well. Now both the semantic and the syntactic proof of the refinement theorem remain valid. Finally it should be noted that refinement as defined in this section is a meaningful notion that can be used in the design of systems *only* if these system are assumed to be sequential (i.e. performing only one action at a time). In the presence of parallel composition, process graphs as presented here are not sufficiently expressive for defining a refinement operator. For this pupose one may better use causality based models of concurrency, such as event structures or Petri nets. This will be the topic of the following chapter.

Chapter IV

Refinement of Actions in Causality Based Models

Rob van Glabbeek & Ursula Goltz

In this chapter we consider an operator for refinement of actions to be used in the design of concurrent systems. Actions on a given level of abstraction are replaced by more complicated processes on a lower level. This is done in such a way that the behaviour of the refined system may be inferred compositionally from the behaviour of the original system and from the behaviour of the processes substituted for actions. We define this refinement operation for causality based models like event structures and Petri nets. For Petri nets, we relate it to other approaches for refining transitions.

Notes This chapter appeared as Arbeitspapiere der GMD 428, Sankt Augustin 1990, and will be published in: Proceedings of the REX Workshop on Stepwise Refinement of Distributed Systems: Models, Formalism, Correctness, Mook, The Netherlands 1989, eds. J.W. de Bakker, W.-P. de Roever & G. Rozenberg, LNCS 430, Springer-Verlag, 1990, pp. 267-300.

The research of the second author was supported by Esprit Basic Research Action 3148 (DEMON).

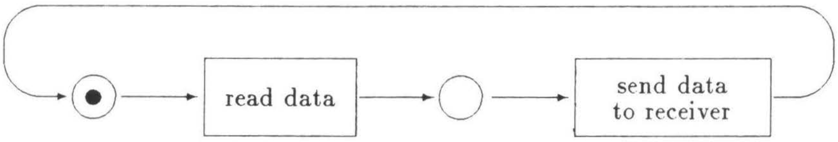
Contents

Introduction

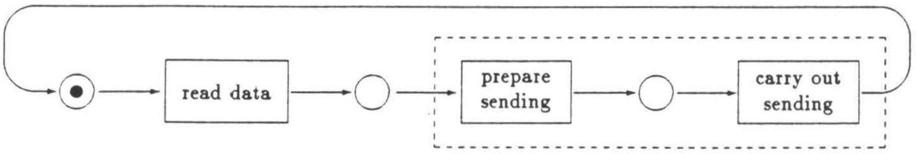
In this chapter we consider the design of concurrent systems in the framework of approaches where the basic building blocks are the actions which may occur in a system. By an action we understand here any activity which is considered as a conceptual entity on a chosen level of abstraction. This allows to design systems in a top–down style, changing the level of abstraction by interpreting actions on a higher level by more complicated processes on a lower level. We refer to such a step in the design of a system as *refinement of actions.* An action could be refined by the sequential execution of several subactions, or by activities happening independently in parallel. One could also implement an action by a set of alternatives, of which only one should be taken.

0.1 Example

Consider the design of a sender, repeatedly reading data and sending them to a certain receiver. A first description of this system is given by the Petri net shown below. An introduction to Petri nets and the way they model concurrent systems can be found in REISIG [110]; the refinement mechanism used in this example will be treated formally in Section 4.

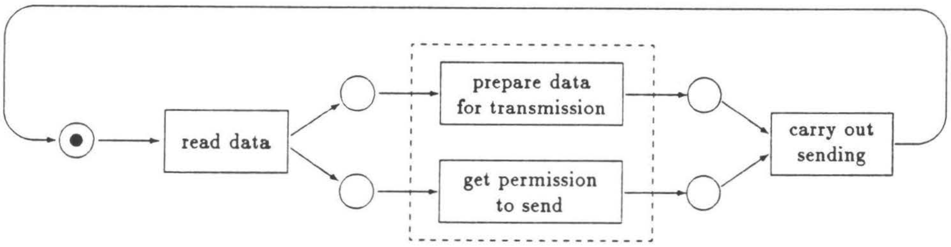


On a slightly less abstract description level the action "send data to receiver" might turn out to consist of two parts "prepare sending" and "carry out sending", to be executed sequentially. This corresponds to the following refined Petri net.



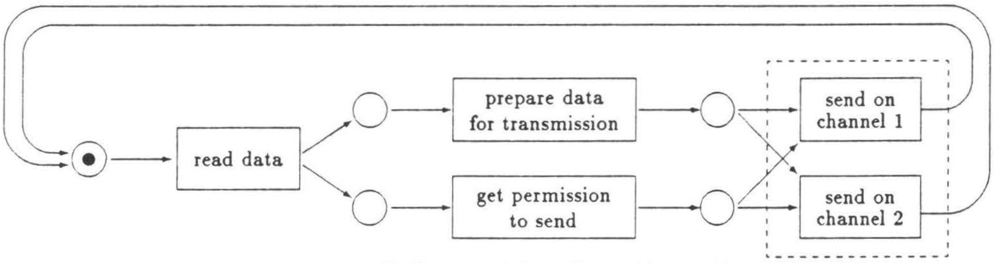
Refinement by a sequential process

Then the action "prepare sending" may be decomposed in two independent activities "prepare data for transmission" and "get permission to send", to be executed on different processors:



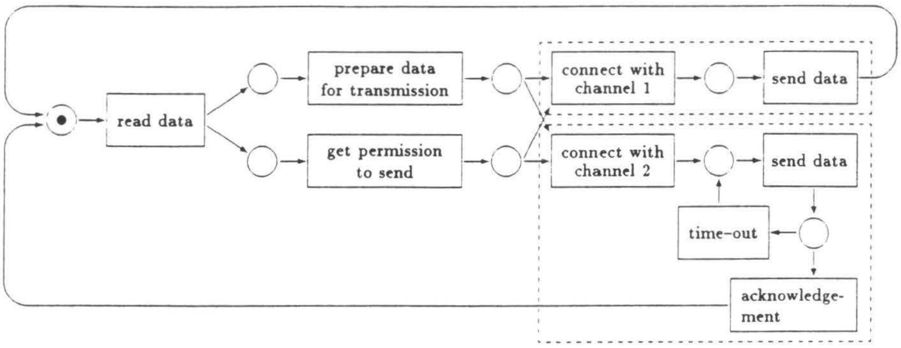
Refinement by a parallel process

Furthermore it may turn out that there are two alternative channels for sending messages. Each time the sender should choose one of them to send a message, perhaps depending on which one is available at the moment.



Refinement by alternative actions

On an even more concrete level of abstraction, channel 2 may happen to be rather unreliable, and getting a message at the other end requires the use of a communication protocol. On the other hand, channel 1 may be found to be reliable, and does not need such a precaution.



Refinement by an infinite process

Here we see that it may happen that the process we have substituted for the action "send on channel 2" does not terminate. It may happen that the attempt of sending data always fails and this prevents the system of reaching its initial state again.

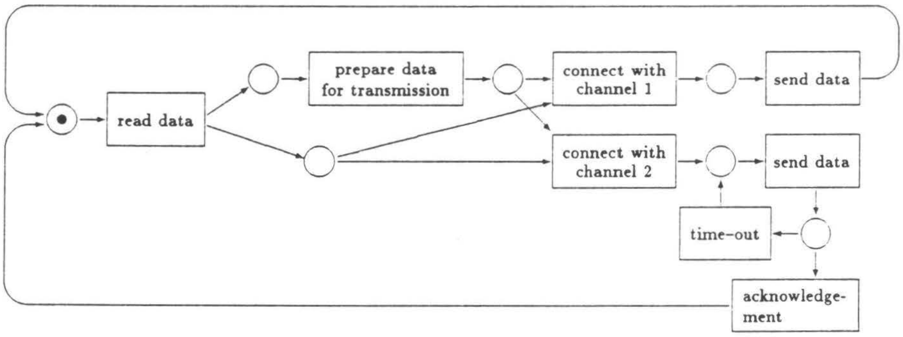
Our aim is to define an operator for refinement of actions, taking as arguments a system description on a given level of abstraction and an interpretation of (some of) the actions on this level by more complicated processes on a lower level, and yielding a system description on the lower level. This should be done in such a way that the behaviour of the refined system may be inferred compositionally from the behaviour of the original system and from the behaviour of the processes substituted for actions.

As illustrated above, we want to allow to substitute rather general kinds of behaviours for actions. We even allow the refinement of an action by an infinite behaviour. This contradicts a common assumption that an action takes only a finite amount of time. It means that when regarding a sequential composition $a; b$ we can not be sure that b occurs under all circumstances; it can only occur if the action a really terminates.

There is one type of refinement that we do not want to allow, namely to "forget" actions by replacing them with the empty process.

0.2 Example

Continuing Example 0.1 we could imagine that getting permission to send turns out to be unnecessary and can be skipped. Hence we replace the corresponding action by the empty behaviour, thus obtaining



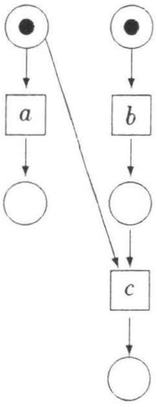
Forgetful refinement

Even though this operation seems natural when applied as in the above example, it may cause drastic changes in the possible behaviours of a system. It may happen that executing a certain action a prevents another action from happening. This property should be preserved under refinement of a. However, if a is completely removed, it cannot prevent anything any more, which can remove a deadlock possibility from the system. For this reason "forgetful" refinements will not be considered here.

0.3 Example

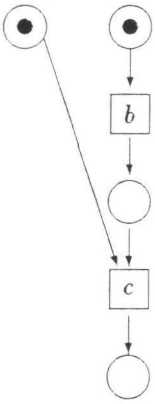
Consider the Petri net

$N =$



and the net obtained when refining a by the empty behaviour:

$N' =$



.

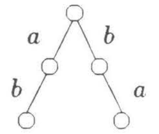
In the first net it is possible to execute a and b, and by this reach a state where no further action is possible. If we try to deduce the behaviour after refinement from the behaviour of N, we would expect that the refined system may reach a state, by executing b, where no more action is possible. However, this is not the case for N'. After b, it is always possible to execute c in N'.

In order to define a suitable refinement operator, one first has to select a model for the description of concurrent systems. The models of concurrency found in the literature can roughly be distinguished in two kinds: those in which the independent execution of two processes is modelled by specifying the possible interleavings of their (atomic) actions, and those in which the causal relations between the actions of a system are represented explicitly. The interleaving based models were devised to describe systems built from actions that are assumed to be instantaneous or indivisible. Nevertheless, one might be tempted to use them also for the description of systems built from actions that may have a duration or structure. However, the following example shows that it is not possible to define the desired compositional refinement operator on such models of concurrency without imposing some restrictions (as already observed in PRATT [108] and CASTELLANO, DE MICHELIS & POMELLO [36]).

0.4 Example

The systems $P = a \parallel b$, executing the actions a and b independently, and $Q = a; b + b; a$, executing either the sequence ab or the sequence ba, cannot be distinguished in interleaving models; they are represented by the same tree in the model of synchronisation trees (MILNER [92]).

tree (P) = tree (Q) =

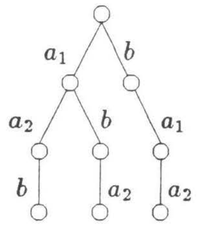


After refining a into the sequential compositon of a_1 and a_2, thereby obtaining the systems

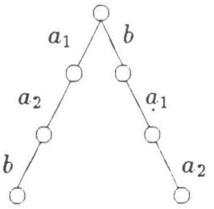
$$P' = (a_1; a_2) \parallel b \quad \text{and} \quad Q' = (a_1; a_2); b + b; (a_1; a_2)\,,$$

their tree representations are different:

tree (P') =



, tree (Q') =

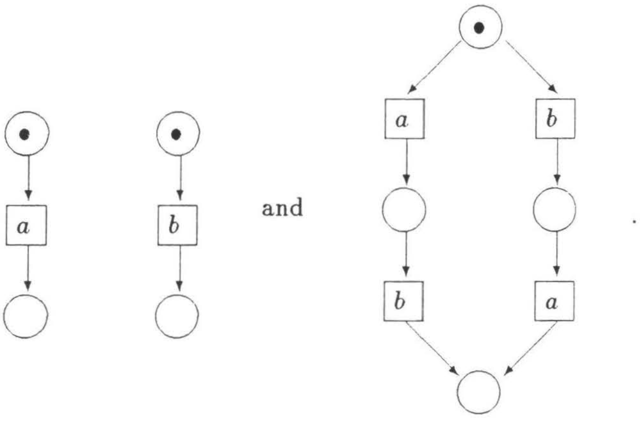


The two systems are even non–equivalent, according to any reasonable semantic equivalence, since only P' can perform the sequence of actions a_1ba_2. Hence, in the model of synchronisation trees the semantic representation of the refined systems is not derivable from the semantic representation of the original systems. The same holds for other interleaving models.

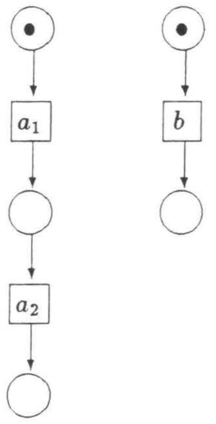
There are still ways left to define a compositional refinement operator on interleaving based models. First of all one could restrict the kind of refinements that are allowed in such a way that situations as in Example 0.4 cannot occur. Of course this would exclude the possibility of refining a in $a_1; a_2$ in either P or Q (or both). Although we consider this to be an interesting option, in this thesis we choose to allow rather general refinements, including at least the one of Example 0.4. Furthermore, some approaches have been proposed which are based on a concept of "atomic" actions; refining an atomic action would then result in an "atomic" process that cannot be "interupted" by other activities (the refinement of P in Example 0.4 would not have the execution a_1ba_2). We will comment on these approaches in the concluding section. In this work we choose not to assume action atomicity in any way, and to allow the parallel or independent execution of actions. Hence interleaving based models are unsuited for our approach. On the other hand we will show that the desired compositional refinement operator *can* be defined on causality based models of concurrency without imposing such restrictions. We will do this for semantic models like Petri nets and event structures. Since these models are being used as a semantics of languages like CCS, we hope that this will lead also to extending these languages by a mechanism for refinement.

0.5 Example

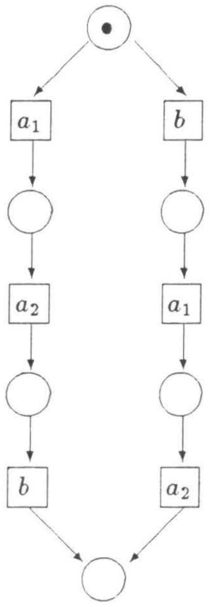
The systems $P = a \parallel b$ and $Q = a; b + b; a$ from Example 0.4 may be represented by the (labelled) Petri nets



The Petri net representations of the refined systems P' and Q', where a is replaced by the sequence $a_1 a_2$, are then derivable by transition refinement from the nets for the original systems. We obtain



and



.

We will use two kinds of semantic models. Both of them are based on the idea of PETRI [104] to model causalities in concurrent systems explicitly and thereby also representing independence of activities. Additionally, the models we use represent the choice structure of systems; they show where decisions between alternative behaviours are taken.

We will not distinguish external and internal actions here; we do not consider abstraction by hiding of actions.

The more basic model, in particular when being concerned more with actions than with states, are *event structures*. We will consider three types of event structures here: *prime event structures* with a binary conflict relation [100], *flow event structures*, which are particularly suited as a semantic model of CCS [32], and, as a more abstract and general model, *configuration structures* (*families of configurations* [128]), where a system is represented by its subsets of events which determine possible executions.

The models considered so far are usually not applied to model systems directly, but rather as the underlying semantics of system description languages like CCS. One of the reasons for this is that infinite behaviours can only be represented by infinite structures (with an infinite set of events). So, finally, we will consider Petri nets as a framework which is directly applicable in the design process. Event structures may be derived from Petri nets as a particularly simple case, but Petri nets are more powerful. For example, infinite behaviours may be represented as finite net structures together with the "token game". However, causality is then no longer a basic notion but has to be derived. Petri nets with their appealing graphical representation are being used extensively for the — more or less formal — representation of systems and — mostly less formal — during the design process. A disciplined way for developing net models systematically by refinement is therefore very important.

We start in Section 1 by presenting the basic notions for prime event structures and by showing how to refine actions by finite, conflict-free behaviours. We show that, for refining actions with more general behaviours, it is convenient to use more expressive models. In Section 2, we introduce flow event structures and show how to refine actions also by (possibly infinite) behaviours with conflicts. We show that, as for prime event structure refinement, the behaviour of a refined flow event structure may be deduced compositionally. In Section 3, we introduce configuration structures and a refinement operation for them. We show that the more "syntactic" constructions in the previous sections are consistent with this general notion. Finally, we give an overview on the work on refinement in Petri nets, and we suggest a rather general notion of refinement of transitions which is still modular with respect to behaviour. Related work is discussed in the concluding section.

1 Refinement of actions in prime event structures

In this section, we show how to refine actions in the most simple form of event structures, prime event structures with a binary conflict relation (NIELSEN, PLOTKIN & WINSKEL [100]). Furthermore, we motivate our move to more general structures in the next two sections because of the limitations of this approach.

We consider systems that are capable of performing actions from a given set *Act* of action names. We will frequently give CCSP-expressions for our examples, to make them easier to understand: + will denote choice (as in CCS), | will denote parallel composition (without communication), $a.P$ performs action a and then behaves like P and *nil* denotes the empty process; a abbreviates $a.nil$ and the unary prefixing operator binds stronger than the binary ones, as usual. Dots in expressions $a.P$ will be omitted. However, this notation is only used for intuition; formally our results are established for event structures.

1.1 Definition

A *(labelled) prime event structure (over an alphabet Act)* is a 4-tuple $\mathcal{E} = (E, \leq, \#, l)$ where

- E is a set of *events*,

- $\leq\, \subseteq E \times E$ is a partial order (the *causality relation*) satisfying the *principle of finite causes*:

$$\forall e \in E : \{d \in E | d \leq e\} \text{ is finite,}$$

- $\# \subseteq E \times E$ is an irreflexive, symmetric relation (the *conflict relation*) satisfying the *principle of conflict heredity*:

$$\forall d, e, f \in E : d \leq e \wedge d \# f \Rightarrow e \# f,$$

- $l : E \longrightarrow Act$ is a *labelling function*.

The components of a prime event structure $\mathcal{E}$ will be denoted by $E_{\mathcal{E}}, \leq_{\mathcal{E}}, \#_{\mathcal{E}}$ and $l_{\mathcal{E}}$. If clear from the context, the index $\mathcal{E}$ will be omitted. As usual, we write $d < e$ for $d \leq e \wedge d \neq e$, etc.

A prime event structure represents a concurrent system in the following way: action names $a \in Act$ represent actions the system might perform, an event $e \in E$ labelled with a represents an occurrence of a during a possible run of the system, $d < e$ means that d is a prerequisite for e and $d \# e$ means that d and e cannot happen both in the same run.

Causal independence *(concurrency)* of events is expressed by the derived relation $co \subseteq E \times E : d\ co\ e$ iff $\neg(d < e \vee e < d \vee d \# e)$. By definition, $<, >, \#$ and co form a partition of $E \times E$.

Throughout the paper, we assume a fixed set *Act* of action names as labelling set. Let $\mathbb{E}_{prime}$ denote the domain of prime event structures labelled over *Act*.

A prime event structure $\mathcal{E}$ is *finite* if $E_{\mathcal{E}}$ is finite; $\mathcal{E}$ is *conflict-free* if $\#_{\mathcal{E}} = \emptyset$. O denotes the empty event structure $(\emptyset, \emptyset, \emptyset, \emptyset)$.

For $X \subseteq E_{\mathcal{E}}$, the *restriction of* $\mathcal{E}$ *to* X is defined as

$$\mathcal{E}\lceil X = (X, \ \leq \cap (X \times X), \ \# \cap (X \times X), \ l\lceil X).$$

Two prime event structures $\mathcal{E}$ and $\mathcal{F}$ are *isomorphic* ($\mathcal{E} \cong \mathcal{F}$) iff there exists a bijection between their sets of events preserving $\leq, \#$ and labelling. Generally, we will not distinguish isomorphic event structures.

Isomorphism classes of conflict-free prime event structures are called *pomsets* (PRATT [108]). They have also been coinsidered under the name *partial words* in GRABOWSKI [65]. Pomsets generated by certain subsets of events may be considered as possible "executions" of the system represented by the event structure. The partial order between action occurrences then represents causal dependencies in the execution. Subsets of events representing executions (called *configurations*) have to be conflict-free; furthermore they must be left-closed with respect to $\leq$ (all prerequisites for any event occurring in the "execution" must also occur). It is assumed that in a finite period only finitely many actions are performed. We will consider only finite executions when describing the behaviour of systems. So, unlike WINSKEL [128], we require configurations to be finite. We will comment on this point in Section 3.

1.2 Definition

i. A subset $X \subseteq E$ of events in a prime event structure $\mathcal{E}$ is *left-closed* in $\mathcal{E}$ iff, for all $d, e \in E$, $e \in X \wedge d \leq e \Rightarrow d \in X$.
   X is *conflict-free in* $\mathcal{E}$ iff $\mathcal{E}\lceil X$ is conflict-free.

ii. A subset $X \subseteq E$ will be called a *(finite) configuration* of a prime event structure $\mathcal{E}$ iff X is finite, left-closed and conflict-free in $\mathcal{E}$. $Conf\,(\mathcal{E})$ denotes the set of all configurations of $\mathcal{E}$. A configuration $X \in Conf\,(\mathcal{E})$ is called *complete* iff $\forall d \in E : d \notin X \Rightarrow \exists e \in X$ with $d \# e$.

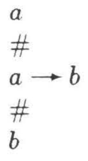
Configurations may be considered as possible states of the system; they determine the remaining behaviour of the system as being the set of all events which have not yet occurred and are not excluded because of conflicts. Note that a configuration X is complete iff it is maximal, i.e. $X \subseteq Y \in Conf\,(\mathcal{E})$ implies $X = Y$.

1.3 Example

Let us consider the event structure $\mathcal{E}$ corresponding to the expression $a|b + ab$.

In graphical representations, only immediate conflicts - not the inherited conflicts - are indicated. The $\leq$-relation is represented by arcs, omitting those derivable by transitivity. Furthermore, instead of events only their labels are displayed; if a label occurs twice it represents two different events. Thus these pictures determine event structures only up to isomorphism.

Following these conventions, $\mathcal{E}$ is represented as



.

The possible executions of $\mathcal{E}$ are represented by the pomsets

$$\emptyset,\ a,\ b,\ \begin{smallmatrix} a \\ b \end{smallmatrix} \text{ and } a \longrightarrow b\,.$$

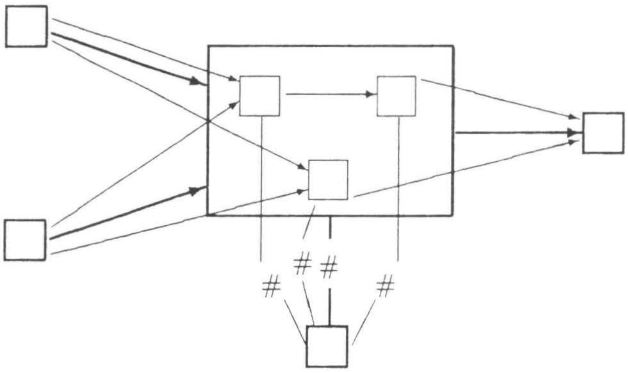
$\begin{smallmatrix} a \\ b \end{smallmatrix}$ and $a \longrightarrow b$ correspond to complete configurations.

We will now define a refinement operation substituting actions by finite, conflict-free, non-empty event structures. As discussed in the introduction, we will not allow forgetful refinements replacing actions by the empty event structure. We will later explain why we have to restrict to finite and conflict-free refinements of actions.

A refinement function will be a function *ref* specifiying, for each action a, an event structure $ref\,(a)$ which is to be substituted for a. Interesting refinements (and also the refinements in our examples) will mostly refine only certain actions, hence replace most actions by themselves. However, for uniformity (and for simplicity in proofs) we consider all actions to be refined.

Given an event structure $\mathcal{E}$ and a refinement function *ref*, we construct the refined event structure $ref\,(\mathcal{E})$ as follows. Each event e labelled by a is replaced by a disjoint copy, $\mathcal{E}_e$, of $ref\,(a)$. The causality and conflict structure is inherited from $\mathcal{E}$: every event which was causally before e will be causally before all events of $\mathcal{E}_e$, all events which causally followed e will causally follow all the events of $\mathcal{E}_e$, and all events in conflict with e will be in conflict with all the events of $\mathcal{E}_e$.

Graphically, the idea may be sketched as follows (in this picture we omit arcs derivable by transitivity and inherited conflicts).



1.4 Definition

(i) A function $ref: Act \rightarrow \mathbb{E}_{prime} - \{O\}$ is called a *refinement function (for prime event structures)* if $\forall a \in Act$: $ref(a)$ is finite and conflict-free.

(ii) Let $\mathcal{E} \in \mathbb{E}_{prime}$ and let *ref* be a refinement function.
Then $ref(\mathcal{E})$ is the prime event structure defined by
- $E_{ref(\mathcal{E})} = \{(e, e') | e \in E_{\mathcal{E}}, e' \in E_{ref(l_{\mathcal{E}}(e))}\}$,
- $(d, d') \leq_{ref(\mathcal{E})} (e, e')$ iff $d <_{\mathcal{E}} e$ or $(d = e \wedge d' \leq_{ref(l_{\mathcal{E}}(d))} e')$,
- $(d, d') \#_{ref(\mathcal{E})} (e, e')$ iff $d \#_{\mathcal{E}} e$,
- $l_{ref(\mathcal{E})}(e, e') = l_{ref(l_{\mathcal{E}}(e))}(e')$.

We show that refinement is a well-defined operation on prime event structures, even when isomorphic prime event structures are identified.

1.5 Proposition

(i) If $\mathcal{E} \in \mathbb{E}_{prime}$ and *ref* is a refinement function then $ref(\mathcal{E})$ is a prime event structure indeed.

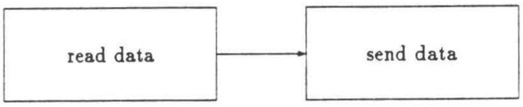
(ii) If $\mathcal{E} \in \mathbb{E}_{prime}$ and *ref*, *ref'* are refinement functions with $ref(a) \cong ref'(a)$ for all $a \in Act$ then $ref(\mathcal{E}) \cong ref'(\mathcal{E})$.

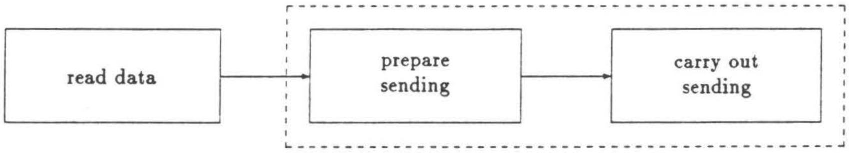
(iii) If $\mathcal{E}, \mathcal{F} \in \mathbb{E}_{prime}$, *ref* is a refinement function and $\mathcal{E} \cong \mathcal{F}$ then $ref(\mathcal{E}) \cong ref(\mathcal{F})$.

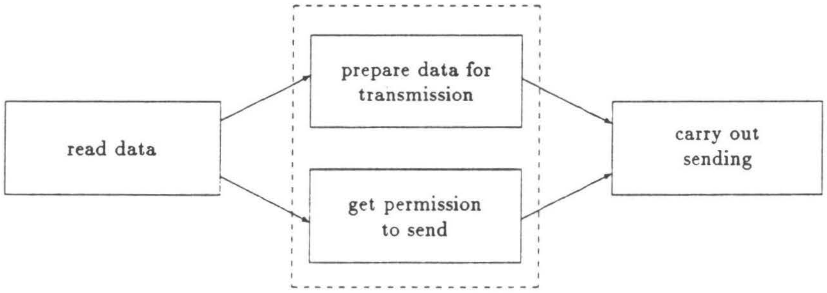
Proof Straightforward. ■

1.6 Example

We consider a simplified version of the sender (Example 0.1) from the introduction. We assume that the sender reads and sends only once. We may carry out the first two steps of the design in terms of prime event structures as follows.







> The next refinement step would require a refinement of an action by conflicting behaviours. This is not possible in our framework up to now.

The reason that we can only refine actions by conflict–free event structures is the axiom of conflict heredity and the notion of configuration in prime event structures. They imply that any event will always occur with a unique history (in terms of its causal predecessors) [128].

Now consider e.g. $\mathcal{E} = \overset{a}{\underset{b}{\downarrow}}$. Replacing a by $c \# d$ would require to duplicate the event labelled by b in some way, since b should then occur either caused by c or by d. Since this would lead to a complicated definition, we will consider more general forms of event structures that do not require duplication in Section 2 and 3.

The restriction to refinement of actions by finite event structures is necessary to ensure that the resulting event structure will obey the axiom of finite causes. In the more general models we will consider later, we will not assume this axiom, and this will allow also refinements by infinite behaviours as discussed in the introduction.

Finally, we show how the behaviour of the refined event structure $ref(\mathcal{E})$ is determined by the behaviour of $\mathcal{E}$ and by the behaviour of the event structures which are substituted for actions.

1.7 Proposition

Let $\mathcal{E} \in \mathbb{E}_{prime}$, let ref be a refinement function.

We call $\widetilde{X}$ a *refinement of configuration* $X \in Conf(\mathcal{E})$ *by ref* iff

- $\widetilde{X} = \bigcup_{e \in X} \{e\} \times X_e$ where $\forall e \in X : X_e \in Conf(ref(l_{\mathcal{E}}(e))) - \{\emptyset\}$,
- $e \in busy(\widetilde{X}) \Longrightarrow e$ maximal in X with respect to $\leq_{\mathcal{E}}$ where $busy(\widetilde{X}) := \{e \in X \mid X_e \text{ not complete}\}$.

Then $Conf(ref(\mathcal{E})) = \{\widetilde{X} \mid \widetilde{X}$ is a refinement of a configuration $X \in Conf(\mathcal{E})\}$.

Proof [54] or as a special case of Proposition 2.8. ■

Hence the configurations of $ref(\mathcal{E})$ are exactly those configurations which are refinements of configurations of $\mathcal{E}$. A refinement of a configuration X of $\mathcal{E}$ is obtained by replacing each event e in X by a non-empty configuration X_e of $ref(l_{\mathcal{E}}(e))$. Events which are causally necessary for other events in X may only be replaced by complete configurations.

2 Refinement of actions in flow event structures

In the previous section, we have indicated that for refining actions by event structures with conflicts more general models than prime event structures are appropriate. In Boudol & Castellani [32] a form of event structures, called *flow event structures*, is suggested which is particularly suited for giving semantics to languages like CCS. Flow event structures are more general than prime events in the following sense: they do not assume conflict heredity and the axiom of finite causes, they allow inconsistent (self-conflicting) events and the causality relation is not required to be transitive and may even contain (syntactic) cycles. This makes it very easy to define operations like parallel composition and restriction, and we will show here that they are also well suited to deal with refinement of actions.

2.1 Definition

A *(labelled) flow event structure (over an alphabet Act)* is a 4-tuple $\mathcal{E} = (E, \prec, \#, l)$ where

- E is a set of *events*,
- $\prec \subseteq E \times E$ is an irreflexive relation, the *flow relation*,
- $\# \subseteq E \times E$ is a symmetric relation, the *conflict relation*,
- $l : E \longrightarrow Act$ is the *labelling function*.

Let $\mathbb{E}$ denote the domain of flow event structures labelled over *Act*. The components of $\mathcal{E} \in \mathbb{E}$ will be denoted by $E_{\mathcal{E}}, \prec_{\mathcal{E}}, \#_{\mathcal{E}}$ and $l_{\mathcal{E}}$. The index $\mathcal{E}$ will be omitted if clear from the context. $\mathcal{E}$ is *conflict-free* if $\#_{\mathcal{E}} = \emptyset$. For $X \subseteq E_{\mathcal{E}}$, $\mathcal{E}\lceil X = (X, \prec_{\mathcal{E}} \lceil X, \#_{\mathcal{E}}\lceil X, l_{\mathcal{E}}\lceil X)$ is the *restriction of* $\mathcal{E}$ *to* X.

Two flow event structures $\mathcal{E}$ and $\mathcal{F}$ are *isomorphic* ($\mathcal{E} \cong \mathcal{F}$) iff there exists a bijection between their sets of events preserving $\prec$, $\#$ and labelling.

The interpretation of the conflict and the flow relation is formalised by defining configurations of flow event structures. Configurations must be conflict free; in particular, self-conflicting events will never occur in any configuration. $d \prec e$ will mean that d is a *possible immediate cause* for e. For an event to occur it is necessary that a *complete* non–conflicting set of its causes has occurred. Here a set of causes is complete if for any cause which is not contained there is a conflicting event which is contained. Finally, no cycles with respect to causal dependence may occur.

2.2 Definition Let $\mathcal{E} \in \mathbb{E}$.

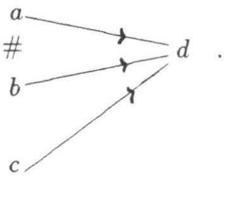
(i) $X \subseteq E$ is *left–closed in* $\mathcal{E}$ *up to conflicts* iff $\forall d, e \in E$: if $e \in X, d \prec e$ and $d \notin X$ then there exists an $f \in X$ with $f \prec e$ and $d\#f$.
$X \subseteq E$ is *conflict-free* iff $\mathcal{E} \lceil X$ is conflict–free.

(ii) $X \subseteq E$ is a *(finite) configuration* of $\mathcal{E}$ iff X is finite, left–closed up to conflicts and conflict–free and does not contain a causality cycle: $\leq_X := (\prec \cap (X \times X))^*$ is an ordering. A configuration X is called *maximal* iff $X \subseteq Y \in Conf(\mathcal{E})$ implies $X = Y$. A configuration X is called *complete* iff $\forall d \in E : d \notin X \Rightarrow \exists e \in X$ with $d\#e$. $Conf(\mathcal{E})$ denotes the set of all configurations of $\mathcal{E}$.

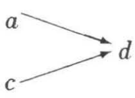
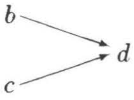
The causal dependence between action occurrences in a configuration may again, as for prime event structures, be represented by a pomset; for $X \in Conf(\mathcal{E})$, we take the isomorphism class of $(X, \leq_X, l_{\mathcal{E}}\lceil X)$.

2.3 Example

The system $((a+b) \; || \; c); d$ may be represented by the flow event structure



(in graphical representations we omit names of events and represent $\prec$ by arcs of the form —→—).

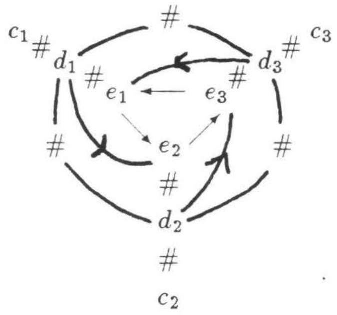
The pomsets  and  correspond to complete configurations.

Note that prime event structures are special flow event structures defining $d \prec e$ iff $d < e$; the definition of configuration then coincides.

However, in contrast to prime event structures, not all *maximal* configurations are complete. Partly this is due to the fact that, in flow event structures, syntactic and semantic conflict not necessarily coincide, (two events are in *semantic conflict* if there is no configuration containing them both). Flow event structures where syntactic and semantic conflict coincide are called *faithful* in Boudol [30]. However, also in faithful flow event structures maximal configurations are not necessarly complete, either due to inconsistent events, but also in flow event structures without inconsistent events, as shown by the following example.

2.4 Example

Let $\mathcal{E}$ =



The configuration $\{c_1, c_2, c_3\}$ is maximal but not complete.

Maximal but incomplete configurations may be interpreted as deadlocking behaviours. Assume that a semantic sequential composition is defined for flow event structures by putting all events in the first component in $\prec$–relation with the events of the second component. Any incomplete maximal configuration of the first component would then disable the second component. Thus, in flow event structures, deadlock and termination may be distinguished.

2.5 Definition

A flow event structure $\mathcal{E}$ is *deadlock-free* iff every maximal configuration of $\mathcal{E}$ is complete.

Refinement of actions in flow event structures may now be defined as follows. We assume a refinement function *ref*: $Act \longrightarrow \mathbb{E} - \{O\}$ (where O denotes the empty flow event structure) and replace each event labelled by a by a disjoint copy of *ref* (a). The conflict and causality structure will just be inherited.

Hence, we may replace actions also by behaviours with conflicts and by infinite behaviours.

2.6 Definition

(i) A function *ref* : $Act \longrightarrow \mathbb{E} - \{O\}$ is called a *refinement function (for flow event structures).*

(ii) Let $\mathcal{E} \in \mathbb{E}$ and let *ref* be a refinement function.
Then the *refinement of $\mathcal{E}$ by ref*, *ref* $(\mathcal{E})$, is the flow event structure defined by
- $E_{ref(\mathcal{E})} = \{(e,e') | e \in E_{\mathcal{E}}, e' \in E_{ref(l_{\mathcal{E}}(e))}\}$,
- $(d,d') \prec_{ref(\mathcal{E})} (e,e')$ iff $d \prec e$ or $(d = e \wedge d' \prec_{ref(l_{\mathcal{E}}(d))} e')$,
- $(d,d') \#_{ref(\mathcal{E})} (e,e')$ iff $d \#_{\mathcal{E}} e$ or $(d = e \wedge d' \#_{ref(l_{\mathcal{E}}(d))} e')$,
- $l_{ref(\mathcal{E})}(e,e') = l_{ref(l_{\mathcal{E}}(e))}(e')$.

As for prime event structures, we verify that *ref* $(\mathcal{E})$ is well–defined, even when isomorphic flow event structures are identified.

2.7 Proposition

(i) If $\mathcal{E} \in \mathbb{E}$ and *ref* is a refinement function then *ref* $(\mathcal{E})$ is a flow event structure indeed.

(ii) If $\mathcal{E} \in \mathbb{E}$ and *ref*, *ref* $'$ are refinement functions with *ref* $(a) \cong$ *ref* $'(a)$ for all $a \in Act$ then *ref* $(\mathcal{E}) \cong$ *ref* $'(\mathcal{E})$.

(iii) If $\mathcal{E}, \mathcal{F} \in \mathbb{E}$, *ref* is a refinement function and $\mathcal{E} \cong \mathcal{F}$ then *ref* $(\mathcal{E}) \cong$*ref* $(\mathcal{F})$.

Proof Straightforward. ■

Finally, we show that, analogously to prime event structures, the behaviour of a refined flow event structure *ref* $(\mathcal{E})$ may be deduced compositionally from the behaviour of $\mathcal{E}$ and the behaviour of the refinements of actions.

2.8 Proposition

Let $\mathcal{E} \in \mathbb{E}$, let *ref* be a refinement function for flow event structures.

We call $\tilde{X}$ a *refinement of configuration $X \in Conf\ (\mathcal{E})$ by ref* iff

- $\widetilde{X} = \bigcup_{e \in X} \{e\} \times X_e$ where $\forall e \in X : X_e \in \mathit{Conf}\,(\mathit{ref}\,(l_{\mathcal{E}}(e))) - \{\emptyset\}$,

- $e \in \mathit{busy}(\ \widetilde{X}\ ) \Longrightarrow e$ maximal in X with respect to $\leq_X$ where $\mathit{busy}\ (\ \widetilde{X}\ ) := \{e \in X \mid X_e \text{ not complete}\}$.

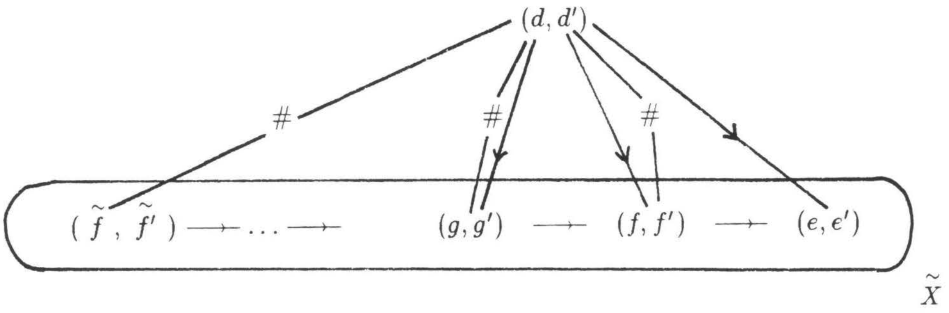
Then $\mathit{Conf}\,(\mathit{ref}\,(\mathcal{E})) = \{\ \widetilde{X} \mid \widetilde{X}$ is a refinement of a configuration $X \in \mathit{Conf}\,(\mathcal{E})\}$.

Proof

"$\subseteq$" Let $\widetilde{X} \in \mathit{Conf}\,(\mathit{ref}\,(\mathcal{E}))$.

First we show that $X := pr_1(\ \widetilde{X}\ ) \in \mathit{Conf}\,(\mathcal{E})$.
X is finite since $\widetilde{X}$ is finite.

X is left–closed in $\mathcal{E}$ up to conflicts:
Let $e \in X, d \in E_{\mathcal{E}}$ with $d \prec_{\mathcal{E}} e$ and $d \notin X$.
We have to show that there exists an $f \in X$ with $f \prec_{\mathcal{E}} e$ and $f \#_{\mathcal{E}} d$.
Since $e \in X$ there must be some $(e, e') \in \widetilde{X}$.
There exists $(d, d') \in E_{\mathit{ref}(\mathcal{E})}, (d, d') \notin \widetilde{X}$ since $\mathit{ref}\,(d) \neq O$ and $d \notin X$.
Furthermore $(d, d') \prec_{\mathit{ref}(\mathcal{E})} (e, e')$ since $d \prec_{\mathcal{E}} e$.
So $\exists (f, f') \in \widetilde{X}$ with $(f, f') \prec_{\mathit{ref}(\mathcal{E})} (e, e')$ and $(f, f') \#_{\mathit{ref}(\mathcal{E})} (d, d')$.
$f \neq d$ since $f \in X, d \notin X \Longrightarrow f \#_{\mathcal{E}} d$.
If $f \neq e$ we have $f \prec_{\mathcal{E}} e$ and we are done.
Assume $f = e$ then $(d, d') \prec_{\mathit{ref}(\mathcal{E})} (f, f')$.
Then $\exists (g, g') \in \widetilde{X}$ with $(g, g') \prec_{\mathit{ref}(\mathcal{E})} (f, f') = (e, f')$ and $(g, g') \#_{\mathit{ref}(\mathcal{E})} (d, d')$.
$g \#_{\mathcal{E}} d$ since $g \neq d$. Furthermore $g \in X$.
If $g \neq f = e$ then $g \prec_{\mathcal{E}} e$ and we are done. Since $\widetilde{X}$ is finite, we will find (by repeating this), after finitely many steps, $(\ \widetilde{f}\ ,\ \widetilde{f}'\ ) \in \widetilde{X}$ with $\widetilde{f} \#_{\mathcal{E}} d$ and $\widetilde{f} \prec_{\mathcal{E}} e$. Hence X is left–closed up to conflicts.



X is conflict–free:
Assume $d, e \in X$ with $d \#_{\mathcal{E}} e$.
Then there exist $(d, d'), (e, e') \in \tilde{X}$, $(d, d') \#_{ref(\mathcal{E})} (e, e')$.
This is a contradiction since $\tilde{X}$ is conflict–free.

Finally we have to show that X does not contain a causality–cycle. Assume $d, e \in X, d \neq e, d \leq_X e$ and $e \leq_X d$ (where $\leq_X$ is derived from $\prec_{\mathcal{E}}$). It is straightforward to verify that this implies $\exists (d, d'), (e, e') \in \tilde{X}$ with $(d, d') \neq (e, e')$, $(d, d') \leq_{\tilde{X}} (e, e')$ and $(e, e') \leq_{\tilde{X}} (d, d')$. This is in contradiction with the cyclefreeness of $\tilde{X}$.

Hence $X = pr_1(\tilde{X}) \in Conf\,(\mathcal{E})$.
We will show that $\tilde{X}$ is a refinement of X.
Let $e \in X$ and $X_e := \{e' \mid (e, e') \in \tilde{X} \}$. By construction $X_e \neq \emptyset$.
Let $\mathcal{E}_e := ref\,(l_{\mathcal{E}}(e))$. We want to show that $X_e \in Conf\,(\mathcal{E}_e)$.

Obviously $X_e \subseteq E_{\mathcal{E}_e}$.
X_e is finite, conflict–free and cycle–free since $\tilde{X}$ is finite, conflict–free and cycle-free. So it only remains to be shown that X_e is left–closed up to conflicts.

Let $d' \in \mathcal{E}_e, d' \prec_{\mathcal{E}_e} e' \in X_e, d' \notin X_e$.
Then $(e, d') \in E_{ref(\mathcal{E})}, (e, d') \prec_{ref(\mathcal{E})} (e, e') \in \tilde{X}$ and $(e, d') \notin \tilde{X}$.
So there exists $(f, f') \in \tilde{X}$ with $(f, f') \prec_{ref(\mathcal{E})} (e, e')$ and $(f, f') \#_{ref(\mathcal{E})} (e, d')$.
$f, e \in X \Longrightarrow \neg(f \#_{\mathcal{E}} e) \Longrightarrow f = e \wedge f' \#_{\mathcal{E}_e} d' \Longrightarrow f' \in X_e$ and $f' \prec_{\mathcal{E}_e} e'$.
Hence $X_e \in Conf\,(\mathcal{E}_e)$.

From what we have shown by now it follows that $\tilde{X} = \bigcup_{e \in X} \{e\} \times X_e$ with $X \in Conf\,(\mathcal{E})$ and, for all $e \in X$, $X_e \in Conf\,(ref\,(l_{\mathcal{E}}(e))) - \{\emptyset\}$.

Now let $e \in busy\ (\tilde{X})$. We have to show that e is maximal in X whith respect to $\leq_X$.

Suppose e is not maximal in X.
Then there exists $f \in X$ with $e \prec_{\mathcal{E}} f$, and there exists $(f, f') \in \tilde{X}$.
Since X_e is not complete there exists $d' \in E_{\mathcal{E}_e} - X_e$ with

$$(*) \forall e' \in X_e : \neg(d' \#_{\mathcal{E}_e} e').$$

We have $(e, d') \prec_{ref\,(\mathcal{E})} (f, f')$, $(e, d') \notin \tilde{X}$.
Since $\tilde{X}$ is a configuration, there then exists $(g, g') \in \tilde{X}$ with
$(g, g') \prec_{ref\,(\mathcal{E})} (f, f')$ and $(g, g') \#_{ref\,(\mathcal{E})} (e, d')$.

Since $g, e \in X$, we have $\neg(g \#_{\mathcal{E}} e)$.
Hence $g = e$ and $g' \in X_e$, $g' \#_{\mathcal{E}_e} d'$.
However this contradicts $(*)$.

"$\supseteq$" Let $\tilde{X}$ be a refinement of $X \in \mathit{Conf}(\mathcal{E})$.
We show that $\tilde{X} \in \mathit{Conf}(\mathit{ref}(\mathcal{E}))$.

It follows in a straightforward manner from the corresponding properties of X and the X_e's that $\tilde{X}$ is finite and conflict-free and contains no causality cycles. Hence it suffices to show that $\tilde{X}$ is left-closed up to conflicts.

So let $(e, e') \in \tilde{X}$, let $(d, d') \in E_{\mathit{ref}(\mathcal{E})} - \tilde{X}$ with $(d, d') \prec_{\mathit{ref}(\mathcal{E})} (e, e')$. We have to show that there exists $(f, f') \in \tilde{X}$ with $(f, f') \prec_{\mathit{ref}(\mathcal{E})} (e, e')$ and $(f, f') \#_{\mathit{ref}(\mathcal{E})} (d, d')$.

First assume $d = e$. Then this follows immediately from the corresponding property of X_e.

Now let $d \neq e$.
If $d \notin X$ then the requirement follows from the corresponding property of X.
So we now consider the remaining case that $d \neq e$ and $d \in X$. Then $d' \notin X_d$.
Since $d \neq e$ we have $d \prec_{\mathcal{E}} e$, hence d is not maximal in X.
Then X_d must be complete.
So $d' \notin X_d$ implies $\exists f' \in X_d$ with $f' \#_{\mathit{ref}(l_{\mathcal{E}}(d))} d'$.
Hence $(d, f') \in \tilde{X}$, $(d, f') \prec_{\mathit{ref}(\mathcal{E})} (e, e')$ and $(d, f') \#_{\mathit{ref}(\mathcal{E})} (d, d')$. ■

We end this section with a lemma that will be useful later on.

2.9 Lemma Let $\mathcal{E} \in \mathbb{E}$, $X \in \mathit{Conf}(\mathcal{E})$ and $\mathit{busy} \subseteq X$.

$$\text{Then } \forall e \in \mathit{busy} : e \text{ maximal in } X \text{ with respect to } \leq_X$$
$$\iff \forall Y \subseteq \mathit{busy} : X - Y \in \mathit{Conf}(\mathcal{E}).$$

Proof
" $\Longrightarrow$ " Let $\mathcal{E} \in \mathbb{E}$, $X \in \mathit{Conf}(\mathcal{E})$, $Y \subseteq X$ and $\forall e \in Y : e$ maximal in X w.r.t. $\leq_X$. It suffices to prove that $X - Y \in \mathit{Conf}(\mathcal{E})$. $X - Y$ is finite and conflict-free and does not contain causality cycles since X has these properties. It remains to be shown that $X - Y$ is left-closed up to conflicts.
Suppose $e \in X - Y$, $d \prec_{\mathcal{E}} e$ and $d \notin X - Y$. If $d \in Y$ then d would be maximal in X w.r.t. $\leq_X$, contradicting $d \prec_{\mathcal{E}} e$. Thus $d \notin X$. Hence there is an $f \in X$ with $f \prec_{\mathcal{E}} e$ and $d \#_{\mathcal{E}} f$. Since $f \prec_{\mathcal{E}} e$, f is not maximal in X w.r.t. $\leq_X$, so $f \in X - Y$, which had to be proven.

" $\Longleftarrow$ " Let $\mathcal{E} \in \mathbb{E}$, $X \in Conf\,(\mathcal{E})$, $d \in X$ and $X - \{d\} \in Conf\,(e)$. It suffices to proof that d is maximal w.r.t. $\leq_X$.
Suppose it is not, then $\exists e \in X$ with $d \prec_{\mathcal{E}} e$. Since $X - \{d\} \in Conf\,(\mathcal{E})$, there exists an $f \in X - \{d\}$ with $f \prec_{\mathcal{E}} e$ and $d \#_{\mathcal{E}} f$, contradicting the conflict-freeness of X. ■

This means that, in Proposition 2.8, the condition "$e \in busy(\ \tilde{X}\ ) \Longrightarrow e$ maximal in X w.r.t. $\leq_X$" can be replaced by "for all $Y \subseteq busy(\ \tilde{X}\ ),\ \ X - Y \in Conf\,(\mathcal{E})$".

3 Configuration structures and refinement of actions

In the previous section we have shown that flow event structures may be used for refinement of actions, even when substituting actions by behaviours with conflicts or by infinite behaviours. However, the refinement operation we have defined depends on the particular "syntax" of flow event structures. In this section, our aim is to define a refinement operation for a very general model of concurrent systems, such that refinement operations for particular representations, as flow event structures, are obtained as a special case.

We will consider a model where a system is represented by its set of configurations. As in the previous sections, occurrences of actions are represented by *events* labelled by the corresponding action names. A *configuration* is a set of events representing a state of the system where exactly its elements have happened. We only consider finite configurations here. Following ideas of Winskel [128] we represent a system by a family of configurations satisfying certain consistency requirements.

3.1 Definition

A *(labelled) configuration structure (over an alphabet Act)* is a pair $\mathcal{C} = (C, l)$ where C is a family of finite sets (*configurations*) such that
- $\emptyset \in C$,
- $X, Y, Z \in C,\ X \cup Y \subseteq Z \Longrightarrow X \cup Y \in C$,
- $X \in C \wedge d, e \in X, d \neq e \Longrightarrow \exists Y \in C$ with $Y \subseteq X$ and $(d \in Y \Longleftrightarrow e \notin Y)$,

and $l : \bigcup_{X \in C} X \rightarrow Act$ is a *labelling function.*

The requirements for a family of sets of events to form a configuration structure may be explained as follows. The initial state of a system is the state where no action has been performed yet. Hence $\emptyset$ is always a configuration.

Now, if two configurations X, Y are contained in a third configuration Z then $X \cup Y$ is consistent or *conflict-free*; e.g. all its elements can happen together in one run. Since both X and Y represent already possible runs, it should then also be possible to execute just the events in X and Y, hence $X \cup Y$ should be a configuration. If we consider two distinct events occurring in some run, then there must be an intermediate state where already one of them has occurred whereas the other has not yet occurred (coincidence can not be enforced). This is guaranteed by the third requirement.

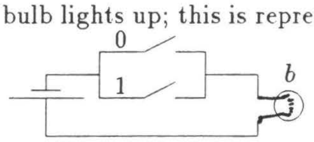
Finally, a remark on our requirement that configurations should be finite. As usual, we assume that in a finite period only finitely many actions may be performed. Now the requirement says that we only consider states that are reachable in a finite period of time. WINSKEL [128] allows configurations to be infinite, thus representing also those states which can be reached in an infinite period of time. However, his infinite configurations are completely determined by the finite ones. Hence configuration structures as defined here are equally expressive as Winskel's families of configurations.

Convention We will denote the components of a configuration structure $\mathcal{C}$ by $C_\mathcal{C}$ and $l_\mathcal{C}$ respectively. By abuse of language, $C_\mathcal{C}$ will also be denoted by $\mathcal{C}$. Furthermore the set $E_\mathcal{C}$ of events of $\mathcal{C}$ is defined by $E_\mathcal{C} = \bigcup_{X \in \mathcal{C}} X$.

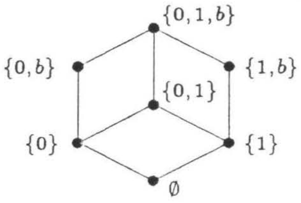
Let $\mathbb{C}$ denote the domain of configuration structures labelled over *Act*.

3.2 Example

We consider the example refered to as a "parallel switch" in [128].
We have two actions 0 and 1 interpreted as closing switch 0 and closing switch 1, respectively, in an electric circuit. As soon as at least one of the switches is closed, a bulb lights up; this is represented as an action b.



This may be represented by the following configuration structure (with a unique correspondence between actions and events):



The b-event may occur here without a unique "causal history"; in the configuration $\{0, 1, b\}$ it is not clear whether b is caused by 0 or by 1.

Usually, the names of events are not important; hence we will not distinguish configuration structures which are isomophic in the sense that they only differ with respect to names of events.

3.3 Definition

A configuration structure isomorphism between two configuration structures $\mathcal{C}, \mathcal{D} \in \mathbb{C}$ is a bijective mapping $f : E_{\mathcal{C}} \longrightarrow E_{\mathcal{D}}$ such that
- $X \in \mathcal{C} \Longleftrightarrow f(X) \in \mathcal{D}$ for $X \subseteq E_{\mathcal{C}}$,
- and $l_{\mathcal{D}}(f(e)) = l_{\mathcal{C}}(e)$ for $e \in E_{\mathcal{C}}$.

$\mathcal{C}$ and $\mathcal{D}$ are *isomorphic* — notation $\mathcal{C} \cong \mathcal{D}$ — if there exists a configuration structure isomorphism between them.

In configuration structures, completeness and maximality of configurations coincide. Deadlock and termination may not be distinguished.

3.4 Definition

A configuration X of a configuration structure $\mathcal{C}$ is called *complete* iff there is no $Y \neq X$ in $\mathcal{C}$ containing X.

We may now associate a configuration structure with each flow event structure (and via this also with each prime event structure).

3.5 Definition Let $\mathcal{E} \in \mathbb{E}$.

The configuration structure of $\mathcal{E}$, $\mathcal{C}(\mathcal{E})$, is defined as

$$\mathcal{C}(\mathcal{E}) = (Conf\,(\mathcal{E}),\ l_{\mathcal{E}} \lceil \bigcup_{X \in Conf(\mathcal{E})} X).$$

There is no unique corresponence in general: different flow event structures may have the same configuration structure (but not vice versa). In particular, the distiction between deadlock and termination is lost.

Next, we define refinement of actions for configuration structures. A refinement will be specified by a function *ref* specifying for each action a a configuration structure $ref\,(a)$ which is to be substituted for a. Again we only consider non–forgetful refinements here, hence $ref\,(a) \neq O$ for all $a \in Act$ where O denotes the empty configuration structure with $C_O = \{\emptyset\}$. Apart from this restriction, we may replace an action by any configuration structure.

3.6 Definition

(i) A function $ref : Act \longrightarrow \mathbb{C} - \{O\}$ is called a *refinement function (for configuration structures)*.

(ii) Let $\mathcal{C}$ be a configuration structure and let *ref* be a refinement function. We call $\widetilde{X}$ a *refinement of a configuration* $X \in \mathcal{C}$ *by ref* iff

- $\widetilde{X} = \bigcup_{e \in X} \{e\} \times X_e$ where $\forall e \in X : X_e \in ref(l_{\mathcal{C}}(e)) - \{\emptyset\}$,
- for all $Y \subseteq busy(\widetilde{X})$, $X - Y \in \mathcal{C}$,
  where $busy(\widetilde{X}) = \{e \in X | X_e \text{ not complete}\}$.

The *refinement of* $\mathcal{C}$ *by ref* is defined as $ref(\mathcal{C}) = (C_{ref(\mathcal{C})}, l_{ref(\mathcal{C})})$ with

$$C_{ref(\mathcal{C})} := \{ \widetilde{X} \mid \widetilde{X} \text{ is a refinement of some } X \in \mathcal{C} \text{ by } ref\}$$

and

$$l_{ref(\mathcal{C})}(e, e') = l_{ref(l_{\mathcal{C}}(e))}(e') \text{ for all } (e, e') \in \bigcup_{\widetilde{X} \in C_{ref(\mathcal{C})}} \widetilde{X}.$$

Intuitively, this definition may be explained as follows.

The configuration structure *ref* $(\mathcal{C})$ is obtained by taking all possible refinements of configurations of $\mathcal{C}$. A refinement of a configuration X of $\mathcal{C}$ is obtained by replacing each event e in X by a non–empty configuration X_e of $ref(l_{\mathcal{C}}(e))$. Events which are causally necessary for other events in X may only be replaced by complete configurations, hence it must be possible to take any subset of "uncompleted" or busy events out of X, again obtaining a configuration.

Next we show that refinement is a well–defined operation on configuration structures, even when isomorphic configuration structures are identified.

3.7 Proposition

(i) If $\mathcal{C} \in \mathbb{C}$ and *ref* is a refinement function then also $ref(\mathcal{C})$ is a configuration structure.

(ii) If $\mathcal{C} \in \mathbb{C}$ and *ref, ref'* are refinement functions with $ref(a) \cong ref'(a)$ for all $a \in Act$ then $ref(\mathcal{C}) \cong ref'(\mathcal{C})$.

(iii) If $\mathcal{C}, \mathcal{D} \in \mathbb{C}$, *ref* is a refinement function and $\mathcal{C} \cong \mathcal{D}$ then $ref(\mathcal{C}) \cong ref$ $(\mathcal{D})$.

Proof (i) cumbersome and omitted here, (ii) and (iii) straightforward. ■

Finally, we want to show that the easier syntactic refinement operation for flow event structures defined in section 2 is consistent with the refinement operation for configuration structures. However, since the distinction between deadlock and termination is lost in configuration structures, this is only true for deadlock-free refinements.

3.8 Theorem

Let $\mathcal{E} \in \mathbb{E}$, let *ref* be a refinement function for flow event structures with $\forall a \in Act : ref(a)$ deadlock-free.

Then $\mathcal{C}(ref(\mathcal{E})) = ref'(\mathcal{C}(\mathcal{E}))$
where $ref'(a) = \mathcal{C}(ref(a))$ for all $a \in Act$.

Proof

It has to be shown that $C_{\mathcal{C}(ref(\mathcal{E}))} = C_{ref'(\mathcal{C}(\mathcal{E}))}$ and $l_{\mathcal{C}(ref(\mathcal{E}))} = l_{ref'(\mathcal{C}(\mathcal{E}))}$. The first requirement translates to

$$Conf(ref(\mathcal{E})) = \{\ \tilde{X} \mid \tilde{X} \text{ is refinement of some } X \in Conf(\mathcal{E}) \text{ by } ref'\}.$$

From Proposition 2.8 we know

$$Conf(ref(\mathcal{E})) = \{\ \tilde{X} \mid \tilde{X} \text{ is a refinement of some } X \in Conf(\mathcal{E}) \text{ by } ref\}.$$

So it suffices to establish that a refinement by *ref'* is the same as a refinement by *ref*. This follows immediately from Proposition 2.8 and Lemma 2.9 in combination with Definition 3.6, provided that for $a \in Act : X$ is a complete configuration of $ref(a)$ iff X is complete in $ref'(a) = \mathcal{C}(ref(a))$. This is the case if *ref* is deadlock-free.

The second requirement is straightforward. ■

4 Refinement of transitions in Petri nets

We start by giving some basic definitions and notations for Petri nets; for explanations and concepts we refer to introductory texts on nets, e.g. REISIG [110].

For simplicity we assume that there is a one to one correspondence between the transitions in the net and the actions that the system modelled by the net can perform; we do not consider nets with labelled transitions. However, we will show later that our approach can easily be extended to this case.

4.1 Definition $N = (S, T, F)$ is called a *net structure* iff

- S is a set (of *places*),

- T is a set (of *transitions*), $S \cap T = \emptyset$,
- $F \subseteq (S \times T) \cup (T \times S)$ such that $\forall t \in T : \exists s, s'$ with sFt and tFs' (transitions have non-empty pre- and postsets) and $\forall s \in S : sFt \Longrightarrow \neg\ tFs$ (no self-loops).

The restrictions we have made here — non-empty pre- and postsets of transitions and no self-loops — will be needed for our refinement construction.

Two nets $N = (S, T, F)$ and $N' = (S', T', F')$ are *isomorphic* — notation $N \cong N'$ — if $T = T'$ and there exists a bijective mapping $f : S \longrightarrow S'$ satisfying $sFt \Longleftrightarrow f(s)F't$ and $tFs \Longleftrightarrow tF'f(s)$.

Generally, we will not distinguish isomorphic net structures.

As usual, we introduce the following notations.
For $x \in S \cup T$, let ${}^\bullet x := \{y \in S \cup T | yFx\}$ (*preset of* x),
$x^\bullet := \{y \in S \cup T | xFy\}$ (*postset of* x).
Let ${}^\circ N := \{x \in S \cup T | {}^\bullet x = \emptyset\}$ (*initial places of* N),
$N^\circ := \{x \in S \cup T | x^\bullet = \emptyset\}$ (*final places of* N).
Note that ${}^\circ N, N^\circ \subseteq S$.

The components of a net N will be denoted by S_N, T_N, F_N (the index is omitted when clear from the context). We will sometimes use the characteristic mapping of F as a function $F : (S \times T) \cup (T \times S) \longrightarrow \{0, 1\}$.

A concurrent system may be modelled by a net structure where the places carry tokens, indicating the state of the system. The dynamic behaviour of the system is derived by the so called firing rule. We assume that all places have unbounded capacities; any mapping $M : S_N \longrightarrow \mathbb{N}$ will be called a *marking* of the net N. However, we will restrict our considerations to one-safe nets here. We will illustrate later why refinement in non-one-safe nets may lead to problems.

4.2 Definition

(N, M_o) is called a P/T-system or a *marked net* iff N is a net structure and $M_o : S \longrightarrow \mathbb{N}$ (*initial marking*).

By abuse of notation, we will use N both for (N, M_o) (when M_o is clear from the context) and for the underlying net structure.

4.3 Definition Let (N, M_o) be a marked net, $M, M' : S \longrightarrow \mathbb{N}$, $t \in T$.

(i) t is *enabled by* M iff $\forall s \in {}^\bullet t : M(s) > 0$.

(ii) M' is reached from M by firing t ($M[t > M'$) iff
t is enabled by M and
$\forall s \in S : M'(s) = M(s) - F(s,t) + F(t,s)$.

The *marking class* $[N, M_o>$ of a marked net (N, M_o) is then defined as the set of all markings reachable from M_o by finitely many transition firings. A marked net is *one-safe* if $\forall M \in [N, M_o>$, $\forall s \in S : M(s) \leq 1$. In one-safe nets, we may use set notations for markings: $M \subseteq S$ is the marking where exactly the places in M carry a token.

Whenever refering to a marked net in the following, we assume it to be one-safe.

A conceptual framework for refinement in Petri nets are *net morphisms* [50]. A net morphism is a mapping between the elements of two net structures such that the distinction between places and transitions is observed to some extent. It is possible to map, for example, a place to a transition, but only if this place is surrounded by transitions with the same image.

4.4 Definition Let $N = (S, T, F)$, $N' = (S', T', F')$ be net structures.

(i) A mapping $f : S \cup T \longrightarrow S' \cup T'$ is called a *net morphism* iff
$\forall x, y \in S \cup T$ with $f(x) \neq f(y)$ and $(x, y) \in F : [(f(x), f(y)) \in F'$ and $x \in S \Leftrightarrow f(x) \in S']$.

(ii) A net morphism $f : S \cup T \longrightarrow S' \cup T'$ is called a *quotient* iff f is surjective and $(x', y') \in F' \Longrightarrow \exists (x, y) \in F$ with $f(x) = x', f(y) = y'$ (surjectivity also with respect to arcs).

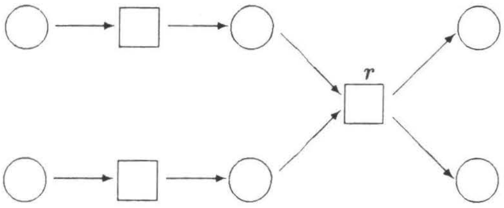
A quotient can be thought of as a factorisation. The net is partitioned such that sorts are preserved: each subset of elements forming a class in this partition must have a boarder consisting just of places or just of transitions and is then considered as one place or one transition, respectively. A quotient N_1 of a net N_2 is considered as an *abstraction* of N_2 (Reisig [111]). Conversely, N_2 is then called a *refinement* of N_1. In this framework, transitions as well as places may be refined.

However, behavioural aspects are not taken precisely into account and this may lead to problems.

4.5 Example

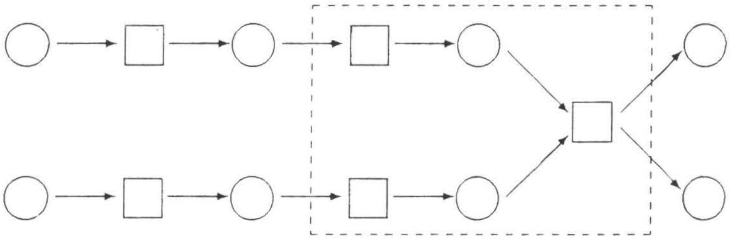
Consider

$N_1 =$



The net N_1 is an abstraction of

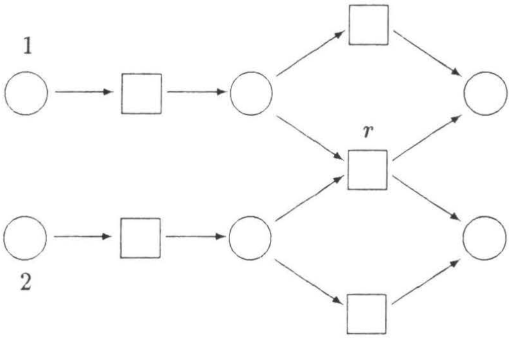
$N_2 =$



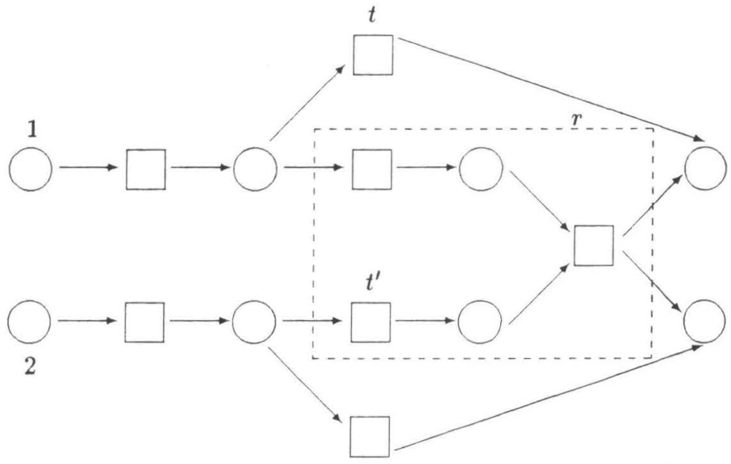
by the quotient mapping all elements inside the broken line to r (and otherwise the identity). Conversely, N_2 is considered as a refinement of N_1.

However, consider the slightly enlarged systems

$N_1' =$



and

$N_2' =$ 

Again, N_1' is a quotient of N_2', hence N_2' may be considered as a refinement of N_1'.

Assuming that places 1 and 2 are initially marked, we find that the net N_1' is deadlock–free in the sense that it is possible to fire transitions until the two final places are both marked. However, even though the part of N_2' corresponding to r is also deadlock-free (namely N_2 is deadlock–free), N_2' may reach a deadlock situation by firing t and t'.

This shows that the notion of a net morphism or quotient is in general not strong enough to reason about the behaviour of refinements in a compositional way. An attempt to restrict it in such a way that behavioural aspects are taken more strongly into account has been made in DESEL & MERCERON [47]. They identify a subclass of morphisms they call *vicinity respecting*. The essential idea is that those net morphisms respect the impact of elements on their environment.

4.6 Definition

A net morphism $f : N \longrightarrow N'$ is said to be *vicinity respecting* iff $\forall x \in S \cup T$:

- $f({}^{\odot}x) = \{f(x)\} \vee f({}^{\odot}x) = {}^{\odot}f(x)$, and
- $f(x^{\odot}) = \{f(x)\} \vee f(x^{\odot}) = f(x)^{\odot}$,

where ${}^{\odot}x := \{x\} \cup {}^{\bullet}x$, $x^{\odot} := \{x\} \cup x^{\bullet}$, respectively.

The morphisms considered in Example 4.5 are not vicinity respecting. We will discuss later to what extent this notion does indeed characterise the refinements we are interested in.

In order to avoid confusion, we have to mention here another notion of morphism suggested for Petri nets by WINSKEL [128]. This notion is particularly tailored to take behavioural aspects into account, however it does not allow to contract for example a line of two transitions into one transition. So it is not suited for treating refinement. More recent approaches in the categorical framework [88, 80] have not yet been evaluated under this aspect.

For the case of refining transitions, which we are interested in here, also more constructive approaches are being considered explaining how to replace a transition in a net by a "refinement net". The problem is to specify how to connect the "refinement net" to the environment of the refined transition, and to investigate what restrictions on refinement nets are then necessary for a sensible refinement operation.

One possibility is to require a one to one correspondence between "input/output-places" of the refinement net and the surrounding places of the refined transition. In VOGLER [124], a construction for this case is proposed, and it is shown that it is then necessary to impose certain restrictions on refinement nets, in particular disallowing initial concurrency (otherwise a situation as in Example 4.5 might occur).

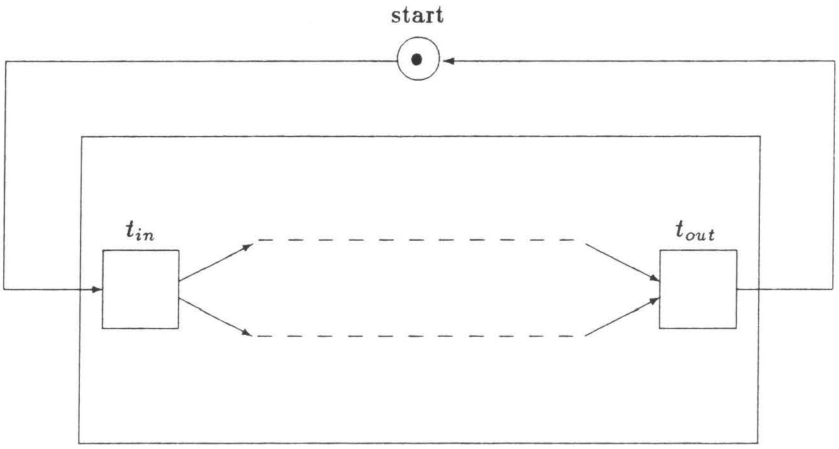
Most constructions for refining transitions are based on distinguishing initial and final transitions in a refinement net and connecting them to the preset and postset, respectively, of the refined transition (VALETTE [123] and subsequently SUZUKI & MURATA [115], VOGLER [125] and BEST, DEVILLERS, KIEHN & POMELLO [27]).

In these approaches, the main idea is that a transition may only be replaced by a net behaving like a transition with respect to its effect on the environment:

- it cannot move without being activated by the environment,
- it has the same possible behaviours whenever it is activated,
- it may not deadlock,
- it consumes and produces tokens in a coincident manner.

The final condition ensures that the problematic situation explained in Example 4.5 may not occur. VALETTE [123] and others ensure this property by allowing only refinements for transitions with at most one initial and at most one final transition. VOGLER [125] generalises this by allowing several initial transitions which must be in conflict (and, symmetrically, the same for final transitions). This means that we may not have initial or final concurrency in refinement nets.

The other requirements are usually ensured by extending the net which is supposed to be substituted for a transition by a new place supplying a token to the initial transition(s) and receiving a token from the final transition(s) and then analysing the behaviour of this net.



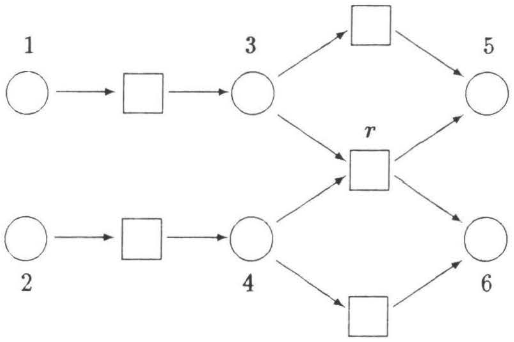
The interesting problem discussed in Example 4.5 was to refine a transition by some behaviour exhibiting initial concurrency. Symmetrically, we also want to allow refinements with final concurrency. This may not be handled in these approaches (VOGLER [124] excludes only initial concurrency). A possibility to get rid of this restriction which has not yet been persued further is to restrict the environment of transitions which are refined.

Here we propose a construction which generalises the approach of VALETTE [123] and VOGLER [125] for the class of one-safe nets without self-loops, and which offers the possibility of refining transitions also with initial and final concurrency. This will be achieved by extending these approaches by specifying explicitly which initial transitions should be concurrent or in conflict (additionally to constraints already imposed by the internal structure of the refinement net). For this, we extend the refinement net with initial places in the preset of initial transitions. Similarly, we add end places specifying the relationsship between final transitions. Clearly, in the refinement net, initial places have no ingoing arcs and final places have no outgoing arcs. When analysing the behaviour of a refinement net, we assume that all initial places (and no final places) carry tokens. As in VALETTE [123], we allow that also other places in a refinement net carry initial tokens. The approaches of VALETTE [123] and VOGLER [125] may be seen as a special case of our approach by splitting the start-place considered above into two places: one initial and one final place. Since we will require as VALETTE [123] that a refinement net has the same possible behaviour whenever it is activated, it is reasonable to assume that the initial places are just those places without ingoing arcs and the final places just those without outgoing arcs. The initial and final places will then be used in the embedding construction to ensure that causal dependencies are preserved by the refenement operation.

4.7 Example

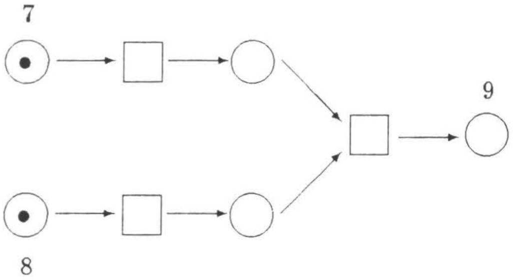
Consider again the net N_1' of Example 4.5.

$N_1' =$



We tried to refine r by two concurrent transitions followed by another transition which causally depends on both of them. This refinement of r may be represented as

$R =$

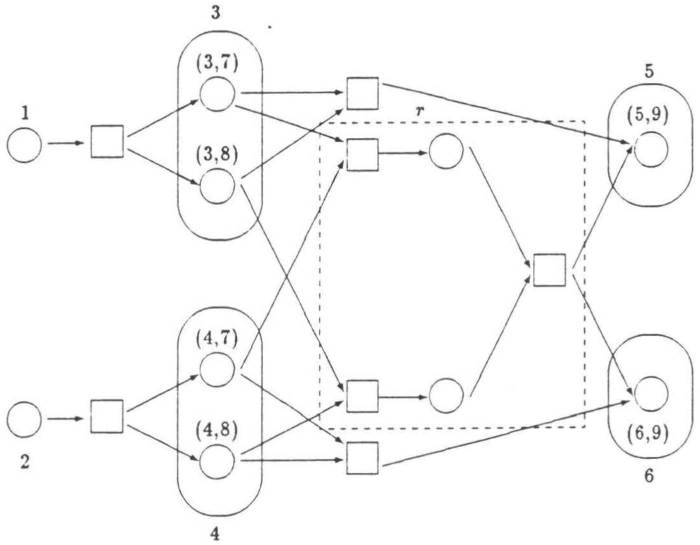


.

Places 7 and 8 are initial places, place 9 is the final place.

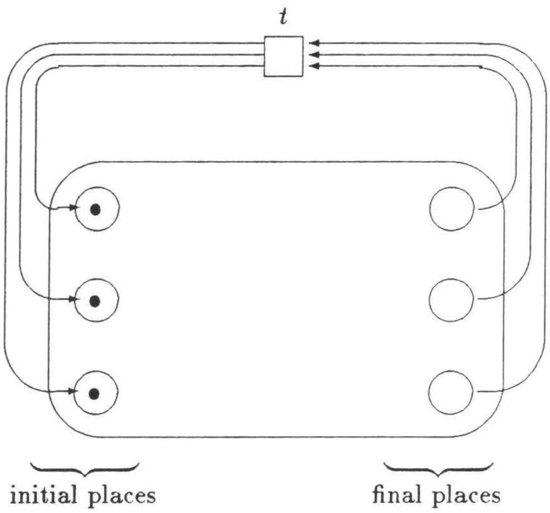
Now R is inserted into N_1' for the transition r by taking the cartesian product of the preplaces of r with the initial places of R and of the postplaces of r with the final places of R. We obtain

$N_2'' =$



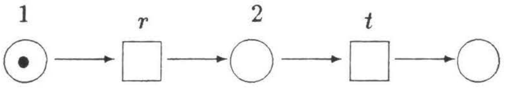
> N_2'' is a again a quotient of N_1', however the mapping between places is no longer the identity. We see that, even though tokens are not removed coincidently by the refinement of r, we have ensured that either both transitions in the refinement of r will fire or none of them, hence N_2'' will not deadlock. This has been achieved by preserving precisely the conflict and causality structure.

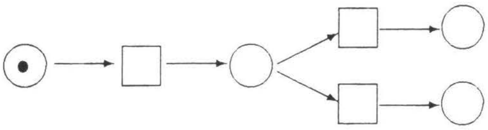
In contrast to the approaches similar to VALETTE [123], we do allow to refine transitions by deadlocking behaviours (where we use the word deadlock in the usual intuitive meaning rather than in the net theoretic sense). The reason is that we do not expect that the properties of the original net, like deadlock-freeness, are preserved by refinement. We only require that the properties of the resulting net are derivable in a compositional way. Whether or not a net to be inserted deadlocks is specified by its behaviour with respect to its final places. A refinement net *deadlocks* if it may reach a situation where no transition may fire but not all its final places are marked. This may be explained by putting the refinement net in a context by connecting its final and its initial places by a transition.



The refinement net deadlocks iff t may not occur.

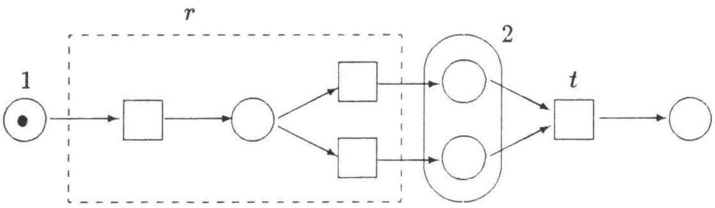
4.8 Example

Let N =  .

Let R =  .

R will deadlock since not all its final places can get a token.

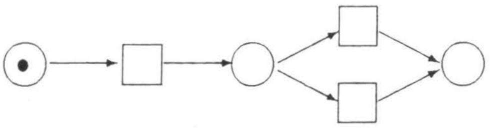
When replacing R for r, we get



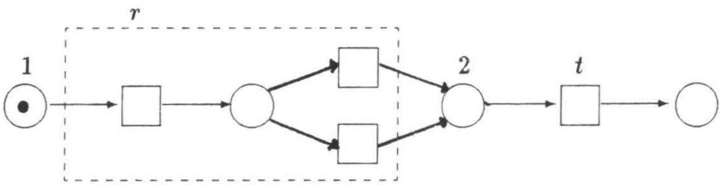
where t will never occur.

However, replacing r by

$R' =$



gives

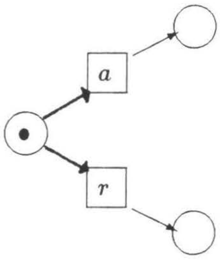


where t will occur.

The next example shows that it is not possible to consider places which have ingoing arcs as initial places of a refinement net.

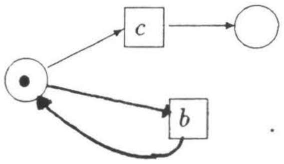
4.9 Example

Let $N =$ and

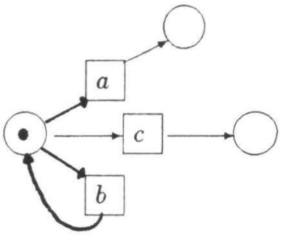


consider the net

$R =$

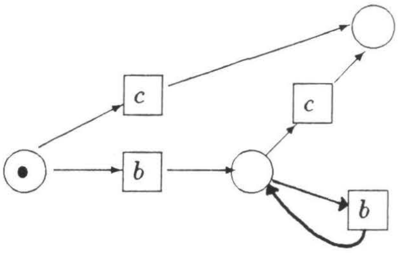


If we would replace R for r, we would obtain

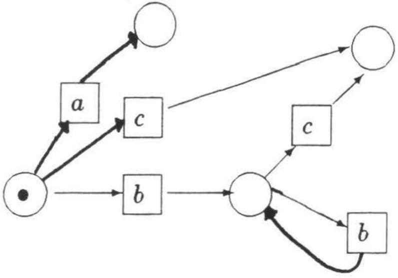


which has not the expected behaviour, since once the refined r has been chosen, no a should be possible any more.

This problem can be solved by using labelled nets and unfolding R into

$R' =$ 

Inserting R' into N yields



which has indeed the expected behaviour.

Next, we will define our construction formally and, in particular, describe formally the requirements on nets which may be inserted for transitions. We will then relate our construction to the notion of vicinity respecting net morphisms and to our approach for refinement in event structures.

4.10 Definition

(N, M_o) with $N = (S, T, F)$ is a *refinement net* iff

- ${}^oN \neq \emptyset$ and $N^o \neq \emptyset$,
- ${}^oN \subseteq M_o$ and $N^o \cap M_o = \emptyset$,
- no $t \in T$ is enabled by $M_o - {}^oN$,
- for any $M \in [N, M_o>$ with $N^o \subseteq M$ we have $M - N^o = M_o - {}^oN$, (N will exhibit identical behaviour when reactivated).

4.11 Definition

Let (N, M_o) be a marked net, let $r \in T_N$.

Let (R, M_o^R) be a refinement net, w.l.o.g. $T_N \cap T_R = \emptyset, S_N \cap S_R = \emptyset$.

Then $N[R/r] := (S, T, F)$ is defined by

$$\begin{aligned} S &:= (S_N - ({}^\bullet r \cup r^\bullet)) \cup (S_R - ({}^\circ R \cup R^\circ)) \cup \mathit{Int} \\ &\qquad \text{where } \mathit{Int} := ({}^\bullet r \times {}^\circ R) \cup (r^\bullet \times R^\circ), \end{aligned}$$

$$T := (T_N - \{r\}) \cup T_R,$$

$$\begin{aligned} F := {} & (F_N \cup F_R)\lceil(S \times T \cup T \times S) \\ & \cup\{((s_N, s_R), t) | (s_N, s_R) \in \mathit{Int}, \\ & (t \in T_N \setminus \{r\} \wedge (s_N, t) \in F_N) \ \vee (t \in T_R \wedge (s_R, t) \in F_R)\} \\ & \cup\{(t, (s_N, s_R)) | (s_N, s_R) \in \mathit{Int}, \\ & (t \in T_N \setminus \{r\} \wedge (t, s_N) \in F_N) \ \vee (t \in T_R \wedge (t, s_R) \in F_R)\} \end{aligned}$$

and $(N, M_o)[R/r] = (N[R/r], M_o^{[R/r]})$ with

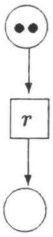
$$\begin{aligned} & M_o^{[R/r]}(s) = M_o(s) \text{ iff } s \in S_N,\ M_o^{[R/r]}(s) = M_o^R(s) \text{ iff } s \in S_R, \\ & M_o^{[R/r]}(s) = M_o(s_N) \text{ iff } s = (s_N, s_R) \in \mathit{Int}. \end{aligned}$$

It is straightforward to verify that $N[R/r]$ is again a one-safe net.

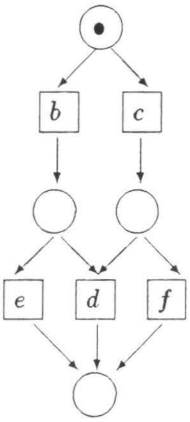
The following example illustrates why we restrict ourselves to one–safe nets (a similar example is given in BEST, DEVILLERS, KIEHN & POMELLO [27]).

4.12 Example

Consider the net $N =$

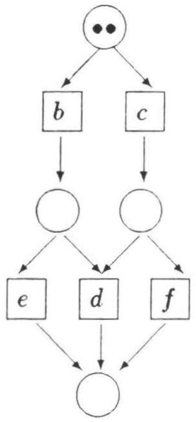


and the refinement $R =$



for r .

When replacing r by R, we would obtain



However, this net has not the expected behaviour, since the two independent occurrences of the refined r–transition may now cooperate and execute d. As remarked in VALETTE [123], this problem can only occur if in N the refined transition can be "two-enabled".

Next we show that the order in which transitions are replaced does not matter. In particular, this means (at least for finite nets) that we can extend our approach to non–injective labellings of transitions by action names by refining all transitions labelled by the same action one by one by disjoint copies of the corresponding refinement net.

4.13 Proposition

Let (N, M_o) be a marked net, $r_1, r_2 \in T_N$, $r_1 \neq r_2$, and let R_1, R_2 be refinement nets. Then $N_1 = ((N, M_o)[R_1/r_1])[R_2/r_2]$ is isomorphic to $N_2 = ((N, M_o)[R_2/r_2])[R_1/r_1]$.

Proof Straightforward. ■

We now show that, for any refinement $N[R/r]$, there exists a canonical vicinity respecting net morphism from $N[R/r]$ to N.

4.14 Proposition

Let (N, M_o) be a marked net, let $r \in T_N$, let R be a refinement net.

Then $f : N[R/r] \longrightarrow N$ with

$$f(x) = \begin{cases} x & \text{iff} \quad x \in (S_N - ({}^\bullet r \cup r^\bullet)) \cup (T_N - \{r\}), \\ r & \text{iff} \quad x \in (S_R - ({}^\circ R \cup R^\circ)) \cup T_R, \\ s_N & \text{iff} \quad x = (s_N, s_R) \in Int \end{cases}$$

is a vicinity respecting morphism, in particular a quotient.

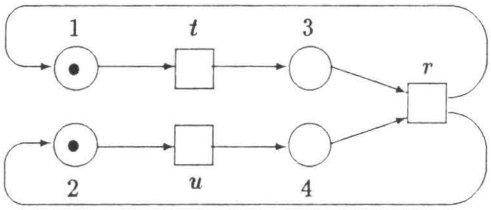
Proof Straightforward ■

We have shown that our construction may be understood in terms of vicinity respecting quotients. However, one could now pose the converse question. May any vicinity respecting quotient which refines only transitions, that is never maps a transition to a place, be generated by our construction? The answer is no, as shown in the following example. However, we would not consider the morphism in this example as a sensible transition refinement.

4.15 Example

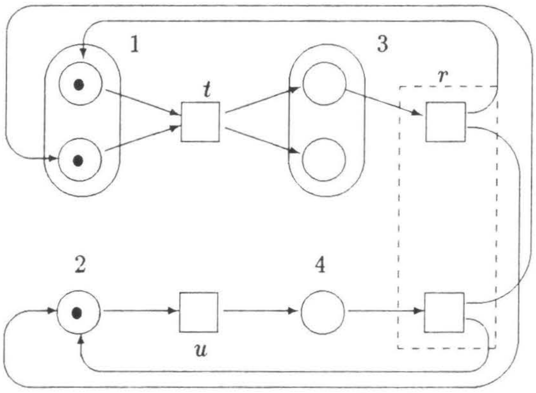
Consider

N_1 =



and

N_2 =



.

The broken lines in N_2 indicate a quotient from N_2 to N_1 which is vicinity respecting and maps no transition to a place. However, we would not like to consider this as a transition refinement. To execute both transitions corresponding to r, an intermediate occurrence of u is necessary. N_2 may not be generated as a refinement of N_1 with our construction.

An interesting problem is to find a further restriction to obtain a class of net morphisms characterising refinement.

Finally, we would like to show that the construction for refinement of transitions we have presented is consistent with the refinement operation on event structures. This would mean in particular, that this construction for nets indeed preserves precisely the conflict– and causality structure. We will show this for the special case of *occurrence nets*, nets with acyclic flow relation and only forward branched places. These nets correspond directly to prime event structures as defined in Section 1. As refinement nets, we will consider special (finite) occurrence nets, called *causal nets*, with only unbranched places. Causal nets correspond to conflict-free prime event structures. This yields precisely the class of refinements which we have considered in Section 1.

4.16 Definition

(i) A net structure N is an *occurrence net* iff

- the transitive closure of F is irreflexive,
- $\forall s \in S_N : |{}^\bullet s| \leq 1$,
- $\#_N$ is irreflexive, where for $x, y \in S \cup T$,
  $x \#_N y \iff \exists t, t' \in T_N$ with $t \neq t', t^\bullet \cap {}^\bullet t' \neq \emptyset$, $tF^* x$ and $t'F^* y$,
- $\forall t \in T_N : \{t' \in T_N \mid t'F^* t\}$ finite (axiom of finite causes).

(ii) A net structure N is a *causal net* iff N is an occurrence net and $\forall s \in S_N : |s^\bullet| \leq 1$.

4.17 Definition Let N be an occurrence net.

The *(prime) event structure of* $N, Ev(N)$, is defined as

$$Ev(N) := (T_N, F^* \lceil T_N, \#_N, id_{T_N}).$$

It is straightforward to verify that $Ev(N)$ is indeed a prime event structure [100].

Using these notions, we may now show the consistency of transition refinement in this class of nets with prime event structure refinement as defined in Section 1.

4.18 Theorem

Let N be an occurrence net, let $r \in T_N$; let R be a finite causal net. Then $Ev(N[R/r]) \cong ref(Ev(N))$ where

$$\begin{aligned} ref(r) &:= Ev(R), \\ ref(t) &:= (\{t\}, \{(t,t)\}, \emptyset, \{(t,t)\}) \text{ for } t \neq r \\ &\quad \text{(identical refinement).} \end{aligned}$$

Proof Omitted. ■

More general consistency results, by unfolding marked nets or associating configuration structures with marked nets and relating with our refinement notion in Section 3, have to be left for further research.

Related work

In this chapter we defined a compositional refinement operator on three kinds of event structures and on Petri nets. Our operator on nets can be regarded as a generalisation of the refinement operators of VALETTE [123], SUZUKI & MURATA [115], BEST, DEVILLERS, KIEHN & POMELLO [27] and VOGLER [125] (although we use a less general kind of nets), and we have compared it with the notions of net morphism (REISIG [111]) and vicinity respecting quotients (DESEL & MERCERON [47]). The operator on *prime* event structures was introduced in VAN GLABBEEK & GOLTZ [54]. It has been defined on sets of pomsets – a linear time variant of the model of prime event structures – in GISCHER [51] and on process graphs modelling only sequential processes in VAN GLABBEEK & WEIJLAND [63] (Section 6 of the previous chapter).

In principle there are two ways to treat "syntactic" action refinement in system description languages like CCS. One of them is to use the CCS–actions for modelling the refinable actions of this paper. In the absence of communication (or synchronisation) refinement can simply be defined as syntactic substitution of an action by a process expression. This approach has been taken in ACETO & HENNESSY [3] and NIELSEN, ENGBERG & LARSEN [99], and has also been mentioned in CASTELLANO, DE MICHELIS & POMELLO [36]. In the presence of communication defining such a refinement operator is much more difficult. A first proposal, for the simple case of an operator only splitting actions in two parts to be executed sequentially, can be found in VAN GLABBEEK & VAANDRAGER [59].

An alternative is to use the actions of CCS for modelling "atomic" or instantaneous actions that cannot be refined, and representing our refinable actions by means of variables or *parameters*. This approach requires a general sequential composition operator and has been carried out in BERGSTRA & TUCKER [26] in the setting of ACP. In particular [26] shows that there is no problem in defining a refinement operator while working in interleaving semantics: atomic actions a, b cannot be refined, so the equation $a \parallel b = a;b + b;a$ is harmless; parameters x, y can be refined, but there is no equation $x \parallel y = x;y + y;x$. Of course the refinement operator, ordinary substitution, is defined in the language (that still contains all information about causal dependence) and not in the associated interleaving model (which would be impossible according to

Example 0.4).

A completely different approach is taken in GORRIERI, MARCHETTI & MONTANARI [64] and BOUDOL [29]. There all actions are assumed to be "atomic", and this property should be preserved if they are refined. In [29] even two kinds of atomicity are proposed, corresponding with two kinds of refinement. In [64] this kind of refinement is carried out in an interleaving based model, as mentioned in the introduction.

Refinement in more concrete programming languages is treated in GRIBOMONT [67].

It is often argued that a concurrent system should not be represented just by a Petri net or an event structure, but rather by an equivalence class of such objects. Action refinement is only well-defined on a quotient domain induced by a semantic equivalence if this equivalence is a congruence for refinement, i.e. if $P = Q \Longrightarrow \mathit{ref}(P) = \mathit{ref}(Q)$. The search for suitable equivalences has been reported e.g. in [36, 54, 53, 125, 3, 99, 27 and 63], and will be the topic of the remaining chapters of this thesis.

Chapter V

Partial Order Semantics for Refinement of Actions

- neither necessary nor always sufficient but appropriate when used with care -

Rob van Glabbeek & Ursula Goltz

Notes: This chapter appeared originally in Bulletin of the EATCS 38, pp. 154–163. It also appeared as Report CS-N8901, Centre of Mathematics and Computer Science, Amsterdam 1989.
Here it serves as an informal summary of the remaining two chapters of this thesis. It uses Petri nets rather then event structures and contains no technicalities like definitions and proofs. Instead more attention has been paid to the examples.
The research of the first author was supported by Esprit project 432 (METEOR).

Originally this chapter was written in continuation of a series of papers in the Bulletin of the EATCS about the relative merits of partial order semantics and interleaving semantics, starting with CASTELLANI, DE MICHELIS & POMELLO [36]. That paper pointed out a significant advantage of partial order semantics, by formulating a desirable property of semantic equivalences that is not met by interleaving equivalences. This property is *preservation under refinement of actions.* A semantic equivalence is *preserved under action refinement* if two equivalent processes remain equivalent after replacing all occurrences of an action a by a more complicated process $r(a)$. For example, $r(a)$ may be a sequence of two actions a_1 and a_2. This property may be desirable in applications where concurrent systems are modelled at different levels of abstraction, and where the actions on an abstract level turn out to represent complex processes on a more concrete level. Therefore for example PRATT [108] and LAMPORT [83] already advocate the use of semantic equivalences that are not based on action atomicity.

CASTELLANO, DE MICHELIS & POMELLO [36] showed by means of a simple example that none of the interleaving equivalences - not even bisimulation - is preserved under action refinement. Furthermore they claim that 'on the other hand, the approaches based on partial order are not constrained to the assumption of atomicity'. Indeed, they give a proof that "linear time" partial order semantics, where a system is identified with the set of its possible (partially ordered) runs, is preserved by refinement. They conclude that 'interleaving semantics is adequate only if the abstraction level at which the atomic actions are defined is fixed. Otherwise, partial order semantics should be considered'.

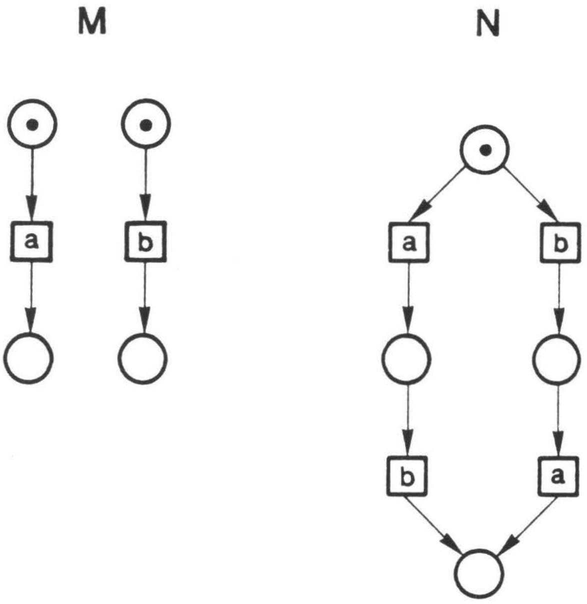
In this chapter we would like to point out that this conclusion is not so obvious. In particular we will argue

- that there are several equivalences based on partial orders which are *not* preserved by refinement (namely when taking the choice structure of systems into account);
- that nevertheless a "branching time" partial order equivalence can be found that is preserved under refinement;
- but that, in order to achieve preservation under refinement it is not necessary to employ partial order semantics: there exist equivalences that abstract from the causal structure of concurrent systems and are still preserved under refinement.

In interleaving semantics, the possible runs of a system are represented as sequences of action occurrences, modelling parallelism by arbitrary interleaving of actions. The example of [36] consisted of the two systems M and N which may not be distinguished in this kind of semantics:

M = $a \,||\, b$ (two actions a and b, executed independently);
N = $a; b + b; a$ (either the sequence ab or the sequence ba is executed).

They have the following Petri net representations (labelling transitions by action names):



It was shown that after refining a into the sequential composition of a_1 and a_2, thereby obtaining the systems

$$M' = (a_1; a_2) \parallel b \text{ and } N' = (a_1; a_2); b + b; (a_1; a_2),$$

M' can perform the sequence of actions a_1ba_2, while N' cannot do this. Hence M' and N' are not equivalent in interleaving semantics.

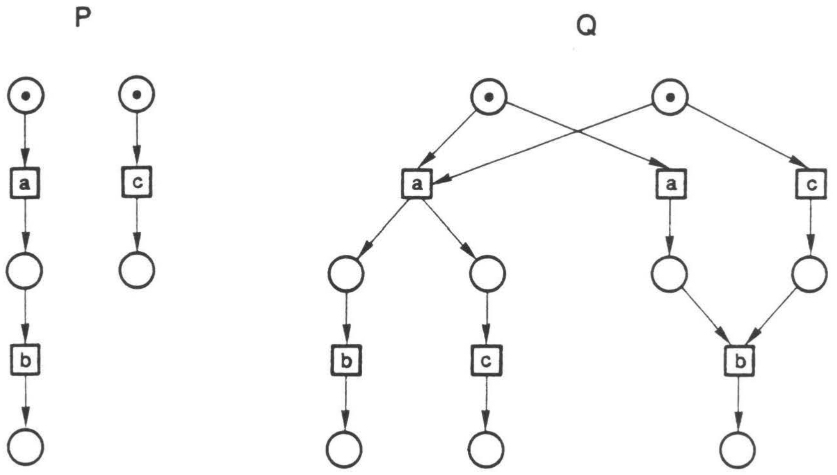
A first attempt to capture parallelism more precisely is made by so called *step semantics.* Here it is specified that in a run of a parallel system several independent actions may occur together in one *step.* We can think of a system having a global clock where at each clock tick several actions occur simultaneously. This view is taken in calculi like SCCS [94], CIRCAL [91] and MEIJE [6]. Step semantics also have been given to CCS in [41] and to TCSP in [116].

It is easy to see that the two systems M and N considered above are already distinguished in step semantics: In M it is possible to execute the step $\{a, b\}$ whereas in N it is not. So the example in [36] is not well chosen to advocate partial order semantics; already step semantics would be sufficient in this case. Therefore, we will now give a slightly more elaborate example. Consider the following two systems:

$P = (a; b) \parallel c,$
$Q = a; (b \parallel c) + (a \parallel c); b.$

In both of these systems the actions a, b and c are executed, and b occurs after completion of a. However, in P the c action occurs independently of both a

and b whereas in Q c may only occur either "causally behind" a or "causally before" b. P and Q may be represented by the following Petri nets (using a construction explained for instance in [59] for implementing $+$).



P and Q are identified when considering their possible sequences of steps. Both of them take into account the five possibilities for c: occurring before a, simultaneous with a, between a and b, simultaneous with b, or after b. However, after substituting $(c_1; c_2)$ for c only the first system can perform the sequence of actions c_1abc_2. Thus also this semantics is not preserved under refinement.

On the other hand, P and Q can be distinguished by considering the *partial orders of action occurrences* they allow.

$$\begin{array}{ccc} a & \rightarrow & b \\ & c & \end{array} \qquad \begin{array}{l} (a \text{ followed by } b \text{ and} \\ \quad \text{independently } c) \end{array}$$

is a computation of P but not of Q. In [36] it was shown that partial order semantics - when identifying a system with its set of possible (partially ordered) runs - is preserved under action refinement.

However, when taking the choice structure of systems into account, the situation becomes less obvious.

Before discussing the problem in detail, we would like to give an overview, by classifying the equivalences being currently investigated (without claiming completeness). They may be positioned in a two dimensional diagram as shown below, distinguishing them firstly with respect to the preserved level of detail in runs of systems (as discussed above) and secondly with respect to the preserved level of detail of the choice structure of systems (we do not consider abstraction from internal actions here). In trace semantics ("linear time" semantics), a system is fully determined by its set of possible runs, thereby completely neglecting the branching structure. On the other end, bisimulation semantics preserve the information where two different courses of action diverge (although branching of identical courses of action is still neglected). In between there are several "decorated trace semantics", where part of the branching structure is taken into account. Mostly these are motivated by the observable behaviour of processes, according to some testing scenerio (see Chapter I).

runs branching structure	sequences of actions	sequences of steps	partial orders
paths	interleaving trace equivalence	step trace equivalence	pomset trace equivalence
⋮ e.g. testing			
bisimulation	interleaving bisimulation equivalence	step bisimulation equivalence	e.g. pomset bisimulation equivalence

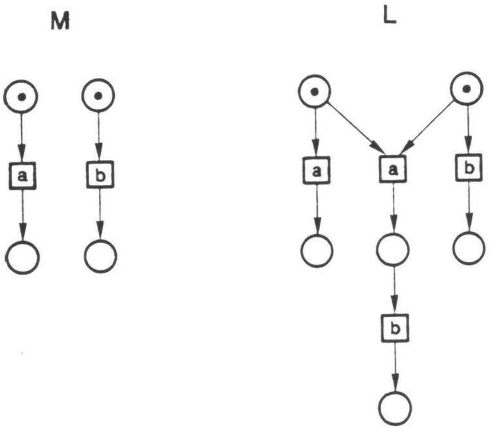
Up to now we have only considered the trace equivalences in the upper row of the diagram. We recalled from [36] that pomset trace equivalence is preserved under action refinement, while interleaving trace equivalence is not. Moreover we have shown that also step trace equivalence is not preserved under refinement. Next we will try to establish similar results for the corresponding branching time equivalences and for the testing equivalences in between.

In interleaving semantics this generalisation is quite simple. As observed in [36], the systems M and N are identified even in interleaving bisimulation semantics while the refined systems M' and N' are not even identified in interleaving trace semantics. So there is one single example showing that neither interleaving bisimulation equivalence nor interleaving trace equivalence is preserved under refinement. As a consequence, also none of the decorated trace equivalences based on interleaving, which are more discriminating then interleaving trace equivalence, but less discriminating then interleaving bisimulation equivalence, is preserved under refinement; in each of the decorated trace semantics based on interleaving, M and N are identified, while M' and N' are distinguished.

Our example against step trace equivalence however cannot be used to show that also step bisimulation equivalence is not preserved under refinement; the systems P and Q happen to be different in step bisimulation semantics already: after performing an a-action the system P is always able to continue with a b-action, whereas Q can perform an a-action and reach a state where it is not possible to continue with b. Nevertheless, the following example shows that also step bisimulation semantics is not preserved under refinement. Consider the two systems M and L which may not be distinguished in step bisimulation semantics:

$M = a \parallel b$	(two actions a and b, executed independently);
$L = a \parallel b + a;b$	(either a and b are executed independently or the sequence ab is executed).

They have the following Petri net representations:



The systems $M' = (a_1;a_2) \parallel b$ and $L' = (a_1;a_2) \parallel b + (a_1;a_2);b$ which are obtained by substituting $a_1;a_2$ for a are no longer step bisimulation equivalent; only L' can perform a_1, and reach a state where it is not possible to continue with b.

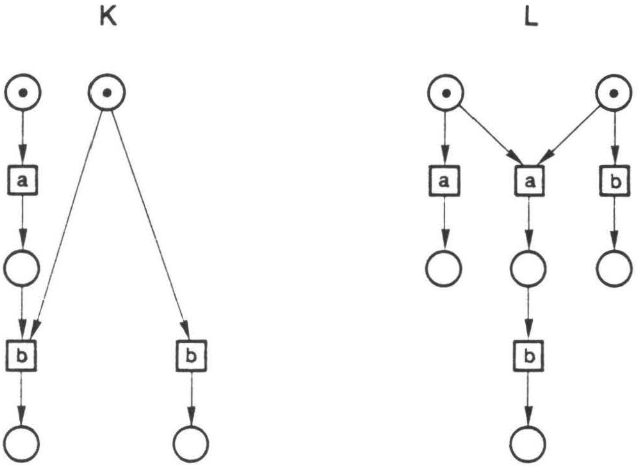
Hence, neither step trace nor step bisimulation equivalence is preserved under refinement. However, M' and L' happen to be step trace equivalent, so none of the previous two examples is adequate for both equivalences. In order to tackle the whole range of equivalences included between step trace and step bisimulation equivalence we need yet another example, which simultaneously shows that both step trace and step bisimulation equivalence are not preserved under refinement. Consider the systems

$$Q = a; (b \parallel c) + (a \parallel c); b \quad \text{and}$$
$$R = Q + P = a; (b \parallel c) + (a \parallel c); b + (a; b) \parallel c.$$

The Petri net associated to Q has been shown before, and the net for R is drawn in [59], where it was also pointed out that Q and R are step bisimulation equivalent. However, after refining c into $c_1; c_2$ the two systems are not even interleaving trace equivalent; only the second system can perform the sequence of actions c_1abc_2. As a consequence, none of the decorated trace equivalences based on steps, such as the step failure semantics of [116], is preserved under refinement.

A rather straightforward combination of the ideas of bisimulation and of capturing causal dependencies by partial orders has been proposed in BOUDOL & CASTELLANI [31]. They suggest to consider transition systems as for the usual interleaving bisimulation, but to label the arcs in these transition systems by *pomsets* (partially ordered multisets of action occurrences) instead of single actions. However, it turns out that the obtained equivalence, usually called *pomset bisimulation*, is not preserved by refinement of actions.

Consider the two systems K and L below.



In both systems either a and b are executed independently or the sequence ab is executed. However, in L the choice between these two options is made at the beginning, while in K this choice can be postponed until the execution of a has been completed.
The system K can behave as follows:

- it performs the single action a and the remaining behaviour is $b + b$, which is identified with b;

- it performs the single action b and the remaining behaviour is a;

- it performs the pomset $\begin{matrix} a \\ b \end{matrix}$ (a and b executed independently) and there is no remaining behaviour;

- or it performs the pomset $a \longrightarrow b$ (a followed by b) and again there is no remaining behaviour.

The behaviour of L can be described in exactly the same way and for this reason the two systems are pomset bisimulation equivalent.

Now let us imagine that a is refined into $a_1; a_2$. The systems K' and L' which are obtained in this way can be distinguished in pomset bisimulation semantics, and even in interleaving bisimulation: only L' can refuse to do a b-action after execution of a_1.

Hence pomset bisimulation semantics is not preserved under refinement of actions. Another example for this are the two terms

$$a;(b+c)+(a \parallel b) \text{ and } a;(b+c)+(a \parallel b)+(a;b)$$

(again refining a into $a_1; a_2$). However the example given before can also be used to show that even the notion of generalised pomset bisimulation, as discussed in Van Glabbeek & Vaandrager [59], is not preserved under refinement. Of course we cannot find an example tackling the whole range of equivalences included between pomset trace and pomset bisimulation semantics, since we already observed that pomset trace equivalence is preserved under refinement. However, the systems K' and L' can already be distinguished in interleaving failure semantics, as employed in [33, 43]. Thus no equivalence that is at least as discriminating as interleaving failure equivalence but less discriminating then pomset bisimulation equivalence can be preserved under refinement.

The interplay of equivalence notions and refinement of actions as discussed up to now is investigated in detail in Chapter VI. There all the equivalence notions and examples presented so far are given formally in the framework of event structures; refinement of actions is performed by replacing actions by non-empty pomsets. That paper concludes by showing that another "partial order bisimulation" is indeed preserved by refinement. In order to avoid technical details, we just ouline these results here.

After we realised that pomset bisimulation is not preserved by refinement, another equivalence was considered, hoping that it would solve the problem (see e.g. Devillers [48]). This equivalence had been considered before under the name NMS partial ordering equivalence in Degano, De Nicola &

MONTANARI [40]. The main idea is to bisimulate transition systems where the states are labelled by their (partially ordered) histories. In the next chapter it is shown that this equivalence is indeed preserved by refinement when we restrict ourselves to systems *without autoconcurrency*, that is to systems which do not allow concurrent occurrences of the same action like in $a \parallel a$. However, for systems with autoconcurrency it turns out that NMS po equivalence is not preserved by refinement. Even more, it does not even respect pomset bisimulation equivalence in this case. The example showing both these facts was suggested to us by Alex Rabinovich who used it to show that this equivalence is not a congruence with respect to a TCSP-like parallel composition. To obtain a conguence, a stronger version of NMS po equivalence was suggested in RABINOVICH & TRAKHTENBROT [109]. In the next chapter it is shown that *this* "partial order bisimulation equivalence" is indeed always preserved by refinement.

So we have shown that it is *not automatically sufficient* to move to partial order semantics for refinement of actions. When considering the choice structure, this has to been done with care. In the remaining part of this note, we argue that on the other hand it is *not even necessary* to move to partial orders (as one may conclude from [36]).

A branching time semantics lying strictly between step semantics and partial order semantics has been proposed in VAN GLABBEEK & VAANDRAGER [59]. This *ST-bisimulation semantics* is based on the idea that actions have a duration, and may overlap in time. Contrary to step semantics, it recognises the possibility that, in $P = (a; b) \parallel c$, action c may have an overlap with both a and b, while b can only occur after completion of a. However when in a run of a system an action b happens after completion of a, it is not taken into account whether or not there is a causal link between the two actions.

Compare for instance the systems M and K that have been presented before.

Both systems perform an a-action and a b-action. In M these actions are always independent, whereas in K it is possible to perform a b-action which causally depends on a: so M and K are distinguished in partial order semantics. However, in ST-bisimulation semantics the only execution of K which is not possible in M (first a and then the b which is causally dependent on this a) can not be distinguished from another execution of K (and of M), namely: first a and then the b which is independent of this a. In K, the choice between both runs is only made after completion of a, and in that state the remaining part of both executions is the same: just b. Hence M and K are identified in ST-bisimulation semantics.

So ST-bisimulation equivalence abstracts from the causal structure of concurrent systems. Nevertheless it is preserved under refinement, as will be shown in Chapter VII. A similar result can be proved for linear time sẽmantics as well. A variant of this can be found in NIELSEN, ENGBERG & LARSEN [99]. Furthermore a variant of failure semantics, based on the same ideas that underly ST-bisimulation semantics has been proposed in [125]. There it is proven that also this equivalence respects refinement.

This shows that indeed partial order semantics (in the strong sense) are not necessary for the type of refinement we have considered. Nevertheless, we need partial order bisimulation semantics when it is required to model the interplay of causality and branching in full detail.

We hope that this note, and the formal versions of it in the next chapters, help to clarify the relationship between various equivalences being currently considered. However, we do not intend to advocate any particular type of equivalence here. We just want to illustrate that the appropriate equivalence notion has to be chosen carefully with regard to the considered questions.

Chapter VI

Equivalence Notions for Concurrent Systems and Refinement of Actions

Rob van Glabbeek & Ursula Goltz

In this chapter we investigate equivalence notions for concurrent systems. We consider "linear time" approaches where the system behaviour is characterised as the set of possible runs as well as "branching time" approaches where the conflict structure of systems is taken into account. We show that the usual interleaving equivalences, and also the equivalences based on *steps* (multisets of concurrently executed actions) are not preserved by refinement of atomic actions. We prove that "linear time" partial order semantics, where causality in runs is explicit, is invariant under refinement. Finally, we consider various bisimulation equivalences based on partial orders and show that the strongest one of them is preserved by refinement whereas the others are not.

Notes This chapter is based on our paper *Equivalence Notions for Concurrent Systems and Refinement of Actions*, Arbeitspapiere der GMD 366, Sankt Augustin 1989, an extended abstract of which has been published in Proc. MFCS 89, LNCS 379, Springer-Verlag, pp. 237–248, 1989. However, the introduction of prime event structures and the refinement operator as well as the proof of its compositionality has been omitted since this is covered by Section 1 of Chapter IV already. Furthermore our results are reformulated in such a way that they are not only valid for prime event structures but also for flow event structures, as presented in Section 2 of Chapter IV. On flow event structures, infinite refinements and refinements with conflicts are no longer excluded, which strenghtens our refinement theorems.

The research of the first author was supported by Esprit project 432 (METEOR).

Contents

Introduction

A large body of research is devoted to equivalence notions for concurrent systems. Most of the equivalence notions currently being considered are based on a semantics where concurrency is modelled by arbitrary interleaving of atomic actions. In PRATT [108] and in CASTELLANO, DE MICHELIS & POMELLO [36] it is pointed out that this approach has a severe drawback. It leads to complications when changing the level of atomicity of events; "... we would like a theory of processes to be just as usable for events having a duration or structure, where a single event can be atomic from one point of view and compound from another" ([108]). In [36], an example is given, showing that the usual interleaving equivalence is not invariant under refinement of actions when this is simply modelled by textual replacement. Both [108] and [36] claim that modelling concurrency by expressing causal dependencies explicitly using partial orders could help to solve this problem. However, the two systems considered in [36] can already be distinguished by considering interleavings of "steps" (multisets of concurrently executable actions). So their example does not show that it is indeed necessary to consider partially ordered executions. Furthermore, their proof of the claim that partial order equivalence is preserved by refinement is only valid for "linear time" partial order semantics, where the set of all possible executions of a system is considered, without taking into account where conflicts are resolved. This is also the model considered by Pratt.

In this chapter, we will consider various equivalence notions based on steps and on partial orders. We will discuss "linear time" semantics, but we will also take the conflict structure of systems into account by considering various forms of bisimulation ("branching time" semantics). We will show that the known equivalences based on steps are not invariant under action refinement. We will rephrase in our framework the proof of [36], showing that "linear time" partial order semantics is indeed robust against changing the level of atomicity. Then we consider several equivalence notions based on "branching time" partial order semantics. We give examples, showing that pomset bisimulation equivalence of BOUDOL & CASTELLANI [31] and also the NMS partial ordering equivalence suggested in DEGANO, DE NICOLA & MONTANARI [40], are not preserved by refinement of atomic actions. An equivalence notion for Petri nets which coincides with the notion of NMS partial ordering equivalence was suggested in DEVILLERS [48] where the refinement problem has also been discussed. We also show that NMS partial ordering equivalence does not imply pomset bisimulation (and vice versa); hence these notions are incomparable. Finally we show that a stronger equivalence notion, proposed in RABINOVICH & TRAKHTENBROT [109] under the name BS-bisimulation, is indeed preserved by refinement. This equivalence does respect pomset bisimulation.

We do not intend to advocate any particular equivalence notion here, the purpose of this investigation is to find out about the consequences of the different approaches. There will certainly be a tradeoff between simplicity and distinguishing power. We just want to illustrate that the appropriate notion has to be chosen carefully with regard to the questions considered.

1 Interleaving semantics

In this paper, concurrent systems are represented by event structures. It is written in such a way that the text applies both to prime event structures as to flow event structures (but in case of prime event structures $\leq_X$ should be read as $\leq$). The reader is refered to Section 1 – for prime event structures – or Section 2 – for flow event structures – of Chapter IV, for an introduction to event structures and action refinement. There it is also explained how configurations model the states of a concurrent system.

We may now ask which actions may occur in a configuration and which configuration is then obtained.

Definition Let $\mathcal{E}$ be an event structure,

i. $X \longrightarrow_{\mathcal{E}} X'$ if $X, X' \in Conf(\mathcal{E})$ and $X \subseteq X'$.

ii. $X \xrightarrow{a} X'$ iff $a \in Act$, $X \longrightarrow_{\mathcal{E}} X'$ and $X' \setminus X = \{e\}$ with $l(e) = a$.

Note that $X \longrightarrow_{\mathcal{E}} X'$ implies that $\mathcal{E}\lceil(X' \setminus X)$ is finite and conflict-free.

Here $X \xrightarrow{a} X'$ says that if $\mathcal{E}$ is in the state represented by X, then it may perform an action a and reach a state represented by X'. Likewise, $X \rightarrow_{\mathcal{E}} X'$ says that $\mathcal{E}$ may evolve from X to X'.

Considering transitions $X \xrightarrow{a} X'$ only, one can define the usual *interleaving semantics*. The simplest form is that of comparing just the possible sequences of action occurrences.

Definition

$w = a_1 \cdots a_n \in Act^*$ is a *(sequential) trace* of an event structure $\mathcal{E}$ iff there exist configurations $X_o, \cdots, X_n$ of $\mathcal{E}$ such that $X_o = \emptyset$ and $X_{i-1} \xrightarrow{a_i} X_i$ $(i = 1, \cdots, n)$.
SeqTraces $(\mathcal{E})$ denotes the set of all sequential traces of an event structure $\mathcal{E}$.

Two event structures $\mathcal{E}, \mathcal{F}$ are called *interleaving trace equivalent* ($\mathcal{E} \approx_{it} \mathcal{F}$) iff $SeqTraces\,(\mathcal{E}) = SeqTraces\,(\mathcal{F})$.

With the concept of labelled transition systems, we obtain a stronger equivalence notion based on the idea of bisimulation [103, 92]. For example, the systems $a(b+c)$ and $ab + ac$ have the same traces but are distinguished by bisimulation equivalence.

Definition Let $\mathcal{E}, \mathcal{F}$ be event structures.

A relation $R \subseteq Conf(\mathcal{E}) \times Conf(\mathcal{F})$ is called an *interleaving bisimulation between $\mathcal{E}$ and $\mathcal{F}$* iff $(\emptyset, \emptyset) \in R$ and if $(X, Y) \in R$ then
- $X \xrightarrow{a} X' \Rightarrow \exists Y'$ with $Y \xrightarrow{a} Y'$ and $(X', Y') \in R$,
- $X \xrightarrow{a} Y' \Rightarrow \exists X'$ with $X \xrightarrow{a} X'$ and $(X', Y') \in R$.

$\mathcal{E}$ and $\mathcal{F}$ are *interleaving bisimulation equivalent* ($\mathcal{E} \approx_{ib} \mathcal{F}$) iff there exists an interleaving bisimulation between $\mathcal{E}$ and $\mathcal{F}$.

Clearly, $\mathcal{E} \approx_{ib} \mathcal{F}$ implies $\mathcal{E} \approx_{it} \mathcal{F}$.

Example 1.1

We now recall the example of [36], showing that both $\approx_{it}$ and $\approx_{ib}$ are not preserved by refinement.
They considered the two systems $P = a|b$ and $Q = ab + ba$, representable by the following event structures.

$$\begin{tikzcd}
\mathcal{E}_P & = & a & b & , & \mathcal{E}_Q & = & a \arrow[d] & \# & b \arrow[d] \\
 & & & & & & & b & & a
\end{tikzcd}$$

In all known interleaving semantics, P and Q are considered equivalent, we have $\mathcal{E}_P \approx_{ib} \mathcal{E}_Q$. However, if we allow to refine the action a into the pomset $a_1 \rightarrow a_2$, this gives rise to the two systems

$$\begin{tikzcd}
\mathcal{E}_{P'} & = & a_1 \arrow[d] & b & , & \mathcal{E}_{Q'} & = & a_1 \arrow[d] & \# & b \arrow[d] \\
 & & a_2 & & & & & a_2 \arrow[d] & & a_1 \arrow[d] \\
 & & & & & & & b & & a_2
\end{tikzcd}$$

and they are not interleaving equivalent; indeed they are not even interleaving trace equivalent: $\mathcal{E}_{P'}$ allows for the sequence $a_1\, b\, a_2$ whereas $\mathcal{E}_{Q'}$ doesn't.

This shows that both interleaving trace equivalence and interleaving bisimulation equivalence are not preserved by action refinement. Even more, the same can be said for all equivalences identifying P and Q and respecting interleaving trace equivalence, e.g. failure equivalence [33], testing equivalence [43].

As an event structure equivalence which is indeed preserved by refinement one could consider event structure isomorphism.

Theorem Let $\mathcal{E}, \mathcal{F}$ be event structures, let ref be a refinement.

Then $\mathcal{E} \cong \mathcal{F} \Rightarrow ref(\mathcal{E}) \cong ref(\mathcal{F})$.

Proof Straightforward. ■

However, the main purpose of introducing an equivalence notion is to abstract from certain details in a system representation. For example, we would like to express that the processes a and $a + a$ exhibit the same behaviour. Furthermore, we would like to identify processes like $(a|(b+c)) + (a|b) + ((a+c)|b)$ and $(a|(b+c)) + ((a+c)|b)$ (absorption law, see [31]). This is not possible when using event structure isomorphism.

Hence, in the sequel we will consider various equivalence notions in between these two extremes (interleaving trace equivalence and event structure isomorphism), taking into account the concurrency and the conflict structure ("branching-time" semantics) in more and more detail.

2 Step semantics

A more discriminating view of concurrent systems than that offered by interleaving semantics is obtained by modelling concurrency as either arbitrary interleaving or simultaneous execution. This view is taken in calculi like SCCS [94], CIRCAL [91] and MEIJE [6]. In TAUBNER & VOGLER [116], this idea is applied to give a non-interleaving semantics to theoretical CSP, called *step failure semantics*. The word *step* originates from Petri net theory where it denotes a set (or multiset) of concurrently executable transitions. Recently, a step semantics for CCS has been defined [41], inspired by [6]. Step semantics give a more precise account of concurrency than interleaving semantics, e.g. the systems $a|b$ and $ab + ba$ are distinguished. This means that the example given in [36] constitutes an argument against interleaving semantics but not against step semantics. We will formalise some step equivalence notions and then discuss examples which show that even these equivalences are not preserved by refinement.

Step semantics are defined by generalising the single action transitions $X \xrightarrow{a} X'$ from Section 1 to transitions of the form $X \xrightarrow{A} X'$ where A is a multiset over *Act*, representing actions occurring concurrently. In particular, we allow actions to occur concurrently with themselves ("autoconcurrency"). Using this new kind of transitions, *step trace equivalence* and *step bisimulation equivalence* are straightforward generalisations of the corresponding interleaving equivalences, see e.g. POMELLO [107].

Definition Let $\mathcal{E}$ be an event structure.

$X \overset{A}{\rightarrow} X'$ iff $A \in \mathbb{N}^{Act}$ (A is a multiset over Act), $X \rightarrow_{\mathcal{E}} X'$ and $X' \setminus X = G$ such that $\forall e, e' \in G \;\; e \; co \; e'$ and $l(G) = A$ where $l(G)(a) = |\{e \in G | l(e) = a\}|$.

Definition

$W = A_1 \cdots A_n$ where $A_i \in \mathbb{N}^{Act}$ $(i = 1, \cdots, n)$ is a *step trace* of an event structure $\mathcal{E}$ iff there exist configurations $X_o, \cdots, X_n$ of $\mathcal{E}$ such that $X_o = \emptyset$ and $X_{i-1} \overset{A_i}{\rightarrow} X_i$ $(i = 1, \cdots, n)$.
StepTraces $(\mathcal{E})$ denotes the set of all step traces of an event structure $\mathcal{E}$.
Two event structures $\mathcal{E}, \mathcal{F}$ are called *step trace equivalent* $(\mathcal{E} \approx_{st} \mathcal{F})$ iff *StepTraces* $(\mathcal{E})$ = *StepTraces* $(\mathcal{F})$.

Definition Let $\mathcal{E}, \mathcal{F}$ be event structures.

A relation $R \subseteq Conf(\mathcal{E}) \times Conf(\mathcal{F})$ is called a *step bisimulation between $\mathcal{E}$ and $\mathcal{F}$* iff $(\emptyset, \emptyset) \in R$ and if $(X, Y) \in R$ then
- $X \overset{A}{\rightarrow} X' \Longrightarrow \exists Y'$ with $Y \overset{A}{\rightarrow} Y'$ and $(X, Y) \in R$,
- $Y \overset{A}{\rightarrow} Y' \Longrightarrow \exists X'$ with $X \overset{A}{\rightarrow} X'$ and $(X, Y) \in R$.

$\mathcal{E}$ and $\mathcal{F}$ are *step bisimulation equivalent* $(\mathcal{E} \approx_{sb} \mathcal{F})$ iff there exists a step bisimulation between $\mathcal{E}$ and $\mathcal{F}$.

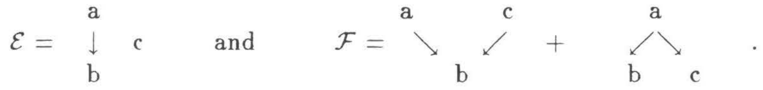
As for interleaving, $\mathcal{E} \approx_{sb} \mathcal{F}$ implies $\mathcal{E} \approx_{st} \mathcal{F}$. Moreover (as far as we know) all other interesting step equivalence notions are positioned somewhere in between (recall that we do not consider abstraction from internal actions).

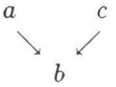
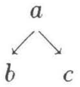
Considering the two systems $P = a|b$ and $Q = ab+ba$ from [36], represented as event structures $\mathcal{E}_P$ and $\mathcal{E}_Q$ in Example 1.1, we find that $\mathcal{E}_P$ and $\mathcal{E}_Q$ are not equivalent in step semantics. The step $\{a, b\}$ is possible in $\mathcal{E}_P$ but not in $\mathcal{E}_Q$. So the example in [36] is not adequate for step semantics. Here we give examples showing that both $\approx_{st}$ and $\approx_{sb}$ are not invariant under refinement of actions, as well as all equivalences included between them, e.g. step failure equivalence.

The following example shows that step trace semantics is in general not invariant under refinement.

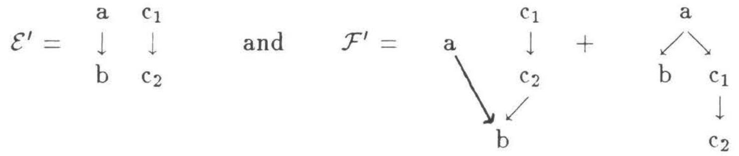
Example 2.1

We consider the two systems



The +-sign in the second system is supposed to indicate that this system either behaves like  or like , that is either performs a and c in parallel and then b, or first a and then b and c in parallel. The +-sign may easily be "implemented" by indicating that all events in the first component are in conflict with all events in the second component and vice versa. (For representing the whole system as a term, we would need to use a sequential composition operator or a TCSP-like parallel composition.)

These two systems are step trace equivalent. However, when refining c into $c_1 \longrightarrow c_2$, the resulting systems

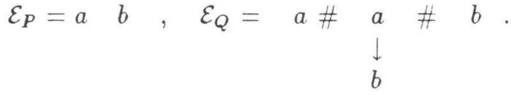


are not step trace equivalent (not even interleaving trace equivalent).

This example shows that $\approx_{st}$ is not preserved by refinement. However, the example is not adequate for step bisimulation equivalence since $\mathcal{E}$ and $\mathcal{F}$ are not step bisimulation equivalent (after performing a, the b is always possible in $\mathcal{E}$ but not always in $\mathcal{F}$). The next example shows that also $\approx_{sb}$ is not preserved by refinement.

Example 2.2

Consider $P = a|b$ and $Q = (a|b) + ab$,



It is easy to verify that $\mathcal{E}_P \approx_{sb} \mathcal{E}_Q$. However, refining a into $a_1 \rightarrow a_2$ yields

$$\begin{tikzcd} \mathcal{E}_{P'} = & a_1 \arrow[d] & b & , & \mathcal{E}_{Q'} = & a_1 \arrow[d] & \# & a_1 \arrow[d] & \# & b \\ & a_2 & & & & a_2 & & a_2 \arrow[d] & & \\ & & & & & & & b & & \end{tikzcd}$$

After the step $\{a_1\}$, the step $\{b\}$ is always possible in $\mathcal{E}_{P'}$. However, in $\mathcal{E}_{Q'}$, it may be the case that the step $\{b\}$ is impossible after executing a_1 (choosing the branch $a_1 \rightarrow a_2 \rightarrow b$). Hence $\mathcal{E}_{P'}$ and $\mathcal{E}_{Q'}$ are not step bisimulation equivalent (not even interleaving bisimulation equivalent).

However, this example is still not suitable for disqualifying the whole range of equivalence notions included between $\approx_{st}$ and $\approx_{sb}$, as the example of [36] does in the interleaving case (see Example 1.1), since the refined systems $\mathcal{E}_{P'}$ and $\mathcal{E}_{Q'}$ turn out to be step trace equivalent. A slightly more complicated example may be given, disqualifying all equivalence notions between $\approx_{sb}$ and $\approx_{st}$.

Example 2.3

First consider the following three systems:

$$\begin{tikzcd} \mathcal{E}_1 = & & \mathrm{a} \arrow[dl] \arrow[dr] & & , & \mathcal{E}_2 = & \mathrm{a} \arrow[dr] & & \mathrm{c} \arrow[dl] & , & \mathcal{E}_3 = & \mathrm{a} \arrow[d] & \mathrm{c} & . \\ & \mathrm{b} & & \mathrm{c} & & & & \mathrm{b} & & & & \mathrm{b} & & \end{tikzcd}$$

Now we consider the two composed systems

$$\mathcal{E} = \mathcal{E}_1 + \mathcal{E}_2, \qquad \mathcal{F} = \mathcal{E}_1 + \mathcal{E}_2 + \mathcal{E}_3.$$

We have $\mathcal{E} \approx_{sb} \mathcal{F}$ [59]. However, when refining c into $c_1 \rightarrow c_2$ only the refinement of $\mathcal{F}$ may perform the sequence of actions $c_1\ a\ b\ c_2$. The resulting systems $\mathcal{E}'$ and $\mathcal{F}'$ are not even interleaving trace equivalent.
So let $\approx$ be an equivalence included between $\approx_{st}$ and $\approx_{sb}$, then also $\mathcal{E} \approx \mathcal{F}$, but $\mathcal{E}' \not\approx \mathcal{F}'$.

Thus we have shown that all the currently known versions of step equivalence are not preserved by refinement.

3 "Linear time" partial order semantics

In [36] it was claimed that equivalence based on considering partially ordered executions is preserved by refinement. In this section we will make this claim more precise. We will show that this is indeed true when considering the set of all possible executions of systems (*traces*), formalising the proof sketch from [36] in terms of event structures. However, in the next section, we will consider equivalence notions taking account of the timing of choices, based on the idea of bisimulation, and we will show that in this case this claim is not so obvious.

In Chapter IV, we discussed that the possible executions of a system may be represented as isomorphism classes of labelled partial orders (*pomsets*), thus taking full account of the causality relation for event occurrences.

Definition

(i) Let $X = (X, \leq_X, l_X)$ and $Y = (Y, \leq_Y, l_Y)$ be partial orders which are labelled over *Act*. X and Y are *isomorphic* ($X \cong Y$) iff there exists a bijection between X and Y respecting the ordering and the labelling. The isomorphism class of a partial order labelled over *Act* is called a *pomset over Act.*

(ii) Let $\mathcal{E}$ be an event structure.
$Pomsets\ (\mathcal{E}) := \{\ [(X, \leq_X, l_{\mathcal{E}} \lceil X)]_{\simeq} \mid X \in Conf(\mathcal{E})\}$.

(iii) Two event structures $\mathcal{E}$ and $\mathcal{F}$ are *pomset trace equivalent*
($\mathcal{E} \approx_{pt} \mathcal{F}$) if $Pomsets\ (\mathcal{E}) = Pomsets\ (\mathcal{F})$.

Clearly, pomset trace equivalence implies step trace equivalence. Example 2.1 shows that pomset trace equivalence is strictly stronger than step trace equivalence. On the other hand, pomset trace equivalence and step bisimulation equivalence (or interleaving bisimulation equivalence) are incomparable: $a(b + c) \substack{\approx_{pt} \\ \not\approx_{sb}} ab + ac$ and for $\mathcal{E}_P$ and $\mathcal{E}_Q$ of Example 2.2, $\mathcal{E}_P \approx_{sb} \mathcal{E}_Q$ but $\mathcal{E}_P \not\approx_{pt} \mathcal{E}_Q$.

Theorem Let $\mathcal{E}, \mathcal{F}$ be flow event structures.

Then $\mathcal{E} \approx_{pt} \mathcal{F}$ implies $ref(\mathcal{E}) \approx_{pt} ref(\mathcal{F})$ for any refinement function *ref*.

Proof
Let $\mathcal{E} \approx_{pt} \mathcal{F}$ and let *ref* be a refinement function. We have to show

$$Pomsets\ (ref\,(\mathcal{E})) = Pomsets\ (ref\,(\mathcal{F})).$$

"$\subseteq$" Let $u \in Pomsets\ (ref\,(\mathcal{E}))$.
Then $u = [(\ \widetilde{X}\ , \leq_{\widetilde{X}}, l_{ref(\mathcal{E})} \lceil\ \widetilde{X}\ )]_{\simeq}$ where $\widetilde{X} \in Conf(ref(\mathcal{E}))$.

With Proposition 1.7 or 2.8 from Chapter IV, we have that $\widetilde{X}$ is a refinement of some configuration X of $\mathcal{E}$. Since *Pomsets* $(\mathcal{E})$ = *Pomsets* $(\mathcal{F})$, there exists $Y \in Conf(\mathcal{F})$ such that $(X, \leq_X, l_\mathcal{E} \lceil X)$ and $(Y, \leq_Y, l_\mathcal{F} \lceil Y)$ are isomorphic. Since isomorphism preserves labelling, we can refine Y to a configuration $\widetilde{Y}$ (by choosing identical refinements for corresponding events) such that

$$(\widetilde{X}, \leq_{\widetilde{X}}, l_{ref(\mathcal{E})} \lceil \widetilde{X}) \cong (\widetilde{Y}, \leq_{\widetilde{Y}}, l_{ref(\mathcal{F})} \lceil \widetilde{Y}),$$

hence $u \in$ *Pomsets* $(ref(\mathcal{F}))$.

" $\supseteq$ " by symmetry. ■

4 "Branching time" partial order semantics

In this section, we discuss several suggestions to define equivalence notions based on partial orders and recording where choices are made. We show that most of these fail in general to be preserved by refinement. Finally we show that the last and strongest notion is indeed invariant with respect to refinement.

4.1 Pomset bisimulation equivalence

In Boudol & Castellani [31] it was suggested to generalise the idea of bisimulation by considering transitions labelled by pomsets. So we consider now transitions $X \xrightarrow{u} X'$ where u is a pomset over Act.

Definition Let $\mathcal{E}$ be an event structure.

$X \xrightarrow{u} X'$ iff $X \rightarrow_\mathcal{E} X'$ and u is the isomorphism class of $\mathcal{E} \lceil (X' \setminus X)$.

Definition Let $\mathcal{E}, \mathcal{F}$ be event structures.

A relation $R \subseteq Conf(\mathcal{E}) \times Conf(\mathcal{F})$ is called a *pomset bisimulation between $\mathcal{E}$ and $\mathcal{F}$* iff $(\emptyset, \emptyset) \in R$ and if $(X, Y) \in R$ then
- $X \xrightarrow{u} X' \Rightarrow \exists Y'$ with $Y \xrightarrow{u} Y'$ and $(X', Y') \in R$,
- $Y \xrightarrow{u} Y' \Rightarrow \exists X'$ with $X \xrightarrow{u} X'$ and $(X', Y') \in R$.

$\mathcal{E}$ and $\mathcal{F}$ are *pomset bisimulation equivalent* ($\mathcal{E} \approx_{pb} \mathcal{F}$) iff there exists a pomset bisimulation between $\mathcal{E}$ and $\mathcal{F}$.

This equivalence notion is clearly stronger than both step bisimulation equivalence and pomset trace equivalence: $\mathcal{E} \approx_{pb} \mathcal{F}$ implies $\mathcal{E} \approx_{sb} \mathcal{F}$ and $\mathcal{E} \approx_{pt} \mathcal{F}$;

moreover, the processes $a|b$ and $(a|b) + ab$ considered in Example 2.2 are *sb*–equivalent but not *pb*–equivalent; $a(b+c)$ and $ab+ac$ are pomset trace equivalent but not *pb*–equivalent.

However, *pb*–equivalence is not preserved by refinement.

Example 4.1

> Consider $a(b+c)+(a|b)$ and $a(b+c)+(a|b)+ab$. We have $P \approx_{pb} Q$. However, when refining a into $a_1 \longrightarrow a_2$ and executing a_1, we may arrive in a situation in the second system where a_2 and b may be only executed sequentially and where c is excluded. This is not possible in the first system.

In VAN GLABBEEK & VAANDRAGER [59], the pomset bisimulation was critisized for violating "the real combination of causality and branching time". The criticism is that only the first system of Example 4.1 has the property that any action a that is causally preceeding b is also preceeding the choice between b and c. Therefore they suggested a generalised pomset bisimulation equivalence, that is finer then pomset bisimulation equivalence, does not identify the two systems of Example 4.1, and still satisfies $a = a + a$ and the absorption law of Section 1.

However, generalised pomset bisimulation equivalence is also not preserved by refinement.

Example 4.2

$$\begin{tikzcd}
 & \mathrm{a} \arrow[d] & & & & \mathrm{a} \arrow[d] & & & \mathrm{a} \arrow[d] \\
\mathcal{E} = & \mathrm{b} & \# & \mathrm{b} & \mathcal{F} = & \mathrm{b} & \# \; \mathrm{b} & + & \mathrm{b}
\end{tikzcd}$$

These two systems are generalised pomset bisimulation equivalent [59]. However, when refining a into $a_1 \longrightarrow a_2$, the resulting systems

$$\begin{tikzcd}
 & a_1 \arrow[d] & & & & & a_1 \arrow[d] & & & & a_1 \arrow[d] \\
\mathcal{E}' = & a_2 \arrow[d] & & & \text{and} & \mathcal{F}' = & a_2 \arrow[d] & & & + & a_2 \arrow[d] \\
 & b & \# & b & & & b & \# & b & & b
\end{tikzcd}$$

are not even interleaving bisimulation equivalent. After the action a_1 the action b is always possible in $\mathcal{E}'$. However in $\mathcal{F}'$ it may be the case that b is impossible after executing a_1 (choosing the branch $a_1 \longrightarrow a_2 \longrightarrow b$).

4.2 History preserving bisimulation

Another equivalence notion based on the idea of bisimulation with partial orders that might be preserved by refinement was suggested independently by Devillers and Van Glabbeek at the workshop on "Combining Compositionality and Concurrency" [101, 48]. It turned out that this notion coincides with the NMS partial ordering equivalence suggested earlier in DEGANO, DE NICOLA & MONTANARI [40]. We rephrase the definition here in terms of event structures as follows.

Definition Let $\mathcal{E}, \mathcal{F}$ be event structures.

> A relation $R \subseteq Conf(\mathcal{E}) \times Conf(\mathcal{F})$ is called a *weak history preserving bisimulation between $\mathcal{E}$ and $\mathcal{F}$* iff $(\emptyset, \emptyset) \in R$ and if $(X, Y) \in R$ then
> - there is an isomorphism between $(X, \leq_X, l_\mathcal{E} \lceil X)$ and $(Y, \leq_Y, l_\mathcal{F} \lceil Y)$,
> - $X \longrightarrow_\mathcal{E} X' \Rightarrow \exists Y'$ with $Y \longrightarrow_\mathcal{F} Y'$ and $(X', Y') \in R$,
> - $Y \longrightarrow_\mathcal{F} Y' \Rightarrow \exists X'$ with $X \longrightarrow_\mathcal{E} X'$ and $(X', Y') \in R$.
>
> $\mathcal{E}$ and $\mathcal{F}$ are *weakly history preserving equivalent* ($\mathcal{E} \approx_{wh} \mathcal{F}$) iff there exists a weak history preserving bisimulation between $\mathcal{E}$ and $\mathcal{F}$.

Note that the isomorphism requirement guarantees that the labels of the events in $X' \setminus X$ and $Y' \setminus Y$ correspond as well.

As observed in [48], it is sufficient to consider only those transitions $X \longrightarrow_\mathcal{E} X'$, (resp. $Y \longrightarrow_\mathcal{F} Y'$) where $X'(Y')$ is obtained from $X(Y)$ by executing exactly one event.

The two systems considered in Example 4.1 are pomset bisimulation equivalent but not weakly history preserving equivalent. However, *wh*-equivalence is not stronger than pomset bisimulation, as shown by the following example; the two notions are in general incomparable. We will show later that *wh*-equivalence does respect pomset bisimulation for systems without autoconcurrency.

The following example will also show that *wh*-equivalence is in general not preserved by refinement. This example was suggested to us by Rabinovich. He used it for showing that $\approx_{wh}$ is not a congruence with respect to a TCSP-like parallel composition.

Example 4.3

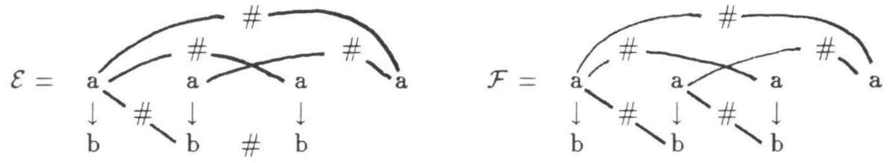
$$\text{Let } \mathcal{E} = \begin{tikzcd} a & \# & a \arrow[d] & & a \arrow[d] \\ & & b & \# & b \end{tikzcd} \quad \text{and } \mathcal{F} = \begin{tikzcd} a & \# & a \arrow[d] \arrow[drr, dash, "\#" description] & & a \arrow[d] \\ & & b & & b \end{tikzcd} .$$

It is straightforward to check that $\mathcal{E} \approx_{wh} \mathcal{F}$. However, $\mathcal{E}$ and $\mathcal{F}$ are not pomset bisimulation equivalent. After executing a, it is alway possible to execute $a \longrightarrow b$ in $\mathcal{E}$, in $\mathcal{F}$ it may be impossible to execute $a \longrightarrow b$ after a. When refining a into $a_1 \longrightarrow a_2$, the resulting systems are no longer wh-equivalent, not even interleaving bisimulation equivalent. This can be proven by providing a formula in Hennessy-Milner logic (Section 1.13 of Chapter I) that is satisfied by the refinement of $\mathcal{F}$, but not by the refinement of $\mathcal{E}$. Such a formula is:

$$a_1\, a_2\, (b\, T \wedge a_1\, \neg b\, T).$$

An equivalence respecting both pomset bisimulation and wh-equivalence may be considered by extending the definition of pomset bisimulation with the requirement that, for any $(X, Y) \in \mathcal{R}$, $\mathcal{E}\lceil X$ and $\mathcal{F}\lceil Y$ should be isomorphic. However, the following example shows that also this equivalence would not preserve refinement.

Example 4.4



As is quite difficult to check, $\mathcal{E}$ and $\mathcal{F}$ are equivalent according to the equivalence notion proposed above, but after refining a into $a_1 \longrightarrow a_2$ they are not even bisimulation equivalent. The formula $a_1\, a_1\, \neg a_2\, \neg b\, T$ is satisfied by $\mathcal{E}$, but not by $\mathcal{F}$.

We finally define a stronger version of history perserving equivalence which will respect pomset bisimulation. This notion was first suggested by Trakhtenbrot, Rabinovich & Hirshfeld in terms of behaviour structures (see [109]). We will show that this equivalence is preserved by refinement. For systems without autoconcurrency, this equivalence coincides with $\approx_{wh}$. This will imply the result that $\approx_{wh}$ is invariant against refinement for systems without autoconcurrency.

Definition Let $\mathcal{E}, \mathcal{F}$ be event structures.

A relation $R \subseteq Conf(\mathcal{E}) \times Conf(\mathcal{F}) \times \mathcal{P}(E_{\mathcal{E}} \times E_{\mathcal{F}})$ is called a *history preserving bisimulation* between $\mathcal{E}$ and $\mathcal{F}$ if $(\emptyset, \emptyset, \emptyset) \in R$ and whenever $(X, Y, f) \in R$ then

- f is an isomorphism between $(X, \leq_X, l_{\mathcal{E}}\lceil X)$ and $(Y, \leq_Y, l_{\mathcal{E}}\lceil Y)$,

- $X \rightarrow_{\mathcal{E}} X' \Rightarrow \exists Y', f'$ with $Y \rightarrow_{\mathcal{F}} Y'$, $(X', Y', f') \in R$ and $f' \lceil X = f$,
- $Y \rightarrow_{\mathcal{F}} Y' \Rightarrow \exists X', f'$ with $X \rightarrow_{\mathcal{E}} X'$, $(X', Y', f') \in R$ and $f' \lceil X = f$.

$\mathcal{E}$ and $\mathcal{F}$ are *history preserving equivalent* ($\mathcal{E} \approx_h \mathcal{F}$) iff there exists a history preserving bisimulation between $\mathcal{E}$ and $\mathcal{F}$.

Clearly, we have $\mathcal{E} \approx_h \mathcal{F} \Rightarrow \mathcal{E} \approx_{wh} \mathcal{F}$. However the two systems of Example 4.3 are not h–equivalent.

Proposition $\mathcal{E} \approx_h \mathcal{F} \Rightarrow \mathcal{E} \approx_{pb} \mathcal{F}$.

Proof
We show that any history preserving bisimulation between $\mathcal{E}$ and $\mathcal{F}$ is also a pomset bisimulation between $\mathcal{E}$ and $\mathcal{F}$ (after leaving out the isomorphism component). Let R be a h-bisimulation, and suppose $(X, Y, f) \in R$ and $X \xrightarrow{u} X'$. Then $X \rightarrow_{\mathcal{E}} X'$, thus $\exists Y', f'$ with $Y \rightarrow_{\mathcal{F}} Y', (X', Y', f') \in R$ and $f' \lceil X = f$. Since f' is an isomorphism and $f' \lceil X = f$, *range* $(f' \lceil (X' \setminus X)) =$ *range* $(f') \setminus$ *range* $(f) = Y' \setminus Y$, so $f' \lceil (X' \setminus X)$ is an isomorphism between $X' \setminus X$ and $Y' \setminus Y$. Hence $Y \xrightarrow{u} Y'$, so R satisfies the first clause of a pomset bisimulation. The second clause follows by symmetry. ■

From this proof we learn that h-bisimulation not only respects pomset bisimulation but even the previous proposal combining weak history preserving equivalence and pomset bisimulation. Thus $\approx_h$ is the strongest equivalence considered so far (except for event structure isomorphism of course). Nevertheless it is possible to abstract from certain details in a system representation: we have $a \approx_h a + a$ and $(a|(b+c)) + (a|b) + ((a+c)|b) \approx_h (a|(b+c)) + ((a+c)|b)$ (absorption law).

We now show that considering only those transitions $X \rightarrow_{\mathcal{E}} X', Y \rightarrow_{\mathcal{F}} Y'$, respectively, where $X'(Y')$ is obtained from $X(Y)$ by executing exactly one event yields the same equivalence. We write $X \triangleright_{\mathcal{E}} X'$ for $X \rightarrow_{\mathcal{E}} X'$ and $|X' \setminus X| = 1$. Let $\approx_{oh}$ be the equivalence notion obtained by replacing $\rightarrow$ by $\triangleright$ in the definition of $\approx_h$.

Proposition For event structures $\mathcal{E}, \mathcal{F}$: $\mathcal{E} \approx_h \mathcal{F}$ iff $\mathcal{E} \approx_{oh} \mathcal{F}$.

Proof
The implication $\mathcal{E} \approx_h \mathcal{F} \Rightarrow \mathcal{E} \approx_{oh} \mathcal{F}$ is trivial. The implication $\mathcal{E} \approx_{oh} \mathcal{F} \Rightarrow \mathcal{E} \approx_h \mathcal{F}$ immediately follows from the observation that whenever $X \rightarrow_{\mathcal{E}} X'$, there exist configurations $X_1, \ldots, X_n (n \in I\!N)$ such that $X = X_1 \triangleright_{\mathcal{E}} \ldots \triangleright_{\mathcal{E}} X_n = X'$. ■

Next we show that $\approx_h$ is preserved by refinement.

Theorem Let $\mathcal{E}, \mathcal{F} \in \mathbb{E}$ and let *ref* be a refinement function.

Then $\mathcal{E} \approx_h \mathcal{F} \Longrightarrow ref(\mathcal{E}) \approx_h ref(\mathcal{F})$.

Proof

Let $R \subseteq Conf(\mathcal{E}) \times Conf(\mathcal{F}) \times \mathcal{P}(E_{\mathcal{E}} \times E_{\mathcal{F}})$ be a history preserving bisimulation between $\mathcal{E}$ and $\mathcal{F}$. Define the relation $\tilde{R}$ by:

$$\tilde{R} = \{(\tilde{X}, \tilde{Y}, \tilde{f}) \in Conf\,(ref\,(\mathcal{E})) \times\ Conf\,(ref\,(\mathcal{F})) \times \mathcal{P}(E_{ref(\mathcal{E})} \times E_{ref(\mathcal{F})})| \\ \exists (X, Y, f) \in R \text{ such that}$$

- $\tilde{X}$ is a refinement of X,
- $\tilde{Y}$ is a refinement of Y
- and $\tilde{f}{:}\tilde{X} \longrightarrow \tilde{Y}$ is a bijection, satisfying $\tilde{f}\,(e, e') = (f(e), e')\}$.

We show that $\tilde{R}$ is a history preserving bisimulation between $ref\,(\mathcal{E})$ and $ref\,(\mathcal{F})$.

i. $(\emptyset, \emptyset, \emptyset) \in \tilde{R}$ since $(\emptyset, \emptyset, \emptyset) \in R$.

ii. Suppose $(\tilde{X}, \tilde{Y}, \tilde{f}) \in \tilde{R}$. Take $(X, Y, f) \in R$ such that
- $\tilde{X}$ is a refinement of X,
- $\tilde{Y}$ is a refinement of Y
- and $\tilde{f}{:}\tilde{X} \longrightarrow \tilde{Y}$ is bijection, satisfying $\tilde{f}\,(e, e') = (f(e), e')$.

Now three things have to be established:

1. $\tilde{f}$ satisfies $(d, d') \leq_{\tilde{X}} (e, e') \Longleftrightarrow \tilde{f}\,(d, d') \leq_{\tilde{Y}} \tilde{f}\,(e, e')$ and $l_{ref(\mathcal{F})}(\tilde{f}\,(e, e')) = l_{ref(\mathcal{E})}(e, e')$.
2. $\tilde{X} \longrightarrow_{ref(\mathcal{E})} \tilde{X}' \Longrightarrow \exists\, \tilde{Y}', \tilde{f}'$ such that $\tilde{Y} \longrightarrow_{ref(\mathcal{F})} \tilde{Y}', \tilde{f}' \lceil \tilde{X} = \tilde{f}$ and $(\tilde{X}', \tilde{Y}', \tilde{f}') \in \tilde{R}$.
3. $\tilde{Y} \longrightarrow_{ref(\mathcal{F})} \tilde{Y}' \Longrightarrow \exists\, \tilde{X}', \tilde{f}'$ such that $\tilde{X} \longrightarrow_{ref(\mathcal{E})} \tilde{X}', \tilde{f}' \lceil \tilde{X} = \tilde{f}$ and $(\tilde{X}', \tilde{Y}', f') \in \tilde{R}$.

ad 1. Straightforward.
ad 2.
Suppose $\tilde{X} \longrightarrow_{ref(\mathcal{E})} \tilde{X}'$, i.e. $\tilde{X}' \in Conf\,(ref\,(\mathcal{E}))$ and $\tilde{X} \subseteq \tilde{X}'$.
We have $\tilde{X}' = \bigcup_{e \in X'} \{e\} \times X'_e$ where $X' \in Conf(\mathcal{E})$ and

$\forall e \in X' : X'_e \in \mathit{Conf}\ (\mathit{ref}\,(l_{\mathcal{E}}(e))) - \{\emptyset\}$.
Then $X = pr_1(\ \widetilde{X}\ )$ and $X' = pr_1(\ \widetilde{X'}\ )$, so $X \longrightarrow_{\mathcal{E}} X'$.

Since R is a history preserving bisimulation,
$\exists Y', f'$ with $Y \longrightarrow_{\mathcal{F}} Y', f' \lceil X = f$ and $(X', Y', f') \in R$.
Let $\widetilde{Y'}\ = \{(f'(e), e') | (e, e') \in\ \widetilde{X}\ \}$
and $\widetilde{f'}\ = \{((e, e'), (f'(e), e')) | (e, e') \in\ \widetilde{X'}\ \}$.
It now suffices to show that $\widetilde{Y'}$ is a refinement of Y', since then it follows immediately with Proposition 1.7 or 2.8 from Chapter IV that
$\widetilde{Y'}\ \in \mathit{Conf}\,(\mathit{ref}\,(\mathcal{F}))$, $\widetilde{Y}\ \longrightarrow_{\mathit{ref}(\mathcal{F})}\ \widetilde{Y'}$ (using that $f' \lceil X = f$), $\widetilde{f'}\ \lceil X =\ \widetilde{f}$ (likewise) and $(\ \widetilde{X'}\ ,\ \widetilde{Y'}\ ,\ \widetilde{f'}\ ) \in\ \widetilde{R}$.

- By construction $\widetilde{Y'}\ = \bigcup_{e \in X'} \{f'(e)\} \times X'_e = \bigcup_{e \in Y'} \{e\} \times Y'_e$ where
  $\forall e \in Y' : Y'_e = X'_{f'^{-1}(e)} \in \mathit{Conf}\,(\mathit{ref}\,(l_{\mathcal{E}}(f'^{-1}(e)))) - \{\emptyset\} =$
  $\mathit{Conf}\,(\mathit{ref}\,(l_{\mathcal{F}}(e))) - \{\emptyset\}$.

- $e \in \mathit{busy}(\widetilde{Y'}) = \{e \in Y' | Y'_e$ not complete $\} \Longleftrightarrow$
  $f'^{-1}(e) \in \mathit{busy}(\widetilde{X'}) = \{e \in X' | X'_e$ not complete $\}$ by construction.
  Furthermore, e maximal in $Y' \Longleftrightarrow f'^{-1}(e)$ maximal in X', since f' is an isomorphism.

  Hence $e \in \mathit{busy}(\widetilde{Y'})$ implies e maximal in Y', since $\widetilde{X'}$ is a refinement of X'.

From this it follows that $\widetilde{Y'}$ is a refinement of Y'.

ad 3. By symmetry. ■

Finally we show that $\approx_{wh}$ and $\approx_h$ coincide for event structures where concurrent events may not carry the same label. As a corrollary we then have that also $\approx_{wh}$ is preserved by refinement in this case and respects pomset bisimulation.

Definition $\mathcal{E}$ is an event structure *without autoconcurrency* iff
$\forall d, e \in E_{\mathcal{E}} : d\ co\ e$ and $l(d) = l(e) \Rightarrow d = e$.

Theorem For event structures $\mathcal{E}, \mathcal{F}$ without autoconcurrency,
$\mathcal{E} \approx_{wh} \mathcal{F} \Leftrightarrow \mathcal{E} \approx_h \mathcal{F}$.

Proof
First note that a wh-bisimulation can be regarded as a h-bisimulation without

the requirements that $f'\lceil X = f$.
Now "$\Leftarrow$" is trivial.
In order to establish "$\Rightarrow$" we will show that any wh-bisimulation between event structures $\mathcal{E}$, $\mathcal{F}$ without autoconcurrency is also a h-bisimulation. So let r be a h-bisimulation between such $\mathcal{E}$ and $\mathcal{F}$, without the requirements $f'\lceil X = f$. We proof that these requirements are met nevertheless. Assume that $(X, Y, f) \in R$ and $X \rightarrow_{\mathcal{E}} X'$. Then there exists $(X', Y', f') \in R$ with $Y \rightarrow_F Y'$. Suppose $f'\lceil X \neq f$. Then there exists an $e \in X$ with $f'(e) \neq f(e)$.

Now observe that if g is an isomorphism between two arbitrary partial orders $(X, \leq_X, l_X)$ and $(Y, \leq_Y, l_Y)$, and $g(e_1) = e_2$ then

$$|\{e' \in X | e' \leq_X e_1\}| = |\{e' \in Y | e' \leq_Y e_2\}|.$$

Hence we cannot have $f'(e) < f(e)$ or $f(e) < f'(e)$.
Since Y' is conflict-free we conclude $f'(e)\ co\ f(e)$.
Moreover, f' and f preserve labelling, so $l_{\mathcal{F}}(f'(e)) = l_{\mathcal{E}}(e) = l_{\mathcal{F}}(f(e))$.
This is a contradiction since $\mathcal{F}$ was assumed to have no autoconcurrency. ■

Corollary Let $\mathcal{E}, \mathcal{F}$ be event structures without autoconcurrency and

let *ref* be a refinement.
Then $\mathcal{E} \approx_{wh} \mathcal{F} \Rightarrow ref(\mathcal{E}) \approx_{wh} ref(\mathcal{F})$.

Conclusion

In this chapter we have shown that equivalences based on interleaving of atomic actions or of steps (multisets of concurrently executable actions) are not preserved when changing the level of atomicity of actions. However, we could show that certain equivalences based on modelling causal relations explicitly by partial orders are indeed preserved by refinement of actions. We considered "linear time" approaches, where the behaviour of a system is equated to the set of possible runs, and "branching time" approaches, where the conflict structure of systems is taken into account. We could show the negative results about the interleaving approaches regardless of the level of detail in modelling the conflict behaviour. However, for the positive results about the partial order approaches, the conflict structure turned out to be crucial. An interesting topic for further research would be to investigate testing equivalences based on partial orders, taking the conflict structure in a weaker form into account. For an overview consider the following diagram:

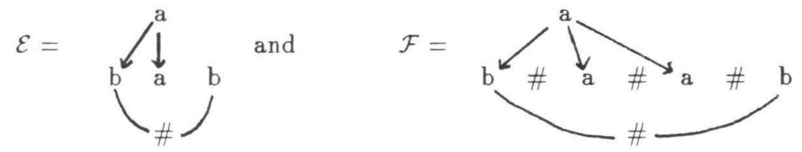
runs conflict structure	sequences of actions	sequences of steps	pomsets
paths	▨ $\approx_{it}$	▨ $\approx_{st}$	$\approx_{pt}$
⋮ e.g. testing	▨	▨	?
bisimulation	▨ $\approx_{ib}$	▨ $\approx_{sb}$	▨ $\approx_{pb}$ $\approx_{wh}$ comb. of $\approx_{pb}$ and $\approx_{wh}$
			$\approx_h$

▨ means: not preserved by refinement

This diagram is not at all complete. A naturally arising question is to what extent it is actually necessary to move to partial orders to achieve invariance of equivalence under refinement (here we have only shown that steps are not sufficient). This question will be addressed in the last chapter of this thesis. Another equivalence being preserved by refinement was proposed by HENNESSY [71,3], however it is defined on a syntactical level and is not applicable to such a wide class of systems as considered here, e.g. it is not possible to treat full CCS.

The refinement operation we have considered replaced actions by arbitrary non-empty event structures. As remarked in Chapter IV, it is debatable whether one should consider refinements where replacing actions by the empty event structure is allowed (*forgetful refinements*). Such refinements can drastically change the structure of processes, they can not be explained by a change in the level of abstraction at which processes are regarded. Nevertheless, our results hold also for forgetful refinements (with slightly more complicated proofs).

Finally we would like to address the question whether history preserving bisimulation as defined here is the coarsest equivalence respecting pomset bisimulation and being preserved by refinement. We conjecture that this is not the case, in particular, that for



$\mathcal{E} \not\approx_h \mathcal{F}$, but for any refinement *ref*, $ref(\mathcal{E}) \approx_{pb} ref(\mathcal{F})$.

Nevertheless, if it is required to model the interplay of causality and branching in full detail, history preserving bisimulation seems to be the coarsest suitable equivalence.

Chapter VII

The Refinement Theorem for ST-bisimulation Semantics

R.J. van Glabbeek

In this chapter I prove that ST-bisimulation equivalence, as introduced in [58], is preserved under refinement of actions. This implies that it is possible to abstract from the causal structure of concurrent systems without assuming action atomicity.

Note: This chapter appeared as Report CS-R9002, CWI, Amsterdam 1990, and will be published in: Proceedings IFIP Working Conference on Programming Concepts and Methods, Sea of Gallilee, Israel 1990 (M. Broy & C.B. Jones, eds.), North-Holland 1990.

TABLE OF CONTENTS

INTRODUCTION

Virtually all semantic equivalences employed in theories of concurrency are defined in terms of *actions* that concurrent systems may perform (cf [1-18]). Mostly, these actions are taken to be *atomic*, meaning that they are considered not to be divisible into smaller parts. In this case, the defined equivalences are said to be based on *action atomicity*.

However, in the top-down design of distributed systems it might be fruitful to model processes at different levels of abstraction. The actions on an abstract level then turn out to represent complex processes on a more concrete level. This methodology does not seem to be compatible with non-divisibility of actions and for this reason PRATT [107], LAMPORT [82] and others plead for the use of semantic equivalences that are not based on action atomicity.

As indicated in CASTELLANO, DE MICHELIS & POMELLO [36], the concept of action atomicity can be formalized by means of the notion of *refinement of actions*. A semantic equivalence is *preserved under action refinement* if two equivalent processes remain equivalent after replacing all occurrences of an action a by a more complicated process $r(a)$. In particular, $r(a)$ may be a sequence of two actions a_1 and a_2. An equivalence is strictly based on action atomicity if it is not preserved under refinement.

Most semantic equivalences can be positioned in a two dimensional classification diagram, such as the one of Figure 1. On the x-axis equivalences are ordered with respect to the preserved level of detail of runs of processes. Three well-known points on this axis are *interleaving semantics*, where runs are represented by sequences of action occurrences, *step semantics*, where runs are represented by sequences of multisets of action occurrences - the multisets (or *steps*) representing simultaneous occurrences - and *partial order semantics*, in which all causal dependencies between action occurrences in runs of processes are preserved. On the y-axis the equivalences are ordered with respect to the preserved level of detail of the branching structure of these runs. Two well-known points on this axis are *trace semantics*, where a process is fully determined by the set of its possible (partial) runs, thereby completely neglecting the branching structure of processes, and *bisimulation semantics*, where also the information is preserved where two different courses of action diverge (although branching of identical courses of action is still neglected). In between there are several *decorated trace semantics*, where part of the branching structure is taken into account. Mostly these are motivated by the observable behaviour of processes, according to some testing scenario. In Figure 1 the equivalences become finer, or more discriminating, when moving upwards or to the right.

$$\begin{tikzcd}
\textit{bisimulation semantics} & \approx_{ib} \arrow[dddd] & \approx_{sb} \arrow[l] \arrow[dddd] & \approx_{h} \arrow[l] \arrow[dddd] \\
\vdots & & & \\
\textit{decorated trace semantics} & & & \\
\vdots & & & \\
\textit{trace semantics} & \approx_{it} & \approx_{st} \arrow[l] & \approx_{pt} \arrow[l] \\
 & \textit{interleaving semantics} & \textit{step semantics} & \textit{partial order semantics}
\end{tikzcd}$$

FIGURE 1. *Semantic equivalences*

In [36], CASTELLANO, DE MICHELIS & POMELLO show by means of a simple example that none of the interleaving equivalences - not even bisimulation - is preserved under action refinement. Furthermore they claim that 'on the other hand, the approaches based on partial order are not constrained to the assumption of atomicity'. Therefore they conclude that 'interleaving semantics is adequate only if the abstraction level at which the atomic actions are defined is fixed. Otherwise, partial order semantics should be considered'.

In [54] (the previous chapter of this thesis), URSULA GOLTZ & I elaborated on this argument by providing examples, showing that also none of the step equivalences is preserved under refinement, and by formalizing the proof sketch of [36] that trace equivalence based on partial orders *is* invariant under refinement. We also wanted to prove this for bisimulation equivalence based on partial orders, but surprisingly we found that none of the partial order bisimulation equivalences proposed before publication of [36] is preserved under action refinement. However, we *did* prove a refinement theorem for a new notion of bisimulation equivalence based on partial orders, proposed recently by Hirshfeld, RABINOVICH & TRAKHTENBROT [108]. We chose to call this equivalence *history preserving bisimulation equivalence*, notation $\approx_h$. Hence, even in bisimulation semantics, the requirements of preservation under action refinement and capturing causal dependencies in processes by means of partial orders can be conciliated. But of course, this still does not show that in case preservation under refinement is required, it is *necessary* to employ partial order semantics. In this chapter I will show that it is not.

Event structures and Petri nets have been established as suitable domains for modelling (both branching and causal aspects of) concurrent systems. Usually a state of a concurrent system is represented by a *configuration* of the associated event structure, or by a *marking* of the associated net. In this chapter I argue that when events or transitions are considered to have a duration or structure, configurations or markings do not properly represent all the states of concurrent systems. Instead I propose to use so-called *ST-configurations* or *ST-markings*. The idea to model a state in a safe labelled marked net as the set of places (*Stellen*) containing a token, together with the set of transitions (*Transitionen*) which are currently firing (an *ST-marking*) originates from VAN GLABBEEK & VAANDRAGER [58]. In this chapter I translate this idea to the realm of event structures by introducing *ST-configurations*.

All interleaving, step and partial order equivalences on event structures or Petri nets considered so far, have been defined in terms of configurations or markings. If the constructions from interleaving semantics are applied on ST-configurations instead of ordinary configurations two new points on the x-axis of Figure 1 emerge. *Split-semantics* is just interleaving semantics, but based on interleaving of beginnings and ends of events, instead of entire events; *ST-semantics* is a refinement of split semantics where in addition a link is required between the beginning and the end of any event. Split semantics is more discriminating than step semantics, whereas ST-semantics is as least as discriminating as split semantics. Furthermore ST-trace semantics is less discriminating than trace semantics based on partial orders and ST-bisimulation semantics is less discriminating than history preserving bisimulation semantics (but incomparable with the other bisimulation semantics based on partial orders proposed so far). Hence the situation is as indicated in Figure 2.

$$\begin{tikzcd}
\approx_{ib} \arrow[d] & \approx_{sb} \arrow[l] \arrow[d] & \approx_{2b} \arrow[l] \arrow[d] & \approx_{STb} \arrow[l] \arrow[d] & \approx_{h} \arrow[l] \arrow[d] \\
\approx_{it} & \approx_{st} \arrow[l] & \approx_{2t} \arrow[l] & \approx_{STt} \arrow[l] & \approx_{pt} \arrow[l] \\
\textit{interleaving semantics} & \textit{step semantics} & \textit{split semantics} & \textit{ST-semantics} & \textit{partial order semantics}
\end{tikzcd}$$

FIGURE 2. *More semantic equivalences*

ST-bisimulation equivalence was introduced by FRITS VAANDRAGER & ME in [58]. In the same paper we observed that for systems without autoconcurrency ST-bisimulation equivalence coincides with split bisimulation equivalence and provided a complete axiomatization on closed ACP-terms for the latter notion. Split bisimulation equivalence was proposed in HENNESSY [70] for a subset of CCS. ACETO & HENNESSY [3] proved that on this subset split bisimulation equivalence is preserved under action refinement. HENNESSY [70] also provided a complete axiomatization for split bisimulation equivalence on this subset. Since - if one forgets about τ-moves - this proof system is sound for ST-bisimulation equivalence, and even for history preserving bisimulation equivalence, it follows that on the domain considered in [70] the three equivalences coincide. In combination with the refinement theorem for history preserving bisimulation equivalence of the previous chapter this yields an alternative proof of Aceto & Hennessy's refinement theorem. Split trace equivalence has been considered in VAANDRAGER [119]. In a joint paper with FRITS VAANDRAGER [60] we will show that on the domain of labelled event structures (prime event structures with binary conflict), or on full CCS, split semantics is not proof against refinement. In fact the equivalences obtained by splitting an event into two parts (its beginning and its end) turned out to be different from the equivalences obtainable by splitting an event into three parts. This was established by means of a rather complicated example (the *owl* example), that also shows that split semantics is strictly less discriminating than ST-semantics. By means of even more complicated examples we established that for each $n \in \mathbb{N}$ split-n semantics is also different from split-n + 1 semantics.

The result contributed by the present chapter is that ST-bisimulation semantics as well as ST-trace semantics are preserved under action refinement. In

[58] it was shown that these semantics do not respect causality. It follows that it is possible to abstract from the causal structure of concurrent systems without assuming action atomicity.

1. Concurrent systems and refinement of actions

Also in this chapter I consider systems that are capable of performing actions from a given set *Act* of action names. Following [54], as my model for this kind of systems I have chosen labelled prime event structures with a binary conflict relation as introduced in Nielsen, Plotkin & Winskel [99]); I could have chosen other models like Petri nets or behaviour structures [108] as well. In this chapter I will not distinguish external and internal actions; I do not consider abstraction by hiding of actions.

Definition. A *(labelled) event structure* (over an alphabet *Act*) is a 4-tuple $\mathcal{E}=(E, <, \sharp, l)$, where

- E is a set of *events*;
- $<\subseteq E\times E$ is a partial order (the *causality relation*) satisfying the principle of *finite causes*:

  $\{e'\in E \mid e'<e\}$ is finite for $e\in E$;

- $\sharp\subseteq E\times E$ is an irreflexive, symmetric relation (the *conflict relation*) satisfying the principle of *conflict heredity*:

  $e_1\sharp e_2<e_3 \Rightarrow e_1\sharp e_3$;

- l: $E\rightarrow Act$ is a *labelling function*.

An event structure represents a concurrent system in the following way: action names $a\in Act$ represent actions the system may perform, an event $e\in E$ labelled with a represents an occurrence of a during a possible run of the system, $e'<e$ means that e' is a prerequisite for e and $e'\sharp e$ means that e' and e cannot happen both in the same run.

One usually writes $e'\leqslant e$ for $e'<e \vee e'=e$, $>$ for $<^{-1}$ and $\geqslant$ for $\leqslant^{-1}$. Causal independence (*concurrency*) of events is expressed by the derived relation $\smile\subseteq E\times E$ defined by: $e'\smile e$ iff $\neg(e'\sharp e \vee e'<e \vee e'>e \vee e'=e)$. By definition $<$, $=$, $>$, $\sharp$ and $\smile$ form a partition of $E\times E$. The concurrency relation $co\subseteq E\times E$, originating from Petri net theory, is defined slightly different from $\smile$: $e'\ co\ e$ iff $e'\smile e \vee e'=e$.

The components of an event structure $\mathcal{E}$ will be denoted by respectively $E_{\mathcal{E}}$, $<_{\mathcal{E}}$, $\sharp_{\mathcal{E}}$ and $l_{\mathcal{E}}$. The derived relations will be denoted $\smile_{\mathcal{E}}$, $co_{\mathcal{E}}$, $\leqslant_{\mathcal{E}}$, $>_{\mathcal{E}}$ and $\geqslant_{\mathcal{E}}$.

Throughout the chapter, I assume a fixed set *Act* of action names as labelling set. Let $\mathbb{E}$ denote the domain of event structures labelled over *Act*.

DEFINITION. An *event structure isomorphism* between two event structures $\mathcal{E},\mathcal{F}\in\mathbb{E}$ is a bijective mapping $f: E_{\mathcal{E}}\rightarrow E_{\mathcal{F}}$ such that

- $f(e) <_{\mathcal{F}} f(e') \Leftrightarrow e <_{\mathcal{E}} e'$,
- $f(e) \#_{\mathcal{F}} f(e') \Leftrightarrow e \#_{\mathcal{E}} e'$ and
- $l_{\mathcal{F}}(f(e)) = l_{\mathcal{E}}(e)$.

$\mathcal{E}$ and $\mathcal{F}$ are *isomorphic* - notation $\mathcal{E}\cong\mathcal{F}$ - if there exists an event structure isomorphism between them. Generally, one does not distinguish isomorphic event structures.

DEFINITION. The *restriction* of an event structure $\mathcal{E}$ to a set $X\subseteq E_{\mathcal{E}}$ of events is the event structure $\mathcal{E}\upharpoonright X=(X, <_{\mathcal{E}}\cap(X\times X), \#_{\mathcal{E}}\cap(X\times X), l_{\mathcal{E}}\upharpoonright X)$.

An event structure $\mathcal{E}$ is *finite* if $E_{\mathcal{E}}$ is finite; $\mathcal{E}$ is *conflict-free* if $\#_{\mathcal{E}}=\varnothing$.
0 denotes the *empty* event structure $(\varnothing, \varnothing, \varnothing, \varnothing)$.

In [36] it is shown that equivalence notions based on interleaving are not preserved when replacing an action in a system by a sequence of two actions. In Section 1 of Chapter IV we considered a more general version of this operation, which I will also use in the present chapter: replacing actions by finite, conflict-free, non-empty event structures. Replacing actions by infinite event structures could in general invalidate the principle of finite causes. As explained in Chapter IV, replacing actions by event structures containing conflicts would require a more sophisticated notion of refinement or, alternatively, a more general form of event structures where the axiom of conflict heredity is dropped, e.g. flow event structures as in Section 2 of Chapter IV. The generalization of the results of this chapter to flow event structures seems to be completely straightforward, but has still to be carried out. Finally, replacing actions by the empty event structure can drastically change the structure of processes; it can not be explained by a change in the level of abstraction at which processes are regarded (Chapter IV). In the concluding section I will discuss possible extensions of my result to these cases.

A refinement will be a function r specifying for each action a an event structure $r(a)$ which is to be substituted for a. Interesting refinements will mostly refine only certain actions, hence replace most actions by themselves. However, for uniformity (and for simplicity in proofs) I consider all actions to be refined.

Given an event structure $\mathcal{E}$ and a refinement r, the refined event structure $r(\mathcal{E})$ is constructed as follows. Each event e labelled by a is replaced by a disjoint copy, $r(e)$, of $r(a)$. The causality and conflict structure is inherited from $\mathcal{E}$: every event which was causally before e will be causally before all events of $r(e)$, all events which causally followed e will causally follow all the events of $r(e)$, and all events in conflict with e will be in conflict with all the events of $r(e)$.

DEFINITION. A *refinement* $r:Act\rightarrow\mathbb{E}-\{0\}$ is a function that takes any action $a\in Act$ into a finite, conflict-free, non-empty event structure $r(a)\in\mathbb{E}$. If $\mathcal{E}\in\mathbb{E}$

and r is a refinement, then $r(\mathcal{E})$ is the event structure defined by:

- $E_{r(\mathcal{E})} = \{(e,e') \mid e \in E_{\mathcal{E}},\ e' \in E_{r(l_{\mathcal{E}}(e))}\}$;
- $(e_1,e_1') <_{r(\mathcal{E})} (e_2,e_2')$ iff $e_1 <_{\mathcal{E}} e_2$ or $(e_1 = e_2 \wedge e_1' <_{r(l_{\mathcal{E}}(e_1))} e_2')$;
- $(e_1,e_1') \#_{r(\mathcal{E})} (e_2,e_2')$ iff $e_1 \#_{\mathcal{E}} e_2$;
- $l_{r(\mathcal{E})}(e,e') = l_{r(l_{\mathcal{E}}(e))}(e')$.

Proposition 1:

i. *If $\mathcal{E} \in \mathbb{E}$ and r is a refinement then $r(\mathcal{E})$ is an event structure indeed.*

ii. *If $\mathcal{E} \in \mathbb{E}$ and r,r' are refinements with $r(a) \cong r'(a)$ for $a \in Act$, then $r(\mathcal{E}) \cong r'(\mathcal{E})$.*

iii. *If $\mathcal{E},\mathcal{F} \in \mathbb{E}$, r is a refinement and $\mathcal{E} \cong \mathcal{F}$, then $r(\mathcal{E}) \cong r(\mathcal{F})$.*

Proof: Straightforward. □

This proposition says that refinement is a well-defined operation on event structures, even when isomorphic event structures are identified.

2. The behaviour of concurrent systems I

Let $\mathcal{E}$ be an event structure, modelling the behaviour of a concurrent system P. Classically, a state S of P is given by a set of events from $\mathcal{E}$. Such a set is called a *configuration*. Its elements represent the occurrences of actions that happened before P reached the state S. If two events e' and e cannot happen both in the same run ($e' \#_{\mathcal{E}} e$) then they also cannot occur in the same configuration. So configurations have to be *conflict-free*. Furthermore, if e occurs in a configuration C and e' is a prerequisite for e ($e' <_{\mathcal{E}} e$) then also e' must occur in C. Hence configurations must be *left-closed* with respect to $<_{\mathcal{E}}$. Finally, as is usual, in this thesis it is assumed that in a finite period only finitely many actions are performed. Therefore, unlike in many other papers, configurations are required to be finite here.

Definition. A set $X \subseteq E_{\mathcal{E}}$ of events in an event structure $\mathcal{E}$ is *left-closed* in $\mathcal{E}$ if for all $e,e' \in E_{\mathcal{E}}$

$$e' <_{\mathcal{E}} e \in X \Rightarrow e' \in X.$$

X is *conflict-free* in $\mathcal{E}$ if $\mathcal{E} \upharpoonright X$ is conflict-free. A *configuration* of $\mathcal{E}$ is a finite, left-closed, conflict-free subset of $E_{\mathcal{E}}$. Let $\mathcal{C}(\mathcal{E})$ be the set of configurations of $\mathcal{E}$. Write $X \longrightarrow_{\mathcal{E}} X'$ if $X,X' \in \mathcal{C}(\mathcal{E})$ and $X \subseteq X'$.

$X \longrightarrow_{\mathcal{E}} X'$ says that both X and X' represent states of the concurrent system represented by $\mathcal{E}$, and that this system may evolve from the state represented by X to the one represented by X'.

As the lemma below will show, the behaviour of a refined event structure $r(\mathcal{E})$ may be deduced from the behaviour of $\mathcal{E}$ and from the behaviour of the event structures which are substituted for actions. On the other hand, one may derive information about the behaviour of $\mathcal{E}$ from the behaviour of $r(\mathcal{E})$.

Let $r(e)$ abbreviate $r(l_{\mathcal{E}}(e))$ and let pr_1 denote projection to the first component.

Lemma 2: *Let $\mathcal{E}$ be an event structure and r a refinement.*

i. $\tilde{C} \subseteq E_{r(\mathcal{E})}$ *is a configuration of* $r(\mathcal{E})$ *iff*

$\tilde{C} = \{(e,e') \mid e \in C,\ e' \in C_e\}$ *where*

C *is a configuration of* $\mathcal{E}$,
C_e *is a configuration of* $r(e)$ *for* $e \in C$,
$C_e = E_{r(e)}$ *if e is not maximal in C with respect to* $<_{\mathcal{E}}$.

ii. *If* $\tilde{C} \longrightarrow_{r(\mathcal{E})} \tilde{C}'$ *then* $pr_1(\tilde{C}) \longrightarrow_{\mathcal{E}} pr_1(\tilde{C}')$.

Proof: See [54]. A similar lemma has been proved in the Chapter IV. □

3. Equivalence notions for concurrent systems I

In this section the semantic equivalences of Figure 1 are defined in terms of configurations.

The interleaving equivalences can be defined by means of the *single action transition relations* $\xrightarrow{a}_{\mathcal{E}} \subseteq \mathcal{C}(\mathcal{E}) \times \mathcal{C}(\mathcal{E})$ for $a \in Act$ and $\mathcal{E} \in \mathbb{E}$.

Definition. $C \xrightarrow{a}_{\mathcal{E}} C'$ iff $C \longrightarrow_{\mathcal{E}} C'$ and $C' - C = \{e\}$ with $l_{\mathcal{E}}(e) = a$.

Here $C \xrightarrow{a}_{\mathcal{E}} C'$ says that if the system represented by $\mathcal{E}$ is in the state represented by C, then it may perform an action a and reach the state represented by C'.

Definition. A sequence $a_1 \cdots a_n \in Act^*$ is a *(sequential) trace* of an event structure $\mathcal{E}$ if there exist configurations $C_0, \cdots, C_n$ of $\mathcal{E}$ such that $C_0 = \varnothing$ and $C_{i-1} \xrightarrow{a_i}_{\mathcal{E}} C_i$ $(i = 1, \cdots, n)$.

SeqTraces$(\mathcal{E})$ denotes the set of all sequential traces of $\mathcal{E}$.

Two event structures $\mathcal{E}$ and $\mathcal{F}$ are *interleaving trace equivalent* - notation $\mathcal{E} \approx_{it} \mathcal{F}$ - if $SeqTraces(\mathcal{E}) = SeqTraces(\mathcal{F})$.

Definition. Let $\mathcal{E}, \mathcal{F} \in \mathbb{E}$. A relation $R \subseteq \mathcal{C}(\mathcal{E}) \times \mathcal{C}(\mathcal{F})$ is called a *(sequential) bisimulation* between $\mathcal{E}$ and $\mathcal{F}$ if $(\varnothing, \varnothing) \in R$ and whenever $(C,D) \in R$ then for $a \in Act$:

- $C \xrightarrow{a}_{\mathcal{E}} C' \Rightarrow \exists D'$ with $D \xrightarrow{a}_{\mathcal{F}} D'$ and $(C',D') \in R$;
- $D \xrightarrow{a}_{\mathcal{F}} D' \Rightarrow \exists C'$ with $C \xrightarrow{a}_{\mathcal{E}} C'$ and $(C',D') \in R$.

$\mathcal{E}$ and $\mathcal{F}$ are *interleaving bisimulation equivalent* - $\mathcal{E} \approx_{ib} \mathcal{F}$ - if there exists a sequential bisimulation between them.

Step equivalences can be defined by generalizing the single action transition relations $\xrightarrow{a}_{\mathcal{E}} \subseteq \mathcal{C}(\mathcal{E}) \times \mathcal{C}(\mathcal{E})$ to *step transition relations* $\xrightarrow{A}_{\mathcal{E}} \subseteq \mathcal{C}(\mathcal{E}) \times \mathcal{C}(\mathcal{E})$, where A is a multiset over Act.

Definition. Let $\mathcal{E}$ be an event structure and $A : Act \rightarrow \mathbb{N}$ a multiset over Act. For $X \subseteq E_{\mathcal{E}}$ let $l_{\mathcal{E}}(X) \in \mathbb{N}^{Act}$ be the multiset of labels of the events from X, defined by $l_{\mathcal{E}}(X)(a) = |\{e \in X \mid l_{\mathcal{E}}(e) = a\}|$.
Then $C \xrightarrow{A}_{\mathcal{E}} C'$ iff $C \longrightarrow_{\mathcal{E}} C'$ and $C' - C = G \subseteq E_{\mathcal{E}}$ such that $\forall e,e' \in G$: $e \; co_{\mathcal{E}} \; e'$ and $l_{\mathcal{E}}(G) = A$.

Here $C \xrightarrow{A}_{\mathcal{E}} C'$ says that if the system represented by $\mathcal{E}$ is in the state represented by C, then it may concurrently perform the multiset of actions A and reach the state represented by C'. Since A is a multiset rather than a set, actions may occur concurrently with themselves ('autoconcurrency').

Definition. A sequence $A_1 \cdots A_n$ of multisets $A_i \in \mathbb{N}^{Act}$ $(i = 1,...,n)$ is a *step trace* of an event structure $\mathcal{E}$ if there exist configurations $C_0, \cdots, C_n$ of $\mathcal{E}$ such that $C_0 = \varnothing$ and $C_{i-1} \xrightarrow{A_i}_{\mathcal{E}} C_i$ $(i = 1,...,n)$.
StepTraces$(\mathcal{E})$ denotes the set of all step traces of $\mathcal{E}$.
Two event structures $\mathcal{E}$ and $\mathcal{F}$ are *step trace equivalent* - $\mathcal{E} \approx_{st} \mathcal{F}$ - if *StepTraces*$(\mathcal{E})$ = *StepTraces*$(\mathcal{F})$.

Definition. Let $\mathcal{E}, \mathcal{F} \in \mathbb{E}$. A relation $R \subseteq \mathcal{C}(\mathcal{E}) \times \mathcal{C}(\mathcal{F})$ is called a *step bisimulation* between $\mathcal{E}$ and $\mathcal{F}$ if $(\varnothing, \varnothing) \in R$ and whenever $(C,D) \in R$ then for $A \in \mathbb{N}^{Act}$:

- $C \xrightarrow{A}_{\mathcal{E}} C' \Rightarrow \exists D'$ with $D \xrightarrow{A}_{\mathcal{F}} D'$ and $(C',D') \in R$;
- $D \xrightarrow{A}_{\mathcal{F}} D' \Rightarrow \exists C'$ with $C \xrightarrow{A}_{\mathcal{E}} C'$ and $(C',D') \in R$.

$\mathcal{E}$ and $\mathcal{F}$ are *step bisimulation equivalent* - $\mathcal{E} \approx_{sb} \mathcal{F}$ - if there exists a step bisimulation between them.

A trace equivalence preserving causal dependencies between action occurrences in runs of processes is the *pomset trace equivalence* as implicitly employed, for instance, in Pratt [107].

Definition. A *partially ordered multiset (pomset)* is an isomorphism class of conflict-free event structures. A pomset u is a *pomset trace* of an event structure $\mathcal{E}$ if u is the isomorphism class of $\mathcal{E} \upharpoonright C$ for some configuration $C \in \mathcal{C}(\mathcal{E})$.
Pomsets$(\mathcal{E})$ denotes the set of all pomset traces of $\mathcal{E}$.
Two event structures $\mathcal{E}$ and $\mathcal{F}$ are *pomset trace equivalent* - $\mathcal{E} \approx_{pt} \mathcal{F}$ - if *Pomsets*$(\mathcal{E})$ = *Pomsets*$(\mathcal{F})$.

Sequential traces, step traces as well as pomset traces of an event structure $\mathcal{E}$ represent possible (partial) runs of the system represented by $\mathcal{E}$. A trace of each of these three types specifies a multiset of actions, executed during such a run. However, whereas sequential and step traces in addition only specify a possible order in which these actions may occur (with and without the possibility of simultaneous occurrences), a pomset trace specifies all causal dependencies between the occurrences of these actions, through the partial order inherited from $\mathcal{E}$. From this information all the possible orders in which the actions may occur can be derived.

Like pomset trace equivalence, most of the equivalences that preserve causal dependencies between occurrences of actions are defined by means of partial orders. Therefore, such equivalences are called *partial order equivalences.* It happens that on $\mathbb{E}$ there is only one reasonable trace equivalence based on partial orders - namely $\approx_{pt}$ - and the same can be said about trace equivalences based on steps and on interleaving and about bisimulation equivalences based on steps and on interleaving. However, of late years several bisimulation equivalences based on partial orders have been defined on $\mathbb{E}$:

1986: the *NMS partial ordering equivalence* of DEGANO, DE NICOLA & MONTANARI [40],

1986: the *pomset bisimulation equivalence* or *equipollence* of BOUDOL & CASTELLANI [31],

1987: the *generalized pomset bisimulation equivalence* of VAN GLABBEEK & VAANDRAGER [58] and

1988: the *behaviour structure bisimulation equivalence* of RABINOVICH & TRAKHTENBROT [108].

In my opinion only the last - and finest - one fully captures the interplay of causality and branching and is most worthy of filling up the right upper corner of Figure 1. Originally it was defined on behaviour structures [108], but in [54] (Chapter VI of this thesis) the notion was defined on event structures as well, under the name *history preserving bisimulation equivalence.*

DEFINITION. Let $\mathcal{E},\mathcal{F}\in\mathbb{E}$. A relation $R\subseteq\mathcal{C}(\mathcal{E})\times\mathcal{C}(\mathcal{F})\times\mathcal{P}(E_{\mathcal{E}}\times E_{\mathcal{F}})$ is called a *history preserving bisimulation* between $\mathcal{E}$ and $\mathcal{F}$ if $(\varnothing,\varnothing,\varnothing)\in R$ and whenever $(C,D,f)\in R$ then:

- $f:C\rightarrow D$ is an isomorphism between $\mathcal{E}\restriction C$ and $\mathcal{F}\restriction D$;
- $C\longrightarrow_{\mathcal{E}}C' \Rightarrow \exists D',f'$ with $D\longrightarrow_{\mathcal{F}}D'$, $(C',D',f')\in R$ and $f'\restriction C=f$;
- $D\longrightarrow_{\mathcal{F}}D' \Rightarrow \exists C',f'$ with $C\longrightarrow_{\mathcal{E}}C'$, $(C',D',f')\in R$ and $f'\restriction C=f$.

$\mathcal{E}$ and $\mathcal{F}$ are *history preserving bisimulation equivalent* - $\mathcal{E}\approx_h\mathcal{F}$ - if there exists a history preserving bisimulation between them.

PROPOSITION 3: *For all equivalences $\approx_1$ and $\approx_2$ defined in this section, the formula*

$$\forall\mathcal{E},\mathcal{F}\in\mathbb{E}:\ \mathcal{E}\approx_1\mathcal{F}\Rightarrow\mathcal{E}\approx_2\mathcal{F}$$

holds iff there is a path $\approx_1\rightarrow\cdots\rightarrow\approx_2$ in Figure 1.

PROOF: The implications follow directly from the definitions; in order to prove the absence of other implications, it suffices to provide counterexamples against $\approx_{pt}\rightarrow\approx_{ib}$, $\approx_{ib}\rightarrow\approx_{st}$ and $\approx_{sb}\rightarrow\approx_{pt}$.

COUNTEREXAMPLES. In the graphical representations of event structures below, the conventions of [119] are followed: the conflict relation is denoted by means of dotted lines, only immediate conflicts - not the inherited ones - are indicated; the causality relation is represented by arrows, omiting those derivable by transitivity; and instead of events only their labels are displayed, if a label occurs twice it represents two different events. Thus these pictures determine event structures only up to isomorphism.

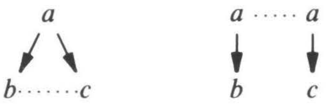


FIGURE 3. *Pomset trace equivalent but not interleaving bisimulation equivalent (standard example)*

The two event structures of Figure 3 are pomset trace equivalent: their pomset traces are $a \rightarrow b$, $a \rightarrow c$, a and the empty pomset. However, they are not interleaving bisimulation equivalent: both systems represented perform first the action a and then either b or c, but the first system makes the choice between b and c after the execution of a whereas the second one starts with making this choice.

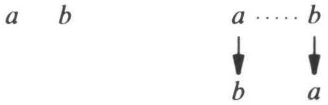


FIGURE 4. *Interleaving bisimulation equivalent but not step trace equivalent (standard example)*

The first system represented in Figure 4 performs two actions a and b concurrently. The second one either performs b after completion of a or vice versa. In interleaving semantics these systems are identified. However, they are not step trace equivalent: only the first system can perform a and b simultaneously.

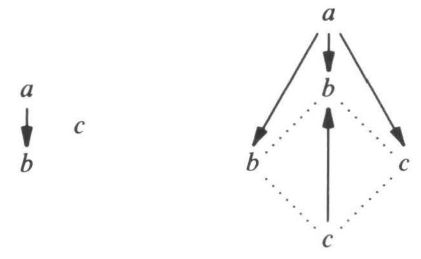


FIGURE 5. *Step bisimulation equivalent but not pomset trace equivalent (new)*

The two systems represented in Figure 5 are step bisimulation equivalent: both systems perform the actions *a*, *b* and *c* exactly once; in both cases *a* is a prerequisite for *b*, and *c* can happen before *a*, simultaneous with *a*, between *a* and *b*, simultaneous with *b*, or after *b*; and in both cases all choices between alternative courses of action are made only when one of the alternatives actually occurs. However, they are not pomset trace equivalent: the pomset resembling the first event structure of Figure 5 is a pomset trace of this first event structure, but not of the second one. □

THEOREM: *Of all equivalences mentioned in this section, only $\approx_{pt}$ and $\approx_h$ are preserved under action refinement.*

PROOF: The two event structures of Figure 5 are step bisimulation equivalent. However, after refining c in $c_1 \rightarrow c_2$ the resulting event structures (below) are not even interleaving trace equivalent.

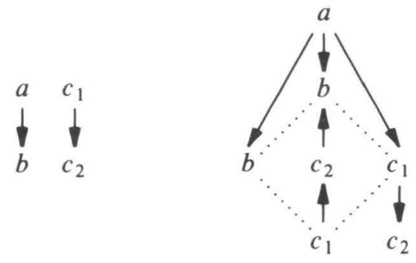


FIGURE 6. *Refined event structures*

Only the first one has a trace $c_1\ a\ b\ c_2$. This shows that no equivalence that is at least as fine as interleaving trace equivalence and at least as coarse as step bisimulation equivalence is preserved under refinement of actions. More counterexamples and the refinement theorems for $\approx_{pt}$ and $\approx_h$ can be found in Chapter VI. □

4. The behaviour of concurrent systems II

4. The behaviour of concurrent systems II. A configuration of an event structure $\mathcal{E}$ represents a state S of the system represented by $\mathcal{E}$ by considering two kinds of events with respect to S: those that happened before the system reached this state and those that did not happen (yet). I argue that when events or transitions are considered to have a duration or structure, such configurations do not properly represent all the states of the represented system. Instead I propose to consider a third kind of events with respect to S: those that are currently happening when the system is in state S. This gives rise to the introduction of *ST-configurations* (a name explained in the introduction).

Definition. An *ST-configuration* of $\mathcal{E}$ is a pair (C,P) of subsets of $E_{\mathcal{E}}$, such that $P \subseteq C$, C is finite and conflict-free and

$$e' <_{\mathcal{E}} e \in C \Rightarrow e' \in P.$$

Thus both P and C are configurations and $C-P$ contains only maximal elements in C. An ST-configuration (C,P) represents the state of a concurrent system where C is the set of events whose execution has been started and P (the *past*) is the set of events whose execution has been completed. An ordinary configuration can be regarded as an ST-configuration with $P=C$. Let $\mathcal{S}(\mathcal{E})$ be the set of ST-configurations of $\mathcal{E}$. Write $(C,P) \longrightarrow_{\mathcal{E}} (C',P')$ if $(C,P),(C',P') \in \mathcal{S}(\mathcal{E})$, $C \subseteq C'$ and $P \subseteq P'$.

As in Section 2, the behaviour of a refined event structure $r(\mathcal{E})$ may be deduced from the behaviour of $\mathcal{E}$ and from the behaviour of the event structures which are substituted for actions.

Notation. For each pair $(\tilde{C}, \tilde{P}) \in \mathcal{P}(E_{r(\mathcal{E})}) \times \mathcal{P}(E_{r(\mathcal{E})})$ with $\tilde{P} \subseteq \tilde{C}$, there are unique sets $C_e, P_e \subseteq E_{r(e)}$ for every $e \in pr_1(\tilde{C})$ such that $\tilde{C} = \{(e,e') \mid e \in pr_1(\tilde{C}),\ e' \in C_e\}$ and $\tilde{P} = \{(e,e') \mid e \in pr_1(\tilde{C}),\ e' \in P_e\}$. In fact $C_e = \{e' \mid (e,e') \in \tilde{C}\}$ and $P_e = \{e' \mid (e,e') \in \tilde{P}\}$. Now $r^{-1}(\tilde{C}, \tilde{P})$ denotes the unique pair $(C,P) \in \mathcal{P}(E_{\mathcal{E}}) \times \mathcal{P}(E_{\mathcal{E}})$ such that $C = pr_1(\tilde{C})$ and $P = \{e \in C \mid P_e = E_{r(e)}\}$.

Lemma 4: *Let $\mathcal{E}$ be an event structure and r a refinement.*

i. $(\tilde{C}, \tilde{P}) \in \mathcal{P}(E_{r(\mathcal{E})}) \times \mathcal{P}(E_{r(\mathcal{E})})$ *is an ST-configuration of $r(\mathcal{E})$ iff*

$\tilde{C} = \{(e,e') \mid e \in C,\ e' \in C_e\}$ *and* $\tilde{P} = \{(e,e') \mid e \in C,\ e' \in P_e\}$ *where*

(C,P) *is an ST-configuration of* $\mathcal{E}$,
(C_e,P_e) *is an ST-configuration of* $r(e)$ *for* $e \in C$,
$P_e = E_{r(e)}$ *iff* $e \in P$.

ii. *If* $(\tilde{C},\tilde{P}) \longrightarrow_{r(\mathcal{E})} (\tilde{C}',\tilde{P}')$ *then* $r^{-1}(\tilde{C},\tilde{P}) \longrightarrow_{\mathcal{E}} r^{-1}(\tilde{C}',\tilde{P}')$.

Proof: i. "$\Rightarrow$". Let $(\tilde{C},\tilde{P}) \in \mathcal{S}(r(\mathcal{E}))$.

First I show that $(C,P) := r^{-1}(\tilde{C},\tilde{P}) \in \mathcal{S}(\mathcal{E})$.

- $P \subseteq C$ by definition.
- C is finite and conflict-free since $\tilde{C}$ is finite and conflict-free.
- Suppose $d <_{\mathcal{E}} e \in C$. I have to show that $d \in P$.
  Since $e \in C = pr_1(\tilde{C})$ there exists $(e,e') \in \tilde{C}$;
  since $r(d)$ is non-empty there exists $(d,d') \in E_{r(\mathcal{E})}$;
  since $d <_{\mathcal{E}} e$ one has $(d,d') <_{r(\mathcal{E})} (e,e') \in \tilde{C}$;
  and since $(\tilde{C},\tilde{P})$ is an ST-configuration it follows that $(d,d') \in \tilde{P} \subseteq \tilde{C}$.
  Thus $d \in C$. So it remains to be proven that $P_d = E_{r(d)}$.
  Obviously $P_d \subseteq E_{r(d)}$.
  Now let $d' \in E_{r(d)}$. Then $(d,d') \in E_{r(\mathcal{E})}$. Exactly as above one obtains $(d,d') \in \tilde{P}$, and hence $d' \in P_d$. Thus $P_d = E_{r(d)}$ and $d \in P$.

Next let $e \in C$. Put $C_e = \{e' \mid (e,e') \in \tilde{C}\}$ and $P_e = \{e' \mid (e,e') \in \tilde{P}\}$.
I show that $(C_e,P_e) \in \mathcal{S}(r(e))$.

- $P_e \subseteq C_e$ since $\tilde{P} \subseteq \tilde{C}$.
- C_e is finite since $\tilde{C}$ is finite.
- C_e is conflict-free since $r(e)$ is conflict-free.
- Suppose $e' <_{r(e)} e'' \in C_e$. Then $(e,e') <_{r(\mathcal{E})} (e,e'') \in \tilde{C}$. Hence $(e,e') \in \tilde{P}$ and $e' \in P_e$.

Finally the third requirement is met by construction.
"$\Leftarrow$". Let $(C,P) \in \mathcal{S}(\mathcal{E})$ and $(C_e,P_e) \in \mathcal{S}(r(e))$ for $e \in C$. Suppose $P_e = E_{r(e)} \Leftrightarrow e \in P$ for $e \in C$. Put $\tilde{C} = \{(e,e') \mid e \in C,\ e' \in C_e\}$ and $\tilde{P} = \{(e,e') \mid e \in C,\ e' \in P_e\}$. I show that $(\tilde{C},\tilde{P}) \in \mathcal{S}(r(\mathcal{E}))$.

- $\tilde{P} \subseteq \tilde{C}$ since $P_e \subseteq C_e$ for $e \in C$.
- $\tilde{C}$ is finite since C and C_e are finite.
- $\tilde{C}$ is conflict-free since $C = pr_1(\tilde{C})$ is conflict-free.
- Suppose $(d,d') <_{r(\mathcal{E})} (e,e') \in \tilde{C}$. Then $d <_{\mathcal{E}} e$ or $d = e \wedge d' <_{r(e)} e'$.
  If $d <_{\mathcal{E}} e$ then $d \in P$, since $e \in C$ and $(C,P) \in \mathcal{S}(\mathcal{E})$. Thus $d \in C$ and $P_d = E_{r(d)}$. Since $d' \in E_{r(d)} = P_d = \{d' \mid (d,d') \in \tilde{P}\}$ it follows that $(d,d') \in \tilde{P}$.
  If $d = e$ then $d' \in P_e = P_d$, since $d' <_{r(e)} e' \in C_e$ and $(C_e,P_e) \in \mathcal{S}(r(e))$. So also in this case one has $(d,d') \in \tilde{P}$, which had to be proved.

ii. Suppose $(\tilde{C},\tilde{P}) \longrightarrow_{r(\mathcal{E})} (\tilde{C}',\tilde{P}')$, i.e. $(\tilde{C},\tilde{P}),(\tilde{C}',\tilde{P}') \in \mathcal{S}(r(\mathcal{E}))$, $\tilde{C} \subseteq \tilde{C}'$

and $\tilde{P} \subseteq \tilde{P}'$. Then $r^{-1}(\tilde{C}, \tilde{P}), r^{-1}(\tilde{C}', \tilde{P}') \in \mathcal{S}(\mathcal{E})$ (by i.) and $r^{-1}(\tilde{C}, \tilde{P}) \longrightarrow_{\mathcal{E}} r^{-1}(\tilde{C}', \tilde{P}')$ by definition. □

I will end this section with a proposition saying that the ST-configurations of an event structure $\mathcal{E}$ describe the behaviour of the represented concurrent system in the same way as the ordinary configurations of the *split* event structure *split*$(\mathcal{E})$, obtained from $\mathcal{E}$ by splitting every action a into the sequence of actions a^+ and a^-, representing the beginning and the end of a.

DEFINITION. For Λ a set of labels, let $\mathbb{E}(\Lambda)$ denote the domain of event structures labelled over Λ. So $\mathbb{E} = \mathbb{E}(Act)$.
A Λ*-refinement* $r: Act \rightarrow \mathbb{E}(\Lambda) - \{0\}$ is a function that takes any action $a \in Act$ into a finite, conflict-free, non-empty event structure $r(a) \in \mathbb{E}(\Lambda)$. So a refinement as defined in Section 1 of this chapter is an *Act*-refinement. If $\mathcal{E} \in \mathbb{E}$ and r is a Λ-refinement, then $r(\mathcal{E}) \in \mathbb{E}(\Lambda)$ is defined exactly as in Section 1.

DEFINITION. Put $Act^{\pm} = \{a^+ \mid a \in Act\} \cup \{a^- \mid a \in Act\}$. Let the $Act^{\pm}$-refinement $split: Act \rightarrow \mathbb{E}(Act^{\pm})$ be defined by $E_{split(a)} = \{a^+, a^-\}$, $a^+ <_{split(a)} a^-$ and $l_{split(a)}(a^+) = a^+$, $l_{split(a)}(a^-) = a^-$. It induces a function $split: \mathbb{E}(Act) \rightarrow \mathbb{E}(Act^{\pm})$. This function was introduced on Petri nets in [58], and on event structures in [119].

PROPOSITION 4: *For each event structure* $\mathcal{E} \in \mathbb{E}$, *there exists a bijective mapping* $i_{\mathcal{E}}: \mathcal{S}(\mathcal{E}) \rightarrow \mathcal{C}(split(\mathcal{E}))$, *such that for* $S \in \mathcal{S}(\mathcal{E})$:

$$S \longrightarrow_{\mathcal{E}} S' \;\Leftrightarrow\; i_{\mathcal{E}}(S) \longrightarrow_{split(\mathcal{E})} i_{\mathcal{E}}(S').$$

PROOF: $i_{\mathcal{E}}(C,P) = \{(e, (l_{\mathcal{E}}(e))^+) \mid e \in C\} \cup \{(e, (l_{\mathcal{E}}(e))^-) \mid e \in P\}$. Verification of all requirements is straightforward. □

5. Equivalence notions for concurrent systems II

In this section the remaining equivalences of Figure 2 are defined in terms of ST-configurations.
The most straightforward generalization of interleaving semantics to the setting of ST-configurations yields split semantics. Split equivalences can be defined by generalizing the single action transition relations $\xrightarrow{a}_{\mathcal{E}} \subseteq \mathcal{C}(\mathcal{E}) \times \mathcal{C}(\mathcal{E})$ to *split transition relations* $\xrightarrow{a^+}_{\mathcal{E}}, \xrightarrow{a^-}_{\mathcal{E}} \subseteq \mathcal{S}(\mathcal{E}) \times \mathcal{S}(\mathcal{E})$, for $a \in Act$ and $\mathcal{E} \in \mathbb{E}$.

DEFINITION. $(C,P) \xrightarrow{a^+}_{\mathcal{E}} (C',P')$ iff $(C,P) \longrightarrow_{\mathcal{E}} (C',P')$, $P' = P$ and $C' - C = \{e\}$ with $l_{\mathcal{E}}(e) = a$.
$(C,P) \xrightarrow{a^-}_{\mathcal{E}} (C',P')$ iff $(C,P) \longrightarrow_{\mathcal{E}} (C',P')$, $C' = C$ and $P' - P = \{e\}$ with $l_{\mathcal{E}}(e) = a$.

Here $(C,P) \xrightarrow{a^+}_{\mathcal{E}} (C',P')$ says that if the system represented by $\mathcal{E}$ is in the state represented by (C,P), then it may start performing an action a and reach the state represented by (C',P').

Furthermore $(C,P) \xrightarrow{a^-}_{\mathcal{E}} (C',P')$ says that if the system is in the state represented by (C,P), then it may end performing an action a and reach the state represented by (C',P').

DEFINITION. A sequence $a_1 \cdots a_n \in (Act^{\pm})^*$ is a *split trace* of an event structure $\mathcal{E}$ if there exist ST-configurations $(C_0,P_0), \cdots ,(C_n,P_n)$ of $\mathcal{E}$ such that $(C_0,P_0)=(\varnothing,\varnothing)$ and $(C_{i-1},P_{i-1}) \xrightarrow{a_i}_{\mathcal{E}} (C_i,P_i)$ $(i=1, \cdots ,n)$. $SplitTraces(\mathcal{E})$ denotes the set of all split traces of $\mathcal{E}$.
Two event structures $\mathcal{E}$ and $\mathcal{F}$ are *split trace equivalent* - $\mathcal{E} \approx_{2t} \mathcal{F}$ - if $SplitTraces(\mathcal{E}) = SplitTraces(\mathcal{F})$.

DEFINITION. Let $\mathcal{E},\mathcal{F} \in \mathbb{E}$. A relation $R \subseteq \mathcal{S}(\mathcal{E}) \times \mathcal{S}(\mathcal{F})$ is called a *split bisimulation* between $\mathcal{E}$ and $\mathcal{F}$ if $((\varnothing,\varnothing),(\varnothing,\varnothing)) \in R$ and whenever $((C,P),(D,Q)) \in R$ then for $a \in Act^{\pm}$:

- $(C,P) \xrightarrow{a}_{\mathcal{E}} (C',P') \Rightarrow \exists D',Q'$ with $(D,Q) \xrightarrow{a}_{\mathcal{F}} (D',Q')$ and $((C',P'),(D',Q')) \in R$;
- $(D,Q) \xrightarrow{a}_{\mathcal{F}} (D',Q') \Rightarrow \exists C',P'$ with $(C,P) \xrightarrow{a}_{\mathcal{E}} (C',P')$ and $((C',P'),(D',Q')) \in R$.

$\mathcal{E}$ and $\mathcal{F}$ are *split bisimulation equivalent* - $\mathcal{E} \approx_{2b} \mathcal{F}$ - if there exists a split bisimulation between them.

Alternatively, split equivalences can be defined as ordinary interleaving equivalences on split event structures, and even as step equivalences on split event structures. The following proposition says that this yields the same trace and bisimulation equivalences as the definitions above.

PROPOSITION 5.1: $\mathcal{E} \approx_{2t} \mathcal{F} \Leftrightarrow split(\mathcal{E}) \approx_{it} split(\mathcal{F}) \Leftrightarrow split(\mathcal{E}) \approx_{st} split(\mathcal{F})$
$\mathcal{E} \approx_{2b} \mathcal{F} \Leftrightarrow split(\mathcal{E}) \approx_{ib} split(\mathcal{F}) \Leftrightarrow split(\mathcal{E}) \approx_{sb} split(\mathcal{F})$.
PROOF: Let $i_{\mathcal{E}}:\mathcal{S}(\mathcal{E}) \rightarrow \mathcal{C}(split(\mathcal{E}))$ be the bijection from the previous proposition, then for $S \in \mathcal{S}(\mathcal{E})$ and $a \in Act^{\pm}$: $S \xrightarrow{a}_{\mathcal{E}} S' \Leftrightarrow i_{\mathcal{E}}(S) \xrightarrow{a}_{split(\mathcal{E})} i_{\mathcal{E}}(S')$. Furthermore, if $C,C' \in \mathcal{C}(split(\mathcal{E}))$ and A is a multiset over $Act^{\pm}$ consisting of the actions $a_1^+, \cdots ,a_n^+ ,b_1^- , \cdots ,b_m^-$ then

$$C \xrightarrow{A}_{split(\mathcal{E})} C' \Leftrightarrow C \xrightarrow{a_1^+}_{split(\mathcal{E})} \cdots \xrightarrow{a_n^+}_{split(\mathcal{E})} \xrightarrow{b_1^-}_{split(\mathcal{E})} \cdots \xrightarrow{b_m^-}_{split(\mathcal{E})} C'.$$

From this the proposition follows immediately. □

Split-semantics is just interleaving semantics, but based on interleaving of beginnings and ends of action occurrences, instead of entire action occurrences. However, since different occurrences of the same action can not be distinguished, it is in general not possible to tell when an occurrence of a^+

and an occurrence of a^- originate from to the same occurrence of a. ST-semantics is a refinement of split semantics, where occurrences of a^+ and a^- are explicitly connected if they represent the beginning and end of the same occurrence of a.

DEFINITION. A *pre-interval sequence* is a triple (E,l,σ) with E a set, $l\colon E \rightarrow Act$ a labelling function and σ a sequence over $E^{\pm}=\{e^+ \mid e \in E\} \cup \{e^- \mid e \in E\}$ whose elements are all different, and which can contain e^- only after e^+ (for $e \in E$). For $l\colon E \rightarrow Act$ define $l^{\pm}\colon E^{\pm} \rightarrow Act^{\pm}$ by $l(e^+)=(l(e))^+$ and $l(e^-)=(l(e))^-$. Let (E,l,σ) with $\sigma=\alpha_1 \cdots \alpha_n \in (E^{\pm})^*$ be a pre-interval sequence and let $1 \leqslant i < j \leqslant n$. α_i and α_j are *connected*, notation $\alpha_i \prec \alpha_j$, if $\alpha_i=e^+$ and $\alpha_j=e^-$ for certain $e \in E$. Now two pre-interval sequences $(E,l,\alpha_1 \cdots \alpha_n)$ and $(E',l',\beta_1 \cdots \beta_m)$ are *isomorphic* if $n=m$, $l^{\pm}(\alpha_i)=l'^{\pm}(\beta_i)$ for $1 \leqslant i \leqslant n$, and $\alpha_i \prec \alpha_j \Leftrightarrow \beta_i \prec \beta_j$ for $1 \leqslant i < j \leqslant n$. An *interval sequence* is an isomorphism class of pre-interval sequences.

EXAMPLE: Let $E=\{e_0,e_1,e_2,e_3,e_4\}$, $l(e_1)=l(e_2)=l(e_3)=a$ and $l(e_0)=l(e_4)=b$. Figure 7 shows a pre-interval sequence over E, together with its associated interval sequence. The connectedness relation $\prec$ is represented by arcs.

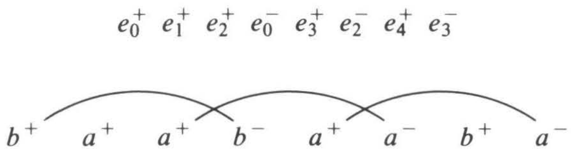

e_0^+ e_1^+ e_2^+ e_0^- e_3^+ e_2^- e_4^+ e_3^-
b^+ a^+ a^+ b^- a^+ a^- b^+ a^-

FIGURE 7. *Pre-interval sequence and interval sequence*

DEFINITION. $(C,P) \xrightarrow{e^+}_{\mathcal{E}} (C',P')$ iff $(C,P) \longrightarrow_{\mathcal{E}} (C',P')$, $P'=P$ and $C'-C=\{e\}$.
$(C,P) \xrightarrow{e^-}_{\mathcal{E}} (C',P')$ iff $(C,P) \longrightarrow_{\mathcal{E}} (C',P')$, $C'=C$ and $P'-P=\{e\}$.

A structure $(E_{\mathcal{E}},l_{\mathcal{E}},\alpha_1 \cdots \alpha_n)$ is a *pre-ST-trace* of an event structure $\mathcal{E}$ if there exist ST-configurations $(C_0,P_0), \cdots, (C_n,P_n)$ of $\mathcal{E}$ such that $(C_0,P_0)=(\varnothing,\varnothing)$ and $(C_{i-1},P_{i-1}) \xrightarrow{\alpha_i}_{\mathcal{E}} (C_i,P_i)$ $(i=1, \cdots, n)$. An *ST-trace* of $\mathcal{E}$ is an interval sequence which is the isomorphism class of a pre-ST-trace of $\mathcal{E}$.
ST-Traces$(\mathcal{E})$ denotes the set of all ST-traces of $\mathcal{E}$.
Two event structures $\mathcal{E}$ and $\mathcal{F}$ are *ST-trace equivalent* - $\mathcal{E} \approx_{STt} \mathcal{F}$ - if *ST-Traces*$(\mathcal{E})$=*ST-Traces*$(\mathcal{F})$.

Next I propose another characterization of ST-trace equivalence that will be more convenient later on.

DEFINITION. $\mathcal{E} \lesssim_{STt} \mathcal{F}$ iff for every chain of ST-configurations

$$(\varnothing,\varnothing) \longrightarrow_{\mathcal{E}} (C_1,P_1) \longrightarrow_{\mathcal{E}} \cdots \longrightarrow_{\mathcal{E}} (C_n,P_n)$$

in $\mathcal{E}$ there is a chain

$$(\varnothing,\varnothing) \longrightarrow_{\mathcal{F}} (D_1,Q_1) \longrightarrow_{\mathcal{F}} \cdots \longrightarrow_{\mathcal{F}} (D_n,Q_n)$$

in $\mathcal{F}$ and a bijection $f:C_n \to D_n$, satisfying $l_{\mathcal{F}}(f(e)) = l_{\mathcal{E}}(e)$, $f(C_i)=D_i$ and $f(P_i)=Q_i$ for $i=1,\cdots,n$.

PROPOSITION 5.2: $\mathcal{E} \approx_{STt} \mathcal{F} \Leftrightarrow (\mathcal{E} \lesssim_{STt} \mathcal{F} \wedge \mathcal{F} \lesssim_{STt} \mathcal{E})$.
PROOF: Write $\mathcal{E} \lesssim^{*}_{STt} \mathcal{F}$ iff for every chain

$$(\varnothing,\varnothing) \xrightarrow{\alpha_1}_{\mathcal{E}} (C_1,P_1) \xrightarrow{\alpha_2}_{\mathcal{E}} \cdots \xrightarrow{\alpha_n}_{\mathcal{E}} (C_n,P_n)$$

in $\mathcal{E}$ (with $\alpha_i \in E^{\pm}_{\mathcal{E}}$) there is a chain

$$(\varnothing,\varnothing) \xrightarrow{\beta_1}_{\mathcal{F}} (D_1,Q_1) \xrightarrow{\beta_2}_{\mathcal{F}} \cdots \xrightarrow{\beta_n}_{\mathcal{F}} (D_n,Q_n)$$

in $\mathcal{F}$ and a bijection $f:C_n \to D_n$, satisfying $l_{\mathcal{F}}(f(e)) = l_{\mathcal{E}}(e)$, $f(C_i)=D_i$ and $f(P_i)=Q_i$ for $i=1,\cdots,n$.
Furthermore write $\mathcal{E} \lesssim^{**}_{STt} \mathcal{F}$ iff for every chain

$$(\varnothing,\varnothing) \xrightarrow{\alpha_1}_{\mathcal{E}} (C_1,P_1) \xrightarrow{\alpha_2}_{\mathcal{E}} \cdots \xrightarrow{\alpha_n}_{\mathcal{E}} (C_n,P_n)$$

in $\mathcal{E}$ (with $\alpha_i \in E^{\pm}_{\mathcal{E}}$) there is a chain

$$(\varnothing,\varnothing) \xrightarrow{\beta_1}_{\mathcal{F}} (D_1,Q_1) \xrightarrow{\beta_2}_{\mathcal{F}} \cdots \xrightarrow{\beta_n}_{\mathcal{F}} (D_n,Q_n)$$

in $\mathcal{F}$ such that $l^{\pm}_{\mathcal{E}}(\alpha_i)=l^{\pm}_{\mathcal{F}}(\beta_i)$ for $1 \leqslant i \leqslant n$, and $\alpha_i < \alpha_j \Leftrightarrow \beta_i < \beta_j$ for $1 \leqslant i < j \leqslant n$.
CLAIM 1: $\mathcal{E} \lesssim_{STt} \mathcal{F} \Leftrightarrow \mathcal{E} \lesssim^{*}_{STt} \mathcal{F}$.
CLAIM 2: $\mathcal{E} \lesssim^{*}_{STt} \mathcal{F} \Leftrightarrow \mathcal{E} \lesssim^{**}_{STt} \mathcal{F}$.
CLAIM 3: $\mathcal{E} \lesssim^{**}_{STt} \mathcal{F} \Leftrightarrow$ *ST-Traces*$(\mathcal{E}) \subseteq$ *ST-Traces*$(\mathcal{F})$.
Now the proposition follows by combination of these claims.
Proof of claim 1: "$\Rightarrow$". Suppose $\mathcal{E} \lesssim_{STt} \mathcal{F}$ and $(\varnothing,\varnothing) \xrightarrow{\alpha_1}_{\mathcal{E}} (C_1,P_1) \xrightarrow{\alpha_2}_{\mathcal{E}} \cdots \xrightarrow{\alpha_n}_{\mathcal{E}} (C_n,P_n)$ is a chain in $\mathcal{E}$ with $\alpha_i \in E^{\pm}_{\mathcal{E}}$. Then there must be a chain $(\varnothing,\varnothing) \longrightarrow_{\mathcal{F}} (D_1,Q_1) \longrightarrow_{\mathcal{F}} \cdots \longrightarrow_{\mathcal{F}} (D_n,Q_n)$ in $\mathcal{F}$ and a bijection $f:C_n \to D_n$, satisfying $l_{\mathcal{F}}(f(e)) = l_{\mathcal{E}}(e)$, $f(C_i)=D_i$ and $f(P_i)=Q_i$ for $i=1,\cdots,n$. Because of this bijection - only considering the 'sizes' of D_i and Q_i - there must be $\beta_i \in E^{\pm}_{\mathcal{F}}$ for $i=1,\cdots,n$ such that $(C_{i-1},P_{i-1}) \xrightarrow{\beta_i}_{\mathcal{E}} (C_i,P_i)$.
"$\Leftarrow$". This follows from the observation that whenever in an event structure $\mathcal{E}$ $(C,P) \longrightarrow_{\mathcal{E}} (C',P')$, there exist ST-configurations $(C_0,P_0),\cdots,(C_k,P_k)$ of $\mathcal{E}$ and a sequence $\alpha_1 \cdots \alpha_k \in (E^{\pm}_{\mathcal{E}})^*$ such that $(C_0,P_0)=(C,P)$, $(C_{i-1},P_{i-1}) \xrightarrow{\alpha_i}_{\mathcal{E}} (C_i,P_i)$ $(i=1,\cdots,k)$, and $(C_k,P_k)=(C',P')$.
Proof of claim 2: "$\Rightarrow$". Let $(\varnothing,\varnothing) \xrightarrow{\alpha_1}_{\mathcal{E}} \cdots \xrightarrow{\alpha_n}_{\mathcal{E}} (C_n,P_n)$ and

$(\varnothing,\varnothing) \xrightarrow{\beta_1}_{\mathcal{F}} \cdots \xrightarrow{\beta_n}_{\mathcal{F}} (D_n,Q_n)$ be chains of ST-configurations in $\mathcal{E}$ and $\mathcal{F}$ with $\alpha_i \in E_{\mathcal{E}}^{\pm}$ and $\beta_i \in E_{\mathcal{F}}^{\pm}$ for $i=1,\cdots,n$ and let $f: C_n \rightarrow D_n$ be a bijection, satisfying $l_{\mathcal{F}}(f(e)) = l_{\mathcal{E}}(e)$, $f(C_i)=D_i$ and $f(P_i)=Q_i$ for $i=1,\cdots,n$. Since $f(C_i)=D_i$ and $f(P_i)=Q_i$ it follows that $\alpha_i=e^+ \Leftrightarrow \beta_i=f(e)^+$ and $\alpha_i=e^- \Leftrightarrow \beta_i=f(e)^-$ for $i=1,\cdots,n$. Hence $l_{\mathcal{E}}^{\pm}(\alpha_i)=l_{\mathcal{F}}^{\pm}(\beta_i)$ for $1\leqslant i\leqslant n$, and $\alpha_i \prec \alpha_j \Leftrightarrow \beta_i \prec \beta_j$ for $1\leqslant i<j\leqslant n$.

"$\Leftarrow$". Let $(\varnothing,\varnothing) \xrightarrow{\alpha_1}_{\mathcal{E}} \cdots \xrightarrow{\alpha_n}_{\mathcal{E}} (C_n,P_n)$ and $(\varnothing,\varnothing) \xrightarrow{\beta_1}_{\mathcal{F}} \cdots \xrightarrow{\beta_n}_{\mathcal{F}} (D_n,Q_n)$ be chains of ST-configurations in $\mathcal{E}$ and $\mathcal{F}$ with $\alpha_i \in E_{\mathcal{E}}^{\pm}$ and $\beta_i \in E_{\mathcal{F}}^{\pm}$ for $i=1,\cdots,n$ such that $l_{\mathcal{E}}^{\pm}(\alpha_i)=l_{\mathcal{F}}^{\pm}(\beta_i)$ for $1\leqslant i\leqslant n$, and $\alpha_i \prec \alpha_j \Leftrightarrow \beta_i \prec \beta_j$ for $1\leqslant i<j\leqslant n$. Note that $C_i=\{e\in E_{\mathcal{E}} \mid \exists j\leqslant i: \alpha_j=e^+\}$ and $P_i=\{e\in E_{\mathcal{E}} \mid \exists j\leqslant i: \alpha_j=e^-\}$ and similarly for D_i and Q_i. Define $f: C_n \rightarrow D_n$ by $f(e)=d \Leftrightarrow \exists i\leqslant n: (\alpha_i=e^+ \wedge \beta_i=d^+)$. Since $l_{\mathcal{E}}^{\pm}(\alpha_i)=l_{\mathcal{F}}^{\pm}(\beta_i)$ for $1\leqslant i\leqslant n$, f is well-defined and bijective, and satisfies $l_{\mathcal{F}}(f(e)) = l_{\mathcal{E}}(e)$ and $f(C_i)=D_i$ for $i=1,\cdots,n$. Finally $e\in P_i \Leftrightarrow \exists k<j\leqslant i: (\alpha_k=e^+ \wedge \alpha_j=e^-) \Leftrightarrow \exists k<j\leqslant i: (\beta_k=f(e)^+ \wedge$
(using $\alpha_k \prec \alpha_j \Leftrightarrow \beta_k \prec \beta_j$) $\beta_j=f(e)^-) \Leftrightarrow f(e)\in Q_i$ so $f(P_i)=Q_i$ for $i=1,\cdots,n$.

Finally claim 3 follows directly from the definitions. □

ST-bisimulation equivalence will be defined in the same style as the alternative characterization of ST-trace equivalence. The connection of occurrences of a^+ and a^- that represent the beginning and end of the same occurrence of a is implemented by means of a bijection between related ST-configurations.

DEFINITION. Let $\mathcal{E},\mathcal{F}\in\mathbb{E}$. A relation $R\subseteq \mathbb{S}(\mathcal{E})\times\mathbb{S}(\mathcal{F})\times\mathcal{P}(E_{\mathcal{E}}\times E_{\mathcal{F}})$ is called an *ST-bisimulation* between $\mathcal{E}$ and $\mathcal{F}$ if $((\varnothing,\varnothing),(\varnothing,\varnothing),\varnothing)\in R$ and whenever $((C,P),(D,Q),f)\in R$ then:

- $f: C\rightarrow D$ is a bijection, satisfying $l_{\mathcal{F}}(f(e)) = l_{\mathcal{E}}(e)$ and $f(P)=Q$;
- $(C,P) \longrightarrow_{\mathcal{E}} (C',P') \Rightarrow \exists D',Q',f'$ with $(D,Q) \longrightarrow_{\mathcal{F}} (D',Q')$, $((C',P'),(D',Q'),f')\in R$ and $f' \upharpoonright C=f$;
- $(D,Q) \longrightarrow_{\mathcal{F}} (D',Q') \Rightarrow \exists C',P',f'$ with $(C,P) \longrightarrow_{\mathcal{E}} (C',P')$, $((C',P'),(D',Q'),f')\in R$ and $f' \upharpoonright C=f$.

$\mathcal{E}$ and $\mathcal{F}$ are *ST-bisimulation equivalent* - $\mathcal{E}\approx_{STb}\mathcal{F}$ - if there exists an ST-bisimulation between them.

Remark that the same equivalence is obtained if in the definition above the general transition relations $\longrightarrow$ are replaced by the split transition relations $\xrightarrow{a}$ for $a\in Act^{\pm}$. One direction follows from the requirements for the bijection f; the other one follows as in the proof of Proposition 5.2. (Analogously, in the previous chapter it was shown that the definition of history preserving bisimulation equivalence is invariant under replacement of the general transition relations $\longrightarrow$ by the single action transition relations $\xrightarrow{a}$ for $a\in Act$.) Now it is not difficult to show that if in this version of the definition of ST-bisimulation equivalence the requirement $f(P)=Q$ would be skipped, the resulting equivalence would be split bisimulation equivalence again. This requirement ensures the connection of occurrences of a^+ and a^- originating from the same occurrence of a.

As for split equivalences, the ST-equivalences can be defined alternatively by means of split event structures. First some preliminary definitions.

DEFINITION. For $\mathcal{E}\in\mathbb{E}(Act^{\pm})$, define the connectedness relation $\prec_{\mathcal{E}}\subseteq E_{\mathcal{E}}\times E_{\mathcal{E}}$ by

$$e'\prec_{\mathcal{E}}e \text{ iff } l_{\mathcal{E}}(e)=a^{-} \text{ for certain } a\in Act \text{ and for } d\in E_{\mathcal{E}}\text{: } (d<_{\mathcal{E}}e \Leftrightarrow d\leqslant_{\mathcal{E}}e').$$

DEFINITION. Write $C\xrightarrow{e}_{\mathcal{E}}C'$ iff $C\longrightarrow_{\mathcal{E}}C'$ and $C'-C=\{e\}$. A sequence $\alpha_1\cdots\alpha_n\in E_{\mathcal{E}}^{\star}$ is a *pre-trace* of an event structure $\mathcal{E}\in\mathbb{E}(Act^{\pm})$ if there exist configurations $C_0,\cdots,C_n$ of $\mathcal{E}$ such that $C_0=\varnothing$ and $C_{i-1}\xrightarrow{\alpha_i}_{\mathcal{E}}C_i$ $(i=1,\cdots,n)$. Two pre-traces $\alpha_1\cdots\alpha_n$ and $\beta_1\cdots\beta_m$ of $\mathcal{E}$ and $\mathcal{F}$ are *$\prec$-isomorphic* if $n=m$, $l_{\mathcal{E}}(\alpha_i)=l_{\mathcal{F}}(\beta_i)$ for $1\leqslant i\leqslant n$, and $\alpha_i\prec_{\mathcal{E}}\alpha_j\Leftrightarrow\beta_i\prec_{\mathcal{F}}\beta_j$ for $1\leqslant i<j\leqslant n$. A *$\prec$-trace* of $\mathcal{E}$ is the isomorphism class of a pre-trace of $\mathcal{E}$. *$\prec$-Traces*$(\mathcal{E})$ denotes the set of all $\prec$-traces of $\mathcal{E}$. Two event structures $\mathcal{E}$ and $\mathcal{F}\in\mathbb{E}(Act^{\pm})$ are *$\prec$-trace equivalent* - $\mathcal{E}\approx_{\prec t}\mathcal{F}$ - if *$\prec$-Traces*$(\mathcal{E})=$*$\prec$-Traces*$(\mathcal{F})$.

DEFINITION. Let $\mathcal{E},\mathcal{F}\in\mathbb{E}(Act^{\pm})$. A relation $R\subseteq\mathcal{C}(\mathcal{E})\times\mathcal{C}(\mathcal{F})\times\mathcal{P}(E_{\mathcal{E}}\times E_{\mathcal{F}})$ is called a *$\prec$-bisimulation* between $\mathcal{E}$ and $\mathcal{F}$ if $(\varnothing,\varnothing,\varnothing)\in R$ and whenever $(C,D,f)\in R$ then:

- $f:C\rightarrow D$ is a bijection, satisfying $l_{\mathcal{F}}(f(e))=l_{\mathcal{E}}(e)$ and $f(e)\prec_{\mathcal{F}}f(e')\Leftrightarrow e\prec_{\mathcal{E}}e'$;
- $C\longrightarrow_{\mathcal{E}}C'\Rightarrow\exists D',f'$ with $D\longrightarrow_{\mathcal{F}}D'$, $(C',D',f')\in R$ and $f'\upharpoonright C=f$;
- $D\longrightarrow_{\mathcal{F}}D'\Rightarrow\exists C',f'$ with $C\longrightarrow_{\mathcal{E}}C'$, $(C',D',f')\in R$ and $f'\upharpoonright C=f$.

$\mathcal{E}$ and $\mathcal{F}$ are *$\prec$-bisimulation equivalent* - $\mathcal{E}\approx_{\prec b}\mathcal{F}$ - if there exists a $\prec$-bisimulation between them.

PROPOSITION 5.3: $\mathcal{E}\approx_{STt}\mathcal{F}\Leftrightarrow split(\mathcal{E})\approx_{\prec t}split(\mathcal{F})$
$\mathcal{E}\approx_{STb}\mathcal{F}\Leftrightarrow split(\mathcal{E})\approx_{\prec b}split(\mathcal{F})$.

PROOF: For $\mathcal{E}\in\mathbb{E}$ define $i:E_{\mathcal{E}}^{\pm}\rightarrow E_{split(\mathcal{E})}$ by $i(e^{+})=(e,(l_{\mathcal{E}}(e))^{+})$ and $i(e^{-})=(e,(l_{\mathcal{E}}(e))^{-})$. Now the bijections $i_{\mathcal{E}}:\mathbb{S}(\mathcal{E})\rightarrow\mathcal{C}(split(\mathcal{E}))$ from Proposition 4 satisfy for $S\in\mathbb{S}(\mathcal{E})$ and $\alpha\in E_{\mathcal{E}}^{\pm}$:

$$S\xrightarrow{\alpha}_{\mathcal{E}}S'\Leftrightarrow i_{\mathcal{E}}(S)\xrightarrow{i(\alpha)}_{split(\mathcal{E})}i_{\mathcal{E}}(S').$$

Hence $\alpha_1\cdots\alpha_n\in(E_{\mathcal{E}}^{\pm})^{\star}$ (actually $(E_{\mathcal{E}},l_{\mathcal{E}},\sigma)$ with $\sigma=\alpha_1\cdots\alpha_n$) is a pre-ST-trace of an event structure $\mathcal{E}$ iff $i(\alpha_1)\cdots i(\alpha_n)\in E_{split(\mathcal{E})}^{\star}$ is a pre-trace of $split(\mathcal{E})$. Furthermore two pre-ST-traces $\alpha_1\cdots\alpha_n$ and $\beta_1\cdots\beta_m$ of $\mathcal{E}$ are isomorphic iff $i(\alpha_1)\cdots i(\alpha_n)$ and $i(\beta_1)\cdots i(\beta_m)$ are $\prec$-isomorphic. Thus *$\prec$-Traces*$(split(\mathcal{E}))$ is derivable from *ST-Traces*$(\mathcal{E})$ and vice versa. From this the first statement of the proposition follows.

As for the second statement, let $\mathcal{E},\mathcal{F}\in\mathbb{E}$.

$$\mathcal{R}(\mathcal{E},\mathcal{F}) = \{((C,P),(D,Q),f)\in\mathcal{S}(\mathcal{E})\times\mathcal{S}(\mathcal{F})\times\mathcal{P}(E_{\mathcal{E}}\times E_{\mathcal{F}})\mid$$
$$f:C\rightarrow D \text{ is a bijection, satisfying } l_{\mathcal{F}}(f(e)) = l_{\mathcal{E}}(e) \text{ and } f(P)=Q\}.$$

For (S,T,f) and $(S',T',f')\in\mathcal{R}(\mathcal{E},\mathcal{F})$ write $(S,T,f)\longrightarrow(S',T',f')$ if $S\longrightarrow_{\mathcal{E}}S'$, $T\longrightarrow_{\mathcal{F}}T'$, and $f'\upharpoonright C=f$.

$$\mathcal{R}_{split}(\mathcal{E},\mathcal{F}) = \{(C,D,f)\in\mathcal{C}(split(\mathcal{E}))\times\mathcal{C}(split(\mathcal{F}))\times\mathcal{P}(E_{split(\mathcal{E})}\times E_{split(\mathcal{F})})\mid$$
$$f:C\rightarrow D \text{ is a bijection, satisfying } l_{\mathcal{F}}(f(e)) = l_{\mathcal{E}}(e) \text{ and } f(e) \prec_{\mathcal{F}} f(e')$$
$$\Leftrightarrow\ e \prec_{\mathcal{E}} e'\}.$$

For (C,D,f) and $(C',D',f')\in\mathcal{R}_{split}(\mathcal{E},\mathcal{F})$ write $(C,D,f)\longrightarrow(C',D',f')$ if $C\longrightarrow_{split(\mathcal{E})}C'$, $D\longrightarrow_{split(\mathcal{F})}D'$, and $f'\upharpoonright C=f$. Define $i:\mathcal{R}(\mathcal{E},\mathcal{F})\rightarrow\mathcal{R}_{split}(\mathcal{E},\mathcal{F})$ by $i(S,T,f)=(i_{\mathcal{E}}(S),i_{\mathcal{F}}(T),i(f))$ where $i_{\mathcal{E}}$ and $i_{\mathcal{F}}$ are the bijections from Proposition 4 and $i(f):i_{\mathcal{E}}(S)\rightarrow i_{\mathcal{F}}(T)$ is defined by $i(f)(e,a^+)=(f(e),a^+)$ and $i(f)(e,a^-)=(f(e),a^-)$. Now it is not difficult to establish that i is a bijection, satisfying

$$(S,T,f)\longrightarrow(S',T',f')\ \Leftrightarrow\ i(S,T,f)\longrightarrow i(S',T',f').$$

From this it follows that $R\subseteq\mathcal{R}(\mathcal{E},\mathcal{F})$ is an ST-bisimulation between $\mathcal{E}$ and $\mathcal{F}$ iff $i(R)=\{i(S,T,f)\mid(S,T,f)\in R\}\subseteq\mathcal{R}_{split}(\mathcal{E},\mathcal{F})$ is a $\prec$-bisimulation between $split(\mathcal{E})$ and $split(\mathcal{F})$. □

PROPOSITION 5.4: *For all equivalences $\approx_1$ and $\approx_2$ on $\mathbb{E}$ defined so far, the formula*

$$\forall\mathcal{E},\mathcal{F}\in\mathbb{E}:\ \mathcal{E}\approx_1\mathcal{F}\Rightarrow\mathcal{E}\approx_2\mathcal{F}$$

holds iff there is a path $\approx_1\rightarrow\cdots\rightarrow\approx_2$ in Figure 2.

PROOF: In order to prove the announced implications, it suffices to restrict attention to the ones corresponding with an arrow $\approx_1\rightarrow\approx_2$ in Figure 2. Five of them are dealt with in Proposition 3 already. In order to prove the implications $\approx_{2t}\rightarrow\approx_{st}$ and $\approx_{2b}\rightarrow\approx_{sb}$, consider, for $\mathcal{E}\in\mathbb{E}$, the mapping $j:\mathcal{C}(\mathcal{E})\rightarrow\mathcal{S}(\mathcal{E})$ defined by $j(C)=(C,C)$. Note that j is a well-defined injection with $range(j)=\{(C,P)\in\mathcal{S}(\mathcal{E})\mid C=P\}$. Now for $C\in\mathcal{C}(\mathcal{E})$, A a multiset over *act*, and $a_1\cdots a_n\in Act$ an arbitrary enumeration of A, it is easily obtained that

$$\exists C':\ C\xrightarrow{A}_{\mathcal{E}}C'\wedge j(C')=(S,T)\ \Leftrightarrow\ j(C)\xrightarrow{a_1^+}_{\mathcal{E}}\cdots\xrightarrow{a_n^+}_{\mathcal{E}}\xrightarrow{a_1^-}_{\mathcal{E}}\cdots\xrightarrow{a_n^-}_{\mathcal{E}}(S,T).$$

From this the required implications follow immediately. In order to prove the remaining six implications, first consider the implications between equivalences on $\mathbb{E}(Act^{\pm})$ displayed in Figure 8. These implications follow immediately from the definitions. The proofs in Chapter VI that $\approx_{pt}$ and $\approx_h$ are preserved under refinement can be trivially extended to a setting with Λ-refinements for

$$\begin{tikzcd}
\approx_{ib} \arrow[d] & \approx_{<b} \arrow[l] \arrow[d] & \approx_{h} \arrow[l] \arrow[d] \\
\approx_{it} & \approx_{<t} \arrow[l] & \approx_{pt} \arrow[l]
\end{tikzcd}$$

FIGURE 8. *Some semantic equivalences on* $\mathbb{E}(Act^{\pm})$

any labelling set Λ. So it follows that

$$\mathcal{E} \approx_{pt} \mathcal{F} \Rightarrow split(\mathcal{E}) \approx_{pt} split(\mathcal{F}) \text{ and } \mathcal{E} \approx_{h} \mathcal{F} \Rightarrow split(\mathcal{E}) \approx_{h} split(\mathcal{F}).$$

Now the remaining six implications on $\mathbb{E}(Act)$ follow from Propositions 5.1 and 5.3.

In order to prove the absence of other implications, it suffices to provide counterexamples against $\approx_{pt} \rightarrow \approx_{ib}$, $\approx_{ib} \rightarrow \approx_{st}$, $\approx_{sb} \rightarrow \approx_{2t}$, $\approx_{2b} \rightarrow \approx_{STt}$ and $\approx_{STb} \rightarrow \approx_{pt}$. The first two counterexamples where given already in Section 3. For the third counterexample consider the two event structures of Figure 5. In Section 3 it was established already that they are step bisimulation equivalent. Furthermore they are not split trace equivalent, since $a^+ \ c^+ \ a^- \ b^+ \ c^- \ b^-$ is a split trace of the first one but not of the second one.

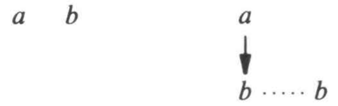


FIGURE 9. *ST-bisimulation equivalent but not pomset trace equivalent (A variant of Example 7.1.2.a.ii of [58]).*

The fourth counterexample will be provided in [60]. For the last counterexample consider the two systems represented in Figure 9. Both systems perform the actions a and b exactly once. In the first system these actions can only be independent, whereas in the second one b can be executed either dependent or independent of a. The difference between the two systems does not occur before (and unless) they reach a state where the execution of a is completed and the execution of b is not yet begun. However, in this state both systems have exactly the same future, consisting of exactly one occurrence of b. Hence they are identified in ST-bisimulation semantics. On the other hand the pomset $a \rightarrow b$ is a pomset trace of the second system, but not of the first. So the two systems are not pomset trace equivalent. This example also shows that ST-semantics does not respect causality. □

6. The refinement theorems

Finally I will prove the announced refinement theorems for ST-semantics. In Van Glabbeek & Vaandrager [60] it will be shown that such a theorem does not hold for split semantics.

Theorem: *Let $\mathcal{E},\mathcal{F}\in\mathbb{E}$ and r be a refinement. Then*

$$\mathcal{E} \approx_{STb} \mathcal{F} \;\Rightarrow\; r(\mathcal{E}) \approx_{STb} r(\mathcal{F}).$$

Proof: Let $R\subseteq\mathbb{S}(E_{\mathcal{E}})\times\mathbb{S}(E_{\mathcal{F}})\times\mathcal{P}(E_{\mathcal{E}}\times E_{\mathcal{F}})$ be an ST-bisimulation between $\mathcal{E}$ and $\mathcal{F}$. Define the relation $\tilde{R}$ by:

$\tilde{R} = \{((\tilde{C},\tilde{P}),(\tilde{D},\tilde{Q}),\tilde{f})\in\mathbb{S}(E_{r(\mathcal{E})})\times\mathbb{S}(E_{r(\mathcal{F})})\times\mathcal{P}(E_{r(\mathcal{E})}\times E_{r(\mathcal{F})})\mid$

$\exists((C,P),(D,Q),f)\in R$ such that $r^{-1}(\tilde{C},\tilde{P})=(C,P)$, $r^{-1}(\tilde{D},\tilde{Q})=(D,Q)$

and $\tilde{f}:\tilde{C}\to\tilde{D}$ is a bijection, satisfying $\tilde{f}(e,e')=(f(e),e')$ and $\tilde{f}(\tilde{P})=\tilde{Q}\,\}$.

I show that $\tilde{R}$ is an ST-bisimulation between $r(\mathcal{E})$ and $r(\mathcal{F})$.

i. $((\varnothing,\varnothing),(\varnothing,\varnothing),\varnothing)\in\tilde{R}$ since $((\varnothing,\varnothing),(\varnothing,\varnothing),\varnothing)\in R$.

ii. Suppose $((\tilde{C},\tilde{P}),(\tilde{D},\tilde{Q}),\tilde{f})\in\tilde{R}$. Take $((C,P),(D,Q),f)\in R$ such that $r^{-1}(\tilde{C},\tilde{P})=(C,P)$, $r^{-1}(\tilde{D},\tilde{Q})=(D,Q)$ and $\tilde{f}:\tilde{C}\to\tilde{D}$ is a bijection, satisfying $\tilde{f}(e,e')=(f(e),e')$ and $\tilde{f}(\tilde{P})=\tilde{Q}$. Now three things have to be established:

1. $\tilde{f}:\tilde{C}\to\tilde{D}$ is a bijection, satisfying $l_{r(\mathcal{F})}(\tilde{f}(e,e')) = l_{r(\mathcal{E})}(e,e')$ and $\tilde{f}(\tilde{P})=\tilde{Q}$.
2. $(\tilde{C},\tilde{P}) \longrightarrow_{r(\mathcal{E})}(\tilde{C}',\tilde{P}') \;\Rightarrow\; \exists\,\tilde{D}',\tilde{Q}',\tilde{f}'$ with $\tilde{f}'\upharpoonright\tilde{C}=\tilde{f}$, $(\tilde{D},\tilde{Q}) \longrightarrow_{r(\mathcal{F})}(\tilde{D}',\tilde{Q}')$ and $((\tilde{C}',\tilde{P}'),(\tilde{D}',\tilde{Q}'),\tilde{f}')\in\tilde{R}$.
3. $(\tilde{D},\tilde{Q}) \longrightarrow_{r(\mathcal{F})}(\tilde{D}',\tilde{Q}') \;\Rightarrow\; \exists\,\tilde{C}',\tilde{P}',\tilde{f}'$ with $\tilde{f}'\upharpoonright\tilde{C}=\tilde{f}$, $(\tilde{C},\tilde{P}) \longrightarrow_{r(\mathcal{E})}(\tilde{C}',\tilde{P}')$ and $((\tilde{C}',\tilde{P}'),(\tilde{D}',\tilde{Q}'),\tilde{f}')\in\tilde{R}$

ad 1. By construction $\tilde{f}:\tilde{C}\to\tilde{D}$ is a bijection, satisfying $\tilde{f}(\tilde{P})=\tilde{Q}$.
Moreover $l_{r(\mathcal{F})}(\tilde{f}(e,e')) = l_{r(\mathcal{F})}(f(e),e') = l_{r(l_{\mathcal{F}}(f(e)))}(e') =$
$= l_{r(l_{\mathcal{E}}(e))}(e') = l_{r(\mathcal{E})}(e,e')$.

ad 2. Suppose $(\tilde{C},\tilde{P}) \longrightarrow_{r(\mathcal{E})}(\tilde{C}',\tilde{P}')$, i.e. $(\tilde{C}',\tilde{P}')\in\mathbb{S}(r(\mathcal{E}))$,

$\tilde{C} \subseteq \tilde{C}'$ and $\tilde{P} \subseteq \tilde{P}'$.

Let $(C',P')=r^{-1}(\tilde{C}', \tilde{P}')$. Using Lemma 4.ii, $(C,P) \longrightarrow_{\mathcal{E}}(C',P')$. Since R is an ST-bisimulation, $\exists D',Q',f'$ with $(D,Q) \longrightarrow_{\mathcal{F}}(D',Q')$, $((C',P'),(D',Q'),f') \in R$ and $f' \restriction C = f$.

Let $\tilde{D}' = \{(f'(e),e') \mid (e,e') \in \tilde{C}'\}$,

$\tilde{Q}' = \{(f'(e),e') \mid (e,e') \in \tilde{P}'\}$ and

$\tilde{f}' = \{((e,e'),(f'(e),e')) \mid (e,e') \in \tilde{C}'\}$.

For $e \in pr_1(\tilde{C}')$ let

$C_e = \{e' \mid (e,e') \in \tilde{C}'\}$ and $P_e = \{e' \mid (e,e') \in \tilde{P}'\}$;

for $d \in pr_1(\tilde{D}')$ let

$D_d = \{e' \mid (d,e') \in \tilde{D}'\}$ and $Q_d = \{e' \mid (d,e') \in \tilde{Q}'\}$.

Remark that $Q_{f'(e)} = \{e' \mid (f'(e),e') \in \tilde{Q}'\} = \{e' \mid (e,e') \in \tilde{P}'\} = P_e$ and similarly $D_{f'(e)} = C_e$.

I prove that $(\tilde{D}, \tilde{Q}) \longrightarrow_{r(\mathcal{F})}(\tilde{D}', \tilde{Q}')$,

$((\tilde{C}', \tilde{P}'),(\tilde{D}', \tilde{Q}'), \tilde{f}') \in \tilde{R}$ and $\tilde{f}' \restriction \tilde{C} = \tilde{f}$.

I start with proving that $(\tilde{D}', \tilde{Q}') \in \mathfrak{S}(r(\mathfrak{F}))$.

$$pr_1(\tilde{D}') = \{f'(e) \mid e \in pr_1(\tilde{C}')\} = f'(C') = D' \qquad (1)$$

so $\tilde{D}' = \{(d,e') \mid d \in D',\ e' \in D_d\}$,

$\tilde{Q}' = \{(d,e') \mid d \in D',\ e' \in Q_d\}$.

Using Lemma 4.i, it is then sufficient to show that

(D',Q') is an ST-configuration of $\mathfrak{F}$,
(D_d,Q_d) is an ST-configuration of $r(l_{\mathfrak{F}}(d))$ for $d \in D'$,
$Q_d = E_{r(l_{\mathfrak{F}}(d))}$ iff $d \in Q'$. (2)

The first requirement is already implicit in $(D,Q) \longrightarrow_{\mathfrak{F}} (D',Q')$.

Since $D' = f'(C')$ one may substitute $f'(e)$ for d and $e \in C'$ for $d \in D'$ in the remaining two requirements.

Since $D_{f'(e)} = C_e$, $Q_{f'(e)} = P_e$, $l_{\mathfrak{F}}(f'(e)) = l_{\mathfrak{E}}(e)$ and $Q' = f'(P')$ they reduce to

(C_e,P_e) is an ST-configuration of $r(l_{\mathfrak{E}}(e))$ for $e \in C'$ and
$P_e = E_{r(l_{\mathfrak{E}}(e))}$ iff $e \in P'$.

These follow from Lemma 4.i, using that $(\tilde{C}', \tilde{P}') \in \mathfrak{S}(r(\mathfrak{E}))$ and $r^{-1}(\tilde{C}', \tilde{P}') = (C',P')$.

Hence $(\tilde{D}', \tilde{Q}') \in \mathfrak{S}(r(\mathfrak{F}))$.

Now (1) and (2) above say that $D' = pr_1(\tilde{D}')$ and $Q' = \{d \in D' \mid Q_d = E_{r(l_{\mathfrak{F}}(d))}\}$. Hence $r^{-1}(\tilde{D}', \tilde{Q}') = (D',Q')$.

It follows that $((\tilde{C}', \tilde{P}'), (\tilde{D}', \tilde{Q}'), \tilde{f}') \in \tilde{R}$.

Finally $\tilde{f}' \restriction \tilde{C} = \tilde{f}$, $\tilde{D} \subseteq \tilde{D}'$ and $\tilde{Q} \subseteq \tilde{Q}'$ by construction, using that $f' \restriction C = f$. With $(\tilde{D}', \tilde{Q}') \in \mathfrak{S}(r(\mathfrak{F}))$, it follows that $(\tilde{D}, \tilde{Q}) \longrightarrow_{r(\mathfrak{F})} (\tilde{D}', \tilde{Q}')$.

ad 3. By symmetry. □

Theorem: *Let $\mathfrak{E}, \mathfrak{F} \in \mathbb{E}$ and r be a refinement. Then*

$$\$\ \mathfrak{E} \approx_{STt} \mathfrak{F} \Rightarrow r(\mathfrak{E}) \approx_{STt} r(\mathfrak{F})\$.$$

Proof: It suffices to proof $\mathfrak{E} \lesssim_{STt} \mathfrak{F} \Rightarrow r(\mathfrak{E}) \lesssim_{STt} r(\mathfrak{F})$, so let $\mathfrak{E}, \mathfrak{F} \in \mathbb{E}$ with $\mathfrak{E} \lesssim_{STt} \mathfrak{F}$ and let r be a refinement. Suppose in $r(\mathfrak{E})$ there is a chain of ST-configurations

$$(\varnothing, \varnothing) \longrightarrow_{r(\mathfrak{E})} (\tilde{C}_1, \tilde{P}_1) \longrightarrow_{r(\mathfrak{E})} \cdots \longrightarrow_{r(\mathfrak{E})} (\tilde{C}_n, \tilde{P}_n).$$

By Lemma 4.ii there is a chain of ST-configurations

$$(\varnothing,\varnothing) \longrightarrow_{\mathcal{E}} (C_1,P_1) \longrightarrow_{\mathcal{E}} \cdots \longrightarrow_{\mathcal{E}} (C_n,P_n)$$

in $\mathcal{E}$ with $(C_i,P_i)=r^{-1}(\tilde{C}_i\ ,\tilde{P}_i\ )$ for $i=1,\cdots,n$. Hence there must be a chain

$$(\varnothing,\varnothing) \longrightarrow_{\mathcal{F}} (D_1,Q_1) \longrightarrow_{\mathcal{F}} \cdots \longrightarrow_{\mathcal{F}} (D_n,Q_n)$$

in $\mathcal{F}$ and a bijection $f: C_n \rightarrow D_n$, satisfying $l_{\mathcal{F}}(f(e)) = l_{\mathcal{E}}(e)$, $f(C_i)=D_i$ and $f(P_i)=Q_i$ for $i=1,\cdots,n$.

Let $\tilde{D}_i\ =\{(f(e),e')\,|\,(e,e')\in \tilde{C}_i\ \}$,

$\tilde{Q}_i\ =\{(f(e),e')\,|\,(e,e')\in \tilde{P}_i\ \}$ and

$\tilde{f}=\{((e,e'),(f(e),e'))\,|\,(e,e')\in \tilde{C}_n\ \}$.

It remains to be shown that

$$(\varnothing,\varnothing) \longrightarrow_{r(\mathcal{F})} (\tilde{D}_1\ ,\tilde{Q}_1\ ) \longrightarrow_{r(\mathcal{F})} \cdots \longrightarrow_{r(\mathcal{F})} (\tilde{D}_n\ ,\tilde{Q}_n\ )$$

is a chain of ST-configurations in $r(\mathcal{F})$ and $\tilde{f}:\tilde{C}_n \rightarrow \tilde{D}_n$ is a bijection satisfying $l_{r(\mathcal{F})}(\tilde{f}(e,e')) = l_{r(\mathcal{E})}(e,e')$, $\tilde{f}(\tilde{C}_i\ )=\tilde{D}_i$ and $\tilde{f}(\tilde{P}_i\ )=\tilde{Q}_i$ for $i=1,\cdots,n$. The only nontrivial part of this consist of proving that $(\tilde{D}_i\ ,\tilde{Q}_i\ )\in\mathcal{S}(r(\mathcal{F}))$ for $i=1,\cdots,n$. This goes exactly as in the previous proof. □

Concluding remarks

In this chapter ten semantic equivalences for concurrent systems are defined on a domain of labelled event structures, and their interdependencies are classified as indicated in Figure 2 of the introduction. It has been established - in [36, 54] and [60] respectively - that interleaving, step and split equivalences are strictly based on action atomicity. In particular, the owl example of [60] shows that no equivalence that can be localized between split bisimulation and interleaving trace equivalence is preserved under refinement of actions. On the other hand it has been shown - in [36] and in the previous Chapter - that the two partial order equivalences of Figure 2 *are* preserved under action refinement and thus need not to be based on action atomicity. Now this chapter added that also ST-trace and ST-bisimulation equivalence are preserved under refinement. So the borderline is between split and ST-semantics.

It should be remarked that at all places where split semantics was used before it was studied for a restricted class of concurrent systems (Petri nets without autoconcurrency in [58], a subset of CCS in [3, 70] and deterministic event structures in [119]) on which it coincides with ST-semantics. The examples of [60] suggest that outside such a class, split semantics is not an interesting notion. The reason for mentioning it in this chapter is that it seems to be a natural simplification of ST-semantics and in order to indicate that for the purposes of this thesis this simplification should not be made.

The refinement operation considered in this chapter replaced actions by finite, conflict-free, non-empty event structures. As remarked earlier, a generalization to infinite refinements, leaving all definitions the same, is incompatible with the principle of finite causes: try to refine a in

$$a \rightarrow b \qquad \text{by} \qquad a_1 \rightarrow a_2 \rightarrow a_3 \rightarrow \quad \cdots .$$

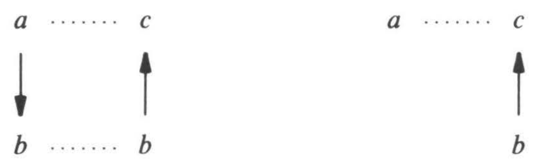
If one would drop this principle, there are (at least) two possibilities of interpreting event structures: events which have an infinite set of causes can happen in a finite time, or they can not. The last interpretation is slightly simpler to grasp, more common, and compatible with the view of this chapter, in which the behaviour of concurrent systems - together with all semantic equivalences - is explained in terms of finite configurations (or ST-configurations) only. Using this interpretation any 'generalized' event structure can be transformed in an ordinary prime event structure satisfying the principle of finite causes, by removing all events that have infinitely many causes. A transformed event structure and its original are equivalent with respect to all equivalences of Figure 2. On the domain of 'generalized' event structures one may drop the restriction that refinements need to be finite, and all theorems and definitions of this chapter remain valid. In fact also all proofs remain valid, since (except in the proof of Proposition 1.i) the principle of finite causes is never used. However, it can be argued that infinite refinements change the behaviour of the considered systems in a way that cannot be explained by a change in the level of abstraction at which processes are regarded: consider a system performing the actions a and b one time each, where the occurrence of b is dependent of the occurrence of a (as depicted above); after replacement of a by an infinite event structure, b cannot happen any more; it occurs in no (finite) configuration. Finally notice that it is also possible to describe this type of refinement on the domain of prime event structures satisfying the principle of finite causes, by adding to the definition of refinement that after refinement in the sense of Section 1, events with infinitely many causes should be left out.

A generalization to refinements containing conflicts can be obtained analogously as the above generalization to infinite refinements, but is technically more complicated. On the domain of prime event structures used in this chapter, refinements with conflicts are incompatible with the principle of conflict heredity: try to replace a in

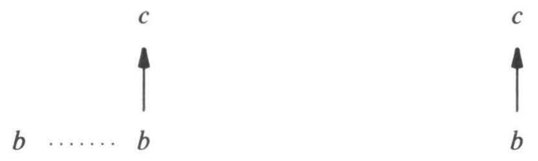
$$a \longrightarrow b \qquad \text{by} \qquad a_1 \cdots\cdots a_2 .$$

This problem has been solved in Chapter IV by moving to a more general form of event structures where the axiom of conflict heredity is dropped, namely flow event structures [32]. On flow event structures we could define a refinement operator for any function $r\!:\!Act \rightarrow \mathbb{E} - \{0\}$, thus allowing both infinite refinements and refinements with conflicts. I expect that all theorems of this chapter remain valid in the setting of flow event structures. Each flow event structure is equivalent to a prime event structure (with respect to any of the equivalences of Figure 2). Hence an alternative solution consists of appending to the definition of refinement some transformation that turns the refined event structure into an equivalent prime event structure.

Contrary to the previous generalization, a generalization of the refinement operator to *forgetful refinements*, where replacing actions by the empty event structure is allowed, does not seem very natural. Such refinements can drastically change the behaviour of concurrent systems and can not be explained by a change in the level of abstraction at which these systems are regarded (Chapter IV). Moreover, unlike the refinement theorems for partial order semantics (Chapter VI) the refinement theorem for ST-bisimulation semantics does not hold for forgetful refinements, as is demonstrated by the following counterexample.



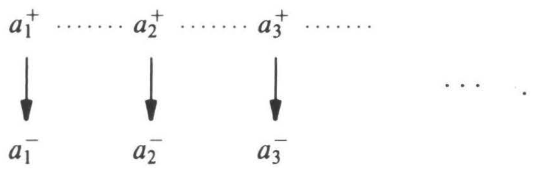
The two event structures above are ST-bisimulation equivalent. However, after replacing a by the empty event structure, the resulting event structures (below) are not ST-bisimulation equivalent.



The refinement theorems for ST-semantics show that in case preservation under refinement is required, it is not necessary to employ partial order semantics. From this the natural question arises if it is necessary to employ at least ST-semantics, i.e. if any equivalence finer then a given interleaving equivalence that is preserved under refinement is also finer then some ST-equivalence. Let $\approx_x$ be an equivalence on $\mathbb{E}$. Define $\approx_{rx}$ by

$$\mathscr{E} \approx_{rx} \mathscr{F} \text{ iff for all refinements } r: Act \to \mathbb{E} - \{0\} \text{ one has } r(\mathscr{E}) \approx_x r(\mathscr{F}).$$

Then, $\approx_{rx}$ is finer then $\approx_x$ and preserved under refinement. Moreover it is coarser then any other equivalence with these properties. In other words, $\approx_{rx}$ is *fully abstract* with respect to $\approx_x$ and refinement. Of course the definition above is parametrized by the concept of refinement. Let $\approx_{rx}$ be defined under reference to general refinements $r: Act \rightarrow \mathbb{E} - \{0\}$ (using flow event structures); and let $\approx_{r'x}$ be defined under reference to refinements as defined in Section 1 of this chapter. Then I conjecture that $\approx_{STb}$ coincides with $\approx_{rib}$, i.e. ST-bisimulation equivalence is fully abstract with respect to interleaving bisimulation equivalence and action refinement, and also $\approx_{STt}$ coincides with $\approx_{rit}$. To be more precise, let r_c be the refinement that replaces actions $a \in Act$ by



Then I think that $\mathcal{E} \approx_{STb} \mathcal{F} \Leftrightarrow r_c(\mathcal{E}) \approx_{ib} r_c(\mathcal{F})$ and likewise $\mathcal{E} \approx_{STt} \mathcal{F} \Leftrightarrow r_c(\mathcal{E}) \approx_{it} r_c(\mathcal{F})$, from which the conjecture follows. Furthermore, together with Walter Vogler I observed that for finite event structures $\approx_{STb}$ even coincides with $\approx_{r'ib}$. On the other hand $\approx_{r'it}$ is strictly coarser then $\approx_{rit}$, as follows from an example in LARSEN [83], see also [60].

In VOGLER [124] a 'failures semantics based on interval semiwords' was presented that can be regarded as the ST-version of failure semantics. He proved that this semantics is preserved under refinement of actions and also established that it is fully abstract with respect to interleaving failure semantics and refinement (allowing refinements with conflicts, but without initial and final parallism, see Section 4 of Chapter IV). The same results he obtained for ST-trace semantics.

Topics for further research include

- generalizing the refinement theorems to a setting with infinite refinements and refinements with conflicts, as in Chapter IV.
- defining 'syntactic refinement' (replacing action symbols by terms in process expressions) on process specification languages, investigating the interaction with communication, proving syntactic refinement theorems and establishing the correspondence with 'semantic refinement', as employed in this chapter (cf. [3, 58, 83, 98]),
- proving the full abstraction results conjectured above,
- proving refinement theorems and full abstraction results for the ST-versions of decorated trace semantics - for failure semantics this has been done already in VOGLER [124] in a setting of Petri nets, and for a variant of trace semantics, in the absence of autoconcurrency, modelling a process as a set of semiwords, this has been done in NIELSEN, ENGBERG & LARSEN [98] and LARSEN [83]
- and generalizing the entire theory to a setting with silent actions, or τ-moves (possibly combining the notions of branching bisimulation (for refinement of systems with silent actions) (Chapter III) and ST-bisimulation or history preserving bisimulation (for refinement of non-

sequential systems)).

PRATT [107] and CASTELLANO, DE MICHELIS & POMELLO [36] use the issue of action atomicity as an argument for using partial order semantics instead of interleaving semantics. This chapter shows that it is not necessary to employ partial order semantics if one does not want to assume action atomicity; ST-semantics turns out to be sufficient. In VAN GLABBEEK & VAANDRAGER [58] we introduced the (related) criterion of *real-time consistency*. A semantics is real-time consistent if it does not identify systems with a different real-time behaviour. Of course interleaving semantics are not real-time consistent, but again the criterion did not force us to consider partial order semantics: also for this purpose ST-bisimulation semantics turned out to be sufficient. Therefore the question remains whether or not there exists a convincing testing scenario, or some natural operator, that reveals the full distinguishing power of partial order semantics.

Samenvatting

In dit proefschrift worden semantieken voor parallelle systemen met elkaar vergeleken. Een systeem is parallel als het verscheidene activiteiten tegelijk kan vertonen. Een semantiek voor parallelle systemen is een criterium dat zegt wanneer twee systemen zich hetzelfde gedragen. Dit is ondermeer van belang voor het correct bewijzen van implementaties van gespecificeerde systemen.

Het proefschrift bevat een introductie en zeven hoofdstukken die alle op afzonderlijke artikelen gebaseerd zijn. Het eerste hoofdstuk bevat een classificatie van semantieken voor een eenvoudig type systemen. Semantieken die in de literatuur voorkomen worden op een uniforme, model-onafhankelijke wijze gepresenteerd. De semantieken worden gemotiveerd met behulp van eenvoudige machinemodellen waarmee men het observeerbare gedrag van systemen kan beschrijven. Voor tien van de semantieken wordt bovendien een complete axiomatizering gegeven.

Hoofdstuk II laat zien hoe semantische begrippen gebruikt kunnen worden in protocolverificaties en andere toepassingen. Dit hoofdstuk wordt geheel in algebraïsche stijl gepresenteerd. Teneinde axiomasystemen te combineren die moeilijk te verenigen semantische noties vertegenwoordigen wordt een nieuwe notie van bewijs geintroduceerd.

Hoofdstuk III introduceert de *vertakkende bisimulatie*, een variant van Milner's 'observatie equivalentie', die de vertakkingsstructuur van systemen beter behoudt. Anders dan observatie equivalentie, blijft de equivalentie behouden onder verfijning van acties (zie verderop) zolang acties niet parallel kunnen plaatsvinden en is zij verenigbaar met modale logica met 'uiteindelijk' operator. Recent onderzoek heeft bovendien uitgewezen dat algorithmen voor het beslissen van vertakkende bisimulatie in practische toepassingen in het algemeen sneller zijn en minder ruimte gebruiken dan de corresponderende algorithmen voor observatie equivalentie.

In het vierde hoofdstuk wordt een operator voor verfijning van acties voorgesteld, en gedefinieerd op drie soorten 'event structures' en op Petri-netten. In 'event structures' en Petri-netten kunnen systemen worden opgebouwd uit bepaalde nog niet nader geïnterpreteerde acties. De verfijningsoperator staat toe om in het ontwerp van parallelle systemen deze acties te vervangen door samengestelde systemen.

In de laatste drie hoofdstukken wordt onderzocht welke semantieken behouden blijven onder actie-verfijning, in die zin dat equivalente systemen equivalent blijven na vervanging van alle voorkomens van bepaalde acties door hun interpretatie op een concreter niveau van abstractie.

References

[1] S. ABRAMSKY (1987): *Observation equivalence as a testing equivalence.* Theoretical Computer Science 53, pp. 225-241.

[2] S. ABRAMSKY & S. VICKERS (1990): *Quantales, observational logic, and process semantics,* Department of Computing Report DOC 90/1, Imperial College.

[3] L. ACETO & M. HENNESSY (1989): *Towards action-refinement in process algebras.* In: Proceedings 4th Annual Symposium on Logic in Computer Science (LICS 89), Asilomar, California, IEEE Computer Society Press, Washington, pp. 138-145.

[4] P. ACZEL (1988): *Non-well-founded sets,* CSLI Lecture Notes No.14, Stanford University.

[5] P. AMERICA (1985): *Definition of the programming language POOL-T.* ESPRIT project 415, Doc. Nr. 91, Philips Research Laboratories, Eindhoven.

[6] D. AUSTRY & G. BOUDOL (1984): *Algèbre de processus et synchronisations.* Theoretical Computer Science 30(1), pp. 91-131.

[7] J.C.M. BAETEN & J.A. BERGSTRA (1988): *Global renaming operators in concrete process algebra.* I&C 78(3), pp. 205-245.

[8] J.C.M. BAETEN, J.A. BERGSTRA & J.W. KLOP (1987): *Conditional axioms and α/β calculus in process algebra.* In: Formal Description of Programming Concepts - III, Proceedings of the third IFIP WG 2.2 working conference, Ebberup 1986 (M. Wirsing, ed.), North-Holland, Amsterdam, pp. 53-75.

[9] J.C.M. BAETEN, J.A. BERGSTRA & J.W. KLOP (1987): *On the consistency of Koomen's fair abstraction rule.* Theoretical Computer Science 51(1/2), pp.

129-176.

[10] J.C.M. BAETEN, J.A. BERGSTRA & J.W. KLOP (1987): *Ready-trace semantics for concrete process algebra with the priority operator.* The Computer Journal 30(6), pp. 498-506.

[11] J.C.M. BAETEN & R.J. VAN GLABBEEK (1987): *Another look at abstraction in process algebra.* In: Proceedings ICALP 87, Karlsruhe (Th. Ottman, ed.), LNCS 267, Springer-Verlag, pp. 84-94.

[12] J.C.M. BAETEN & R.J. VAN GLABBEEK (1987): *Merge and termination in process algebra.* In: Proceedings 7^{th} Conference on Foundations of Software Technology & Theoretical Computer Science, Pune, India (K.V. Nori, ed.), LNCS 287, Springer-Verlag, pp. 153-172.

[13] J.W. DE BAKKER, J.A. BERGSTRA, J.W. KLOP & J.-J.CH. MEYER (1984): *Linear time and branching time semantics for recursion with merge.* Theoretical Computer Science 34, pp. 135-156.

[14] J.W. DE BAKKER, J.N. KOK, J.-J.CH. MEYER, E.-R. OLDEROG & J.I. ZUCKER (1986): *Contrasting themes in the semantics of imperative concurrency.* In: Current trends in concurrency (J.W. de Bakker, W.-P. de Roever & G. Rozenberg, eds.), LNCS 224, Springer-Verlag, pp. 51-121.

[15] J.W. DE BAKKER & J.I. ZUCKER (1982): *Processes and the denotational semantics of concurrency.* I&C 54(1/2), pp. 70-120.

[16] J.A. BERGSTRA (1985): *A process creation mechanism in process algebra.* Logic Group Preprint Series Nr. 2, CIF, State University of Utrecht, to appear in: Applications of process algebra, (J.C.M. Baeten, ed.), Cambridge University Press, 1990, pp. 81-88.

[17] J.A. BERGSTRA, J. HEERING & P. KLINT (1988): *Module algebra (revised version).* Report P8823, Programming Research Group, University of Amsterdam, to appear in: JACM. This report is a revised version of CWI Report CS-R8617, Amsterdam 1986.

[18] J.A. BERGSTRA & J.W. KLOP (1984): *The algebra of recursively defined processes and the algebra of regular processes.* In: Proceedings ICALP 84, Antwerp (J. Paredaens, ed.), LNCS 172, Springer-Verlag, pp. 82-95.

[19] J.A. BERGSTRA & J.W. KLOP (1984): *Process algebra for synchronous communication.* I&C 60(1/3), pp. 109-137.

[20] J.A. BERGSTRA & J.W. KLOP (1985): *Algebra of communicating processes with abstraction.* Theoretical Computer Science 37(1), pp. 77-121.

[21] J.A. BERGSTRA & J.W. KLOP (1988): *A complete inference system for regular processes with silent moves.* In: Proceedings Logic Colloquium 1986 (F.R. Drake & J.K. Truss, eds.), North Holland, Hull, pp. 21-81.

[22] J.A. BERGSTRA & J.W. KLOP (1989): *Process theory based on bisimulation semantics.* In: Proceedings REX School/Workshop on Linear Time, Branching Time and Partial Order in Logics and Models for Concurrency, Noordwijkerhout (J.W. de Bakker, W.-P. de Roever & G. Rozenberg, eds.), LNCS 354, Springer-Verlag, pp. 50-122.

[23] J.A. BERGSTRA, J.W. KLOP & E.-R. OLDEROG (1987): *Failures without chaos: a new process semantics for fair abstraction.* In: Formal Description of Programming Concepts - III, Proceedings of the third IFIP WG 2.2

working conference, Ebberup 1986 (M. Wirsing, ed.), North-Holland, Amsterdam, pp. 77-103.

[24] J.A. BERGSTRA, J.W. KLOP & E.-R. OLDEROG (1988): *Readies and failures in the algebra of communicating processes*. SIAM Journal on Computing 17(6), pp. 1134-1177.

[25] J.A. BERGSTRA & J. TIURYN (1987): *Process algebra semantics for queues*. Fund. Inf. X, pp. 213-224.

[26] J.A. BERGSTRA & J.V. TUCKER (1985): *Top down design and the algebra of communicating processes*. SCP 5(2), pp. 171-199.

[27] E. BEST, R. DEVILLERS, A. KIEHN & L. POMELLO (1989): *Fully concurrent bisimulation*. Technical Report LIT 202, Université Libre de Bruxelles, Laboratoire d' Informatique Theorique.

[28] B. BLOOM, S. ISTRAIL & A.R. MEYER (1988): *Bisimulation can't be traced: preliminary report*. In: Conference Record of the 15^{th} ACM Symposium on Principles of Programming Languages (POPL), San Diego, California, pp. 229-239.

[29] G. BOUDOL (1989): *Atomic actions (note)*. Bulletin of the EATCS 38, pp. 136-144.

[30] G. BOUDOL (1990): *Computations of distributed systems, part 1: flow event structures and flow nets*, Report INRIA Sophia Antipolis, in preparation.

[31] G. BOUDOL & I. CASTELLANI (1987): *On the semantics of concurrency: partial orders and transition systems*. In: Proceedings TAPSOFT 87, Vol. I (H. Ehrig, R. Kowalski, G. Levi & U. Montanari, eds.), LNCS 249, Springer-Verlag, pp. 123-137.

[32] G. BOUDOL & I. CASTELLANI (1989): *Permutation of transitions: an event structure semantics for CCS and SCCS*. In: Proceedings REX School/Workshop on Linear Time, Branching Time and Partial Order in Logics and Models for Concurrency, Noordwijkerhout (J.W. de Bakker, W.-P. de Roever & G. Rozenberg, eds.), LNCS 354, Springer-Verlag, pp. 411-427.

[33] S.D. BROOKES, C.A.R. HOARE & A.W. ROSCOE (1984): *A theory of communicating sequential processes*. JACM 31(3), pp. 560-599.

[34] M.C. BROWNE, E.M. CLARKE & O. GRUMBERG (1988): *Characterizing finite Kripke structures in propositional temporal logic*. Theoretical Computer Science 59(1,2), pp. 115-131.

[35] M. BROY (1987): *Views of queues*. Report MIP-8704, Fakultät für Mathematik und Informatik, Universität Passau.

[36] L. CASTELLANO, G. DE MICHELIS & L. POMELLO (1987): *Concurrency vs Interleaving: an instructive example*. Bulletin of the EATCS 31, pp. 12-15.

[37] E.M. CLARKE & E.A. EMERSON (1981): *Design and synthesis of synchronization skeletons using branching-time temporal logic*. In: Proceedings of the Workshop on Logics of Programs, Yorktown Heights (D. Kozen, ed.), LNCS 131, Springer-Verlag, pp. 52-71.

[38] PH. DARONDEAU (1982): *An enlarged definition and complete axiomatisation of observational congruence of finite processes*. In: Proceedings international symposium on programming: 5th colloquium, Aarhus (M. Dezani-

Ciancaglini & U. Montanari, eds.), LNCS 137, Springer-Verlag, pp. 47-62.

[39] Ph. Darondeau & P. Degano (1989): *About semantic action refinement.* Technical Report TR - 11/89, Dipartimento di Informatica, Università di Pisa, to appear in: Fundamenta Informaticae.

[40] P. Degano, R. De Nicola & U. Montanari (1987): *Observational equivalences for concurrency models.* In: Formal Description of Programming Concepts - III, Proceedings of the third IFIP WG 2.2 working conference, Ebberup 1986 (M. Wirsing, ed.), North Holland, pp. 105-129.

[41] P. Degano, R. De Nicola & U. Montanari (1988): *A distributed operational semantics for CCS based on condition/event systems.* Acta Informatica 26(1/2), pp. 59-91.

[42] R. De Nicola (1987): *Extensional equivalences for transition systems.* Acta Informatica 24, pp. 211-237.

[43] R. De Nicola & M. Hennessy (1984): *Testing equivalences for processes.* Theoretical Computer Science 34, pp. 83-133.

[44] R. De Nicola, U. Montanari & F.W. Vaandrager (1990): *Back and forth bisimulations,* submitted. To appear as CWI report.

[45] R. De Nicola & F.W. Vaandrager (1990): *Three logics for branching bisimulation,* to appear as CWI Report. Extended abstract to appear in: Proceedings 5th Annual Symposium on Logic in Computer Science (LICS 90), Philadelphia, USA, IEEE Computer Society Press, Washington.

[46] T. Denvir, W. Harwood, M. Jackson & M. Ray (1985): *The analysis of concurrent systems, Proceedings of a Tutorial and Workshop, Cambridge University 1983,* LNCS 207, Springer-Verlag.

[47] J. Desel & A. Merceron (1989): *Vicinity respecting net morphisms.* In: Proceedings of the 10th International Conference on Petri Nets, Bonn, pp. 115-138.

[48] R. Devillers (1988): *On the definition of a bisimulation notion based on partial words.* Petri Net Newsletter 29, Gesellschaft für Informatik, Bonn, pp. 16-19.

[49] E.A. Emerson & J.Y. Halpern (1986): *'Sometimes' and 'Not Never' revisited: on branching time versus linear time temporal logic.* JACM 33(1), pp. 151-178.

[50] H.J. Genrich & E. Stankiewicz-Wiechno (1980): *A dictionary of some basic notions of Petri nets.* In: Net Theory and Applications, Proceedings advanced course on general net theory of processes and systems, Hamburg 1979 (W. Brauer, ed.), LNCS 84, Springer-Verlag, pp. 519-535.

[51] J.L. Gischer (1988): *The equational theory of pomsets.* Theoretical Computer Science 61, pp. 199-224.

[52] R.J. van Glabbeek (1987): *Bounded nondeterminism and the approximation induction principle in process algebra.* In: Proceedings STACS 87 (F.J. Brandenburg, G. Vidal-Naquet & M. Wirsing, eds.), LNCS 247, Springer-Verlag, pp. 336-347.

[53] R.J. van Glabbeek (1990): *The refinement theorem for ST-bisimulation semantics.* Report CS-R9002, Centrum voor Wiskunde en Informatica, Amsterdam, to appear in: Proceedings IFIP Working Conference on

Programming Concepts and Methods, Sea of Gallilee, Israel 1990 (M. Broy & C.B. Jones, eds.), North-Holland, 1990.

[54] R.J. VAN GLABBEEK & U. GOLTZ (1989): *Equivalence notions for concurrent systems and refinement of actions.* Arbeitspapiere der GMD 366, Sankt Augustin, extended abstract in: Proceedings 14th Symposium on Mathematical Foundations of Computer Science (MFCS), Porąbka-Kozubnik, Poland, August/September 1989 (A. Kreczmar & G. Mirkowska, eds.), LNCS 379, Springer-Verlag, pp. 237-248.

[55] R.J. VAN GLABBEEK & U. GOLTZ (1989): *Partial order semantics for refinement of actions - neither necessary nor always sufficient but appropriate when used with care.* Bulletin of the EATCS 38, pp. 154-163.

[56] R.J. VAN GLABBEEK & J.J.M.M. RUTTEN (1989): *The processes of De Bakker and Zucker represent bisimulation equivalence classes.* In: J.W. de Bakker, 25 jaar semantiek, liber amicorum, pp. 243-246.

[57] R.J. VAN GLABBEEK, S.A. SMOLKA, B. STEFFEN & C.M.N. TOFTS (1990): *Reactive, generative, and stratified models of probabilistic processes,* to appear in: Proceedings 5th Annual Symposium on Logic in Computer Science (LICS 90), Philadelphia, USA, IEEE Computer Society Press, Washington.

[58] R.J. VAN GLABBEEK & F.W. VAANDRAGER (1987): *Petri net models for algebraic theories of concurrency.* In: Proceedings PARLE conference, Eindhoven, Vol. II (Parallel Languages) (J.W. de Bakker, A.J. Nijman & P.C. Treleaven, eds.), LNCS 259, Springer-Verlag, pp. 224-242.

[59] R.J. VAN GLABBEEK & F.W. VAANDRAGER (1988): *Modular specifications in process algebra - with curious queues.* Report CS-R8821, Centrum voor Wiskunde en Informatica, Amsterdam, under revision for TCS. An extended abstract appeared in: Algebraic Methods: Theory, Tools and Applications (M. Wirsing & J.A. Bergstra, eds.), LNCS 394, Springer-Verlag, pp. 465-506.

[60] R.J. VAN GLABBEEK & F.W. VAANDRAGER (1990): *The difference between splitting in n and $n+1$,* in preparation.

[61] R.J. VAN GLABBEEK & W.P. WEIJLAND (1989): *Branching time and abstraction in bisimulation semantics (extended abstract).* In: Information Processing 89 (G.X. Ritter, ed.), North Holland, pp. 613-618.

[62] R.J. VAN GLABBEEK & W.P. WEIJLAND (1989): *Refinement in branching time semantics.* Report CS-R8922, Centrum voor Wiskunde en Informatica, Amsterdam, also appeared in: Proceedings AMAST Conference, May 1989, Iowa, USA, pp. 197-201.

[63] R. GORRIERI, S. MARCHETTI & U. MONTANARI (1988): *A^2CCS: a simple extension of CCS for handling atomic actions.* In: Proceedings CAAP 88, Nancy, France (M. Daughet & M. Nivat, eds.), LNCS 299, Springer-Verlag, pp. 258-270.

[64] J. GRABOWSKI (1981): *On partial languages.* Fundamenta Informaticae IV(2), pp. 427-498.

[65] S. GRAF & J. SIFAKIS (1987): *Readiness semantics for regular processes with silent actions.* In: Proceedings ICALP 87, Karlsruhe (Th. Ottman, ed.),

LNCS 267, Springer-Verlag, pp. 115-125.

[66] E.P. GRIBOMONT (1989): *Stepwise refinement and concurrency: a small exercise.* In: Mathematics of program construction (J.L.A. van de Snepscheut, ed.), LNCS 375, Springer-Verlag, pp. 219-238.

[67] J.F. GROOTE & F.W. VAANDRAGER (1988): *Structured operational semantics and bisimulation as a congruence.* Report CS-R8845, Centrum voor Wiskunde en Informatica, Amsterdam, under revision for I&C. An extended abstract appeared in: Proceedings ICALP 89, Stresa (G. Ausiello, M. Dezani-Ciancaglini & S. Ronchi Della Rocca, eds.), LNCS 372, Springer-Verlag, pp. 423-438.

[68] J.F. GROOTE & F.W. VAANDRAGER (1990): *An efficient algorithm for branching bisimulation and stuttering equivalence.* Report CS-R9001, Centrum voor Wiskunde en Informatica, Amsterdam, to appear in: Proceedings ICALP 90, Warwick, UK, LNCS, Springer-Verlag.

[69] M. HENNESSY (1985): *Acceptance trees.* JACM 32(4), pp. 896-928.

[70] M. HENNESSY (1988): *Axiomatising finite concurrent processes.* SIAM Journal on Computing 17(5), pp. 997-1017.

[71] M. HENNESSY & R. MILNER (1980): *On observing nondeterminism and concurrency.* In: Proceedings ICALP 80 (J. de Bakker & J. van Leeuwen, eds.), LNCS 85, Springer-Verlag, pp. 299-309, a preliminary version of:

[72] M. HENNESSY & R. MILNER (1985): *Algebraic laws for nondeterminism and concurrency.* JACM 32(1), pp. 137-161.

[73] M. HENNESSY & G.D. PLOTKIN (1980): *A term model for CCS.* In: Proceedings 9^{th} Symposium on Mathematical Foundations of Computer Science (MFCS) (P. Dembiński, ed.), LNCS 88, Springer-Verlag, pp. 261-274.

[74] C.A.R. HOARE (1980): *Communicating sequential processes.* In: On the construction of programs - an advanced course (R.M. McKeag & A.M. Macnaghten, eds.), Cambridge University Press, pp. 229-254.

[75] C.A.R. HOARE (1985): *Communicating sequential processes*, Prentice-Hall International.

[76] HE JIFENG & C.A.R. HOARE (1987): *Algebraic specification and proof of a distributed recovery algorithm.* Distributed Computing 2(1), pp. 1-12.

[77] B. JONSSON & J. PARROW (1989): *Deciding bisimulation equivalences for a class of non-finite-state programs.* In: Proceedings STACS 89, Paderborn (B. Monien & R. Cori, eds.), LNCS 347, Springer-Verlag.

[78] J.K. KENNAWAY (1981): *Formal semantics of nondetermism and parallelism.* Ph.D. Thesis, University of Oxford.

[79] W. KORCZYŃSKI (1988): *An algebraic characterization of concurrent systems.* Fundamenta Informaticae 11(2), pp. 171-194.

[80] C.P.J. KOYMANS & J.C. MULDER (1986): *A modular approach to protocol verification using process algebra.* Logic Group Preprint Series Nr. 6, CIF, State University of Utrecht, to appear in: Applications of process algebra, (J.C.M. Baeten, ed.), Cambridge University Press, 1990, pp. 261-306.

[81] L. LAMPORT (1983): *What good is temporal logic?.* In: Information Processing 83 (R.E. Mason, ed.), North Holland, pp. 657-668.

[82] L. Lamport (1986): *On interprocess communication.* Distributed Computing 1, pp. 77-101.

[83] K.S. Larsen (1988): *A fully abstract model for a process algebra with refinements.* Master Thesis, Aarhus University, Denmark.

[84] K.G. Larsen & R. Milner (1987): *A complete protocol verification using relativized bisimulation.* In: Proceedings ICALP 87, Karlsruhe (Th. Ottmann, ed.), LNCS 267, Springer-Verlag, pp. 126-135.

[85] K.G. Larsen & A. Skou (1988): *Bisimulation through probabilistic testing.* R 88-29, Institut for Elektroniske Systemer, Afdeling for Matematik og Datalogi, Aalborg Universitetscenter, a preliminary report appeared in: Conference Record of the 16th Annual ACM Symposium on Principles of Programming Languages (POPL), Austin, Texas, ACM Press, New York 1989.

[86] S. Mauw (1987): *An algebraic specification of process algebra, including two examples.* Report FVI 87-06, Dept. of Computer Science, University of Amsterdam, extended abstract in: Algebraic Methods: Theory, Tools and Applications (M. Wirsing & J.A. Bergstra, eds.), LNCS 394, Springer-Verlag, pp. 507-554.

[87] S. Mauw & G.J. Veltink (1988): *A process specification formalism.* Report P8814, Programming Research Group, University of Amsterdam, to appear in: Fundamenta Informaticae.

[88] J. Meseguer & U. Montanari (1988): *Petri nets are monoids: a new algebraic foundation for net theory.* In: Proceedings 3th Annual Symposium on Logic in Computer Science (LICS 88), Edinburgh, IEEE Computer Society Press, Washington, pp. 155-164.

[89] A.R. Meyer (1985): *Report on the 5th international workshop on the semantics of programming languages in Bad Honnef.* Bulletin of the EATCS 27, pp. 83-84.

[90] G.J. Milne (1985): *CIRCAL and the representation of communication, concurrency, and time.* TOPLAS 7(2), pp. 270-298.

[91] R. Milner (1980): *A Calculus of Communicating Systems*, LNCS 92, Springer-Verlag.

[92] R. Milner (1981): *A modal characterisation of observable machine-behaviour.* In: Proceedings CAAP 81 (G. Astesiano & C. Böhm, eds.), LNCS 112, Springer-Verlag, pp. 25-34.

[93] R. Milner (1983): *Calculi for synchrony and asynchrony.* Theoretical Computer Science 25, pp. 267-310.

[94] R. Milner (1985): *Lectures on a Calculus for Communicating Systems.* In: Seminar on Concurrency (S.D. Brookes, A.W. Roscoe & G. Winskel, eds.), LNCS 197, Springer-Verlag, pp. 197-220.

[95] R. Milner (1989): *Communication and concurrency*, Prentice-Hall International.

[96] F. Moller (1989): *Axioms for concurrency.* Ph.D. Thesis, Report CST-59-89, Department of Computer Science, University of Edinburgh.

[97] J.D. Monk (1976): *Mathematical logic*, Springer-Verlag.

[98] M. Nielsen, U. Engberg & K.S. Larsen (1989): *Fully abstract models*

for a process language with refinement. In: Proceedings REX School/Workshop on Linear Time, Branching Time and Partial Order in Logics and Models for Concurrency, Noordwijkerhout (J.W. de Bakker, W.-P. de Roever & G. Rozenberg, eds.), LNCS 354, Springer-Verlag, pp. 523-548.

[99] M. NIELSEN, G.D. PLOTKIN & G. WINSKEL (1981): *Petri nets, event structures and domains, part I*. Theoretical Computer Science 13(1), pp. 85-108.

[100] E.-R. OLDEROG, U. GOLTZ & R.J. VAN GLABBEEK (1988): *Combining compositionality and concurrency, summary of a GMD-workshop, Königswinter, March 1988*. Arbeitspapiere der GMD 320, Sankt Augustin.

[101] E.-R. OLDEROG & C.A.R. HOARE (1986): *Specification-oriented semantics for communicating processes*. Acta Informatica 23, pp. 9-66.

[102] D.M.R. PARK (1981): *Concurrency and automata on infinite sequences*. In: Proceedings 5th GI Çonference (P. Deussen, ed.), LNCS 104, Springer-Verlag, pp. 167-183.

[103] C.A. PETRI (1977): *Non-sequential processes*. Interner Bericht ISF-77-05, Gesellschaft für Mathematik und Datenverarbeitung, Sankt Augustin.

[104] I.C.C. PHILLIPS (1987): *Refusal testing*. Theoretical Computer Science 50, pp. 241-284.

[105] A. PNUELI (1985): *Linear and branching structures in the semantics and logics of reactive systems*. In: Proceedings ICALP 85, Nafplion (W. Brauer, ed.), LNCS 194, Springer-Verlag, pp. 15-32.

[106] L. POMELLO (1986): *Some equivalence notions for concurrent systems. An overview*. In: Advances in Petri Nets 1985 (G. Rozenberg, ed.), LNCS 222, Springer-Verlag, pp. 381-400.

[107] V.R. PRATT (1986): *Modelling concurrency with partial orders*. International Journal of Parallel Programming 15(1), pp. 33-71.

[108] A. RABINOVICH & B.A. TRAKHTENBROT (1988): *Behavior Structures and Nets*. Fundamenta Informaticae 11(4), pp. 357-404.

[109] W. REISIG (1985): *Petri nets - an introduction*, EATCS Monographs on Theoretical Computer Science, Volume 4, Springer-Verlag.

[110] W. REISIG (1987): *Petri nets in software engineering*. In: Petri Nets: Applications and Relationships to Other Models of Concurrency, Advances in Petri Nets 1986, Part II; Proceedings of an Advanced Course, Bad Honnef, September 1986 (W. Brauer, W. Reisig & G. Rozenberg, eds.), LNCS 255, Springer-Verlag, pp. 63-96.

[111] W.C. ROUNDS & S.D. BROOKES (1981): *Possible futures, acceptances, refusals and communicating processes*. In: Proceedings 22nd Annual Symposium on Foundations of Computer Science, Nashville, USA 1981, IEEE, New York, pp. 140-149.

[112] D.T. SANNELLA & A. TARLECKI (1988): *Toward formal development of programs from algebraic specifications: implementations revisited*. Acta Informatica 25, pp. 233-281.

[113] D.T. SANNELLA & M. WIRSING (1983): *A kernel language for algebraic specification and implementation (extended abstract)*. In: Proceedings

International Conference on Foundations of Computation Theory, Borgholm (M. Karpinski, ed.), LNCS 158, pp. 413-427, long version: Report CSR-131-83, Dept. of Computer Science, Univ. of Edinburgh, 1983.

[114] I. SUZUKI & T. MURATA (1983): *A method for stepwise refinement and abstraction of Petri nets.* Journal of Computer and System Sciences 27(1), pp. 51-76.

[115] D.A. TAUBNER & W. VOGLER (1989): *Step failure semantics and a complete proof system.* Acta Informatica 27, pp. 125-156.

[116] F.W. VAANDRAGER (1986): *Verification of two communication protocols by means of process algebra.* Report CS-R8608, Centrum voor Wiskunde en Informatica, Amsterdam.

[117] F.W. VAANDRAGER (1986): *Process algebra semantics of POOL.* Report CS-R8629, Centrum voor Wiskunde en Informatica, Amsterdam, to appear in: Applications of process algebra, (J.C.M. Baeten, ed.), Cambridge University Press, 1990, pp. 173-236.

[118] F.W. VAANDRAGER (1988): *Some observations on redundancy in a context.* Report CS-R8812, Centrum voor Wiskunde en Informatica, Amsterdam, to appear in: Applications of process algebra, (J.C.M. Baeten, ed.), Cambridge University Press, 1990, pp. 237-260.

[119] F.W. VAANDRAGER (1988): *Determinism* $\rightarrow$ *(event structure isomorphism = step sequence equivalence).* Report CS-R8839, Centrum voor Wiskunde en Informatica, Amsterdam, to appear in: Theoretical Computer Science.

[120] F.W. VAANDRAGER (1989): personal communication.

[121] F.W. VAANDRAGER (1990): *Algebraic techniques for concurrency and their application.* Ph.D. Thesis, University of Amsterdam.

[122] R. VALETTE (1979): *Analysis of Petri nets by stepwise refinements.* Journal of Computer and System Sciences 18, pp. 35-46.

[123] W. VOGLER (1987): *Behaviour preserving refinements of Petri nets.* In: Proceedings 12^{th} International Workshop on Graph-Theoretic Concepts in Computer Science, Bernried, 1986 (G. Tinhofer & G. Schmidt, eds.), LNCS 246, Springer-Verlag, pp. 82-93.

[124] W. VOGLER (1989): *Failures semantics based on interval semiwords is a congruence for refinement.* Bericht TUM-I8905, Institut für Informatik, Technische Universität München, to appear in: Proceedings STACS 90, LNCS, Springer-Verlag.

[125] D.J. WALKER (1990): *Bisimulation and divergence.* I&C 85(2), pp. 202-241.

[126] W.P. WEIJLAND (1989): *Synchrony and asynchrony in process algebra.* Ph.D. Thesis, University of Amsterdam.

[127] G. WINSKEL (1987): *Event structures.* In: Petri Nets: Applications and Relationships to Other Models of Concurrency, Advances in Petri Nets 1986, Part II; Proceedings of an Advanced Course, Bad Honnef, September 1986 (W. Brauer, W. Reisig & G. Rozenberg, eds.), LNCS 255, Springer-Verlag, pp. 325-392.

Propositions

Added to the thesis

Comparative concurrency semantics and refinement of actions

R.J. van Glabbeek

May 16, 1990

1. Let $x \in E \subseteq \mathbb{R}^n$, E open, $V \subseteq \mathbb{R}^n$ and $f: E \rightarrow V$ a two times continuous differentiable bijection. Then any sufficiently small open ball around x in E is mapped on a convex subset of V.

   From this it follows in an elementary way that every open covering of a paracompact differentiable manifold has a refinement $\mathfrak{U} = \{U_i\}_{i \in I}$, with the property that each non-empty finite intersection $U_{i_0} \cap \cdots \cap U_{i_k}$ is diffeomorphic with $\mathbb{R}^n$.

   See: R.J. van Glabbeek, *Good coverings*, Report nr. 3, Mathematical Institute, University of Leiden, The Netherlands 1985.

2. In the following formulation, Craig's interpolation theorem holds for equational logic:
   If β is an equation and A a set of equations such that $A \vdash \beta$, then there exists a finite set I of equations, the signature of which is contained in that of A and that of β, such that $A \vdash I$ and $I \vdash \beta$.

   See: P.H. Rodenburg & R.J. van Glabbeek, *An interpolation theorem in equational logic*, Report CS-R8838, Centre for Mathematics and Computer Science, Amsterdam 1988.

3. Consider the language with CCS operators 0, action-prefixing and +, and a parallel composition operator $\|$ without synchronization. A closed term P is *prime* (up to a semantic equivalence $\approx$) if $P \not\approx 0$ and $P \approx Q \| R$ implies $R \approx 0$ or $Q \approx 0$. R. Milner proved that any closed term in this language can be expressed uniquely, up to (interleaving) bisimulation equivalence, as the parallel composition of a set of primes. The following example shows that this *unique decomposition theorem* for bisimulation semantics does not generalize to failure semantics.

   $$(a + aa) \| (a + aa) = a \| (a + aa + aaa).$$

   In fact this example works for all semantics from completed trace semantics to ready trace semantics in Figure 1 of Chapter I of this thesis.

   See: R. Milner & F. Moller, *Unique decomposition of processes*, to appear in Bulletin of the EATCS 41, 1990.

4. Consider the process modules REC, AIP and KFAR, as defined in Chapter II of this thesis, let BPA* be the submodule of ACP$_\tau$ + PR consisting of the axioms A, T, TI, and PR, and let CA (the *commutativity of abstraction*) be the module $\tau_{\{a\}} \circ \tau_{\{b\}} = \tau_{\{b\}} \circ \tau_{\{a\}}$. All these modules are valid in various models of concurrency. However, their combination is inconsistent in the sense that it doesn't respect deadlock behaviour:

   $$\text{BPA}^* + \text{REC} + \text{AIP} + \text{CA} + \text{KFAR} \vdash \tau = \tau + \tau\delta.$$

See: R.J. VAN GLABBEEK, *Bounded nondeterminism and the approximation induction principle in process algebra.* In: Proceedings STACS 87 (F.J. Brandenburg, G. Vidal-Naquet & M. Wirsing, eds.), LNCS 247, Springer-Verlag, pp. 336-347, 1987.

5. When assuming maximal parallelism, ST-failure trace equivalence and all finer equivalences are real-time consistent, whereas step equivalences and ST-readiness equivalence are not (using the combined terminology of Chapters I and VII of this thesis).

   Real-time consistency was defined in: R.J. VAN GLABBEEK & F.W. VAANDRAGER, *Petri net models for algebraic theories of concurrency.* In: Proceedings PARLE conference, Eindhoven, Vol II (Parallel Languages) (J.W. de Bakker, A.J. Nijman & P.C. Treleaven, eds.), LNCS 259, Springer-Verlag, pp. 224-242, 1987.

6. The use of 'we' in sentences like 'We will now prove our main theorem' in scientific publications with only one author can be interpreted in only a few ways:
   (i) The author (mistakenly) expects his audience to join him in proving his main theorem;
   (ii) The author claims to be royal;
   (iii) The author tries to shift the scientific responsibility for his theorem on the professional community as a whole - for instance in order to increase his authority on the subject and/or to display humbleness by sharing his result with others.
   I prefer the use of 'I'.

   Compare: A. RAND, *Anthem*, New American Library.

7. In social and political disputes the preferable position lies in the middle about as often as mountain-tops can be found half-way up a slope.