Centrum voor Wiskunde en Informatica
Centre for Mathematics and Computer Science

R.J. van Glabbeek

The linear time - branching time spectrum

Computer Science/Department of Software Technology　　Report CS-R9029　　July

The Centre for Mathematics and Computer Science is a research institute of the Stichting Mathematisch Centrum, which was founded on February 11, 1946, as a nonprofit institution aiming at the promotion of mathematics, computer science, and their applications. It is sponsored by the Dutch Government through the Netherlands Organization for the Advancement of Research (N.W.O.).

The Linear Time - Branching Time Spectrum

R.J. van Glabbeek
Centre for Mathematics and Computer Science
P.O. Box 4079, 1009 AB Amsterdam, The Netherlands

In this paper 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.

1980 Mathematics Subject Classification (Zentralblatt für Mathematik): 68B10.
1985 Mathematics Subject Classification (Mathematical Reviews): 68Q55.
1990 CR Classification scheme (Computing Reviews): F.3.2.
Key Words & Phrases: Concurrency, Linear time, Branching time, Button pushing experiments, Generative machines, Reactive machines, Bisimulation semantics, Trace semantics, Failure semantics, Failure trace semantics, Readiness semantics, Ready trace semantics, Simulation semantics, Ready simulation semantics, Nondeterminism, Complete axiomatizations.
Notes: The research reported in this paper has been initiated at the Centre for Mathematics and Computer Science, and finalized at the Technical University of Munich (Institut für Informatik der Technischen Universität, Arcisstraße 21, D-8000 München 2). It has been supported by Sonderforschungsbereich 342 of the TU München. Part of it was carried out in the preparation of a course Comparative Concurrency Semantics, given at the University of Amsterdam, spring 1988.
This paper appeared before as Chapter I of my Ph.D Thesis *Comparative concurrency semantics and refinement of actions*, Free University, Amsterdam 1990. It also appears as SFB-Bericht Nr. 342/../90 A, Institut für Informatik, Technische Universität München, 1990. An extended abstract will be published in: Proceedings CONCUR 90 (J.C.M. Baeten & J.W. Klop, eds.), LNCS, Springer-Verlag, 1990.

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

Report CS-R9029
Centre for Mathematics and Computer Science
P.O. Box 4079, 1009 AB Amsterdam, The Netherlands

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 paper 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 paper, 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.

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,

1. This field of research is called *comparative concurrency semantics*, a terminology first used by MEYER in [24].
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 [5].
4. Strictly speaking processes do not perform actions, but systems do. However, for reasons of convenience, this paper sometimes uses the word process, when actually referring to a system of which the process is the behaviour.

- 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 paper 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 paper.

Finitely branching, concrete, sequential processes. Being a first step, this paper 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 [21]). 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). The coarsest one (i.e. the semantics making the most identifications) is *trace semantics*, as presented in HOARE [20]. 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 [27]. Bisimulation semantics is the standard semantics for the system description language CCS (MILNER [25]). The notion of bisimulation was introduced in PARK [29]. Bisimulation equivalence is a refinement of *observational equivalence*, as introduced by HENNESSY & MILNER in [17]. On the domain of finitely branching, concrete, sequential processes, both equivalences coincide. Also the semantics of DE BAKKER & ZUCKER, presented in [6], 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 paper it is called *completed trace semantics*. *Failure semantics* is introduced in BROOKES, HOARE & ROSCOE [9], and used in the construction of a model for the system description language CSP (HOARE [19,21]). It is finer than completed trace semantics. The semantics based on *testing equivalences*, as developed in DE NICOLA & HENNESY [12], coincides with failure semantics on the domain of finitely branching, concrete, sequential processes, as do the semantics of KENNAWAY [22] and DARONDEAU [10]. This has been established in DE NICOLA [11]. In OLDEROG & HOARE [28] *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 [31] (there called *barbed semantics*), BAETEN, BERGSTRA & KLOP [4] and POMELLO [32] (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 [30]. *Simulation equivalence*, based on the classical notion of *simulation* (see e.g. PARK [29]), is independent of the last five semantics. *Ready simulation semantics* was introduced in BLOOM, ISTRAIL & MEYER [8] under the name *GSOS trace congruence*. It is finer than ready trace as well as simulation equivalence. In LARSEN & SKOU [23] a more operational characterization of this equivalence was given under the name *⅔-bisimulation equivalence*. This characterization resembles the one used in this paper. Finally *2-nested simulation equivalence*, introduced in GROOTE & VAANDRAGER [15], is located between ready simulation and bisimulation equivalence, and *possible-futures semantics*, as proposed in ROUNDS & BROOKES [33], can be positioned between 2-nested simulation

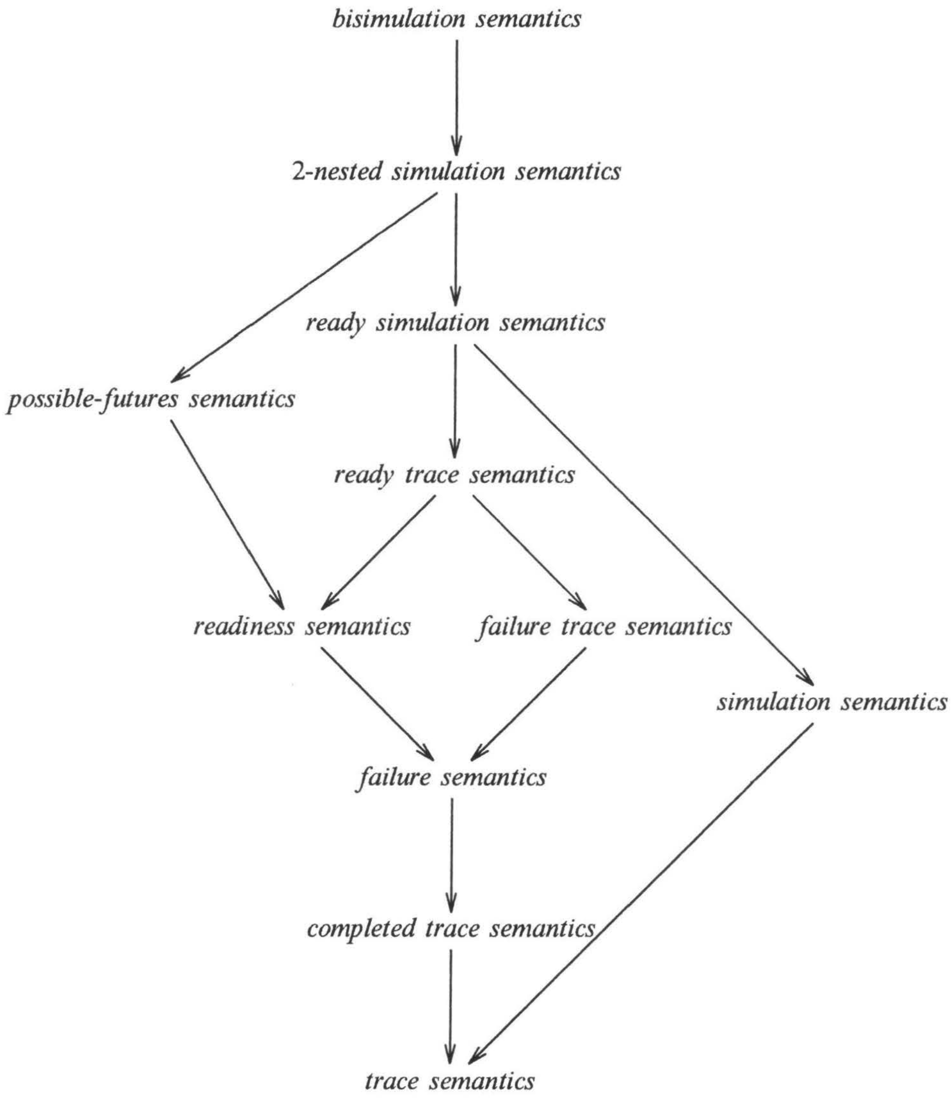


FIGURE 1. *The linear time - branching time spectrum*

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 [14] and semantics that make almost no identifications and are hardly used for system verification.

About the contents. In the first section of this paper 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 [29]. Finally, in Section 3, 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.

1. SEMANTIC EQUIVALENCES ON LABELLED TRANSITION SYSTEMS

1.1. Labelled transition systems. In this paper 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 paper 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 $\mathbb{A}$ on which infix written binary predicates $\xrightarrow{a}$ are defined for each action $a \in Act$. The elements of $\mathbb{A}$ 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 $(\mathbb{A}, \rightarrow)$ with $\mathbb{A}$ a class and $\rightarrow \subseteq \mathbb{A} \times Act \times \mathbb{A}$, such that for $p \in \mathbb{A}$ and $a \in Act$ the class $\{q \in \mathbb{A} \mid p \xrightarrow{\sigma} q\}$ is a set.

Let for the remainder of this section $(\mathbb{A}, \rightarrow)$ 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 $p \in \mathbb{A}$ is *finitely branching* if for each $q \in \mathbb{A}$ with $p \xrightarrow{\sigma} q$ for some $\sigma \in Act^*$, the set $\{(a,r) \mid q \xrightarrow{a} r,\ a \in Act,\ r \in \mathbb{A}\}$ is finite.

In the following, several semantic equivalences on $\mathbb{A}$ 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 [25], 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*.

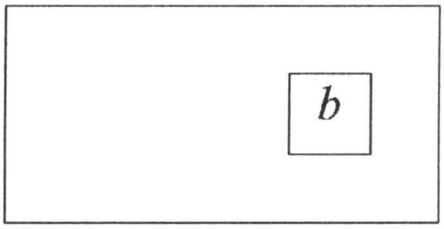


FIGURE 2. *The completed trace machine*

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 choses an execution path that is consistent with its position in the labelled transition system $(\mathbb{A},\rightarrow)$. During this execution always an action 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). 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 choses an execution path that fits with its position in $(\mathbb{A},\rightarrow)$, 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

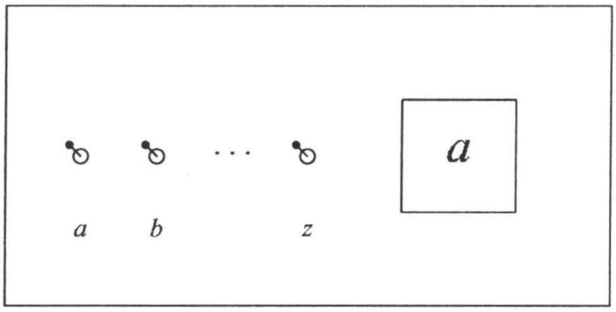


FIGURE 3. *The failure trace machine*

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 [21]. In BROOKES, HOARE & ROSCOE [9], 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 [11] several equivalences, that were proposed in KENNAWAY [22], DARONDEAU [10] and DE NICOLA & HENNESY [12], 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* $X \Leftrightarrow (\sigma, X) \notin F(p)$ [11]. □

In HENNESSY [16], 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 $a \in Act$. 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 *Act*, the actions representing information obtained

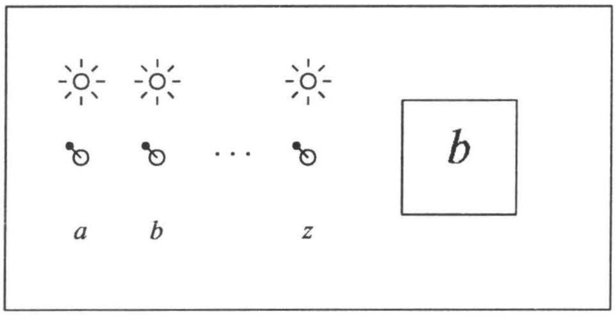


FIGURE 4. *The ready trace machine*

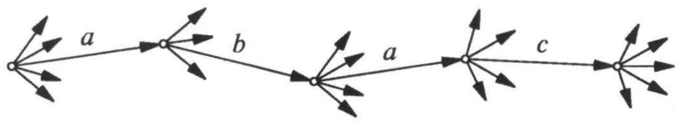
from the display, and the sets of actions representing information obtained from the lights. Such a sequence is called a *ready trace* of the process, and the subsets occurring in a ready trace are referred to as *menus*. The information about the free and blocked actions is now redundant. The set of all ready traces of a process is called its *ready trace set*, and constitutes its observable behaviour.

DEFINITIONS:

- The *ready trace relations* $\overset{\sigma}{\ast\!\!\!\rightarrow}$ for $\sigma \in (Act \cup \mathcal{P}(Act))^*$ are defined inductively by:
  1. $p \overset{\epsilon}{\ast\!\!\!\rightarrow} p$, for any process p.
  2. $p \overset{a}{\longrightarrow} q$ implies $p \overset{a}{\ast\!\!\!\rightarrow} q$.
  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 $\overset{\sigma}{\longrightarrow}$ 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 [4], PNUELI [31] and POMELLO [32] 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 $p \overset{a_1}{\longrightarrow} p_1 \overset{a_2}{\longrightarrow} \cdots \overset{a_n}{\longrightarrow} p_n$ 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 [4, 31, 32] 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 [33]. 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* $\models \subseteq \mathbb{A} \times \mathcal{L}_S$ is defined inductively by:
  1. $p \models T$ for all $p \in \mathbb{A}$.
  2. $p \models \phi \wedge \psi$ if $p \models \phi$ and $p \models \psi$.
  3. $p \models a\phi$ if for some $q \in \mathbb{A}$: $p \xrightarrow{a} q$ 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 [29]). 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 $s \subsetneq t$, if there is a simulation R with pRq.
p and q are *similar*, notation $p \leftrightarrows q$, if $p \subsetneq q$ and $q \subsetneq p$.

PROPOSITION 1.4: *Similarity is an equivalence on the domain of processes.*
PROOF: It has to be checked that $p \subsetneq p$, and $p \subsetneq q \;\&\; q \subsetneq r \;\Rightarrow\; p \subsetneq q$.
- 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 $p, q \in \mathbb{A}$ be finitely branching processes. Then $p \leftrightarrows q \;\Leftrightarrow\; S(p) = S(q)$.
PROOF: See HENNESSY & MILNER [18]. □

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). 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.

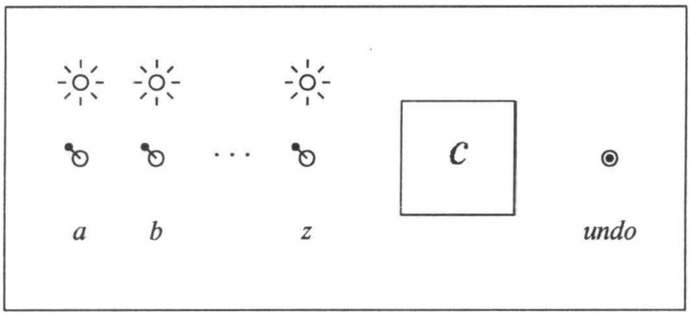


FIGURE 5. *The ready simulation machine*

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.

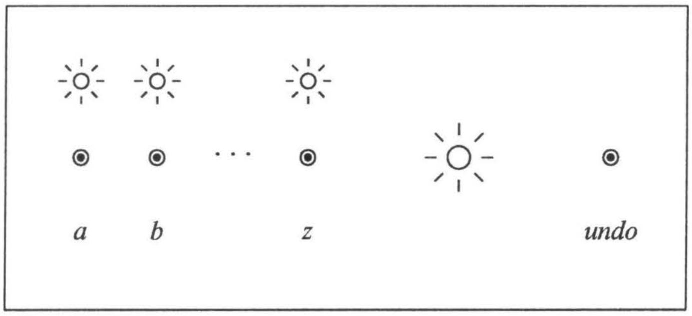


FIGURE 6. *The reactive ready simulation machine*

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 [25,26].

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* [30]. 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 $p \in \mathbb{A}$ and $\sigma \in FT(p)$. Put $Cont(\sigma)=\{a \in Act \mid \sigma a \in FT(p)\}$.
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 [8].

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 $a \notin I(p)$.

An alternative family of testing scenarios with reactive machines can be obtained by allowing the observer to try to depress more than one button at a time. In order to influence a particular choice, the observer could already start exercising pressure on buttons during the execution of the preseeding action (when no button can go down). When this preseeding action is finished, at most one of the buttons will go down. These testing scenarios are equipotent with the generative ones: putting pressure on a button is equivalent to setting the corresponding switch on 'free'.

1.12. 2-nested simulation semantics. *2-nested simulation equivalence* popped up naturally in GROOTE & VAANDRAGER [15] 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* [15]: 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 [25] 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 [25] 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 [26], 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 [23]. They assumed that every transition in a transition system has a certain 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 [8], 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* [17] (*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 [17] provided the following characterization of this equivalence.

DEFINITION: Let $p, q \in \mathbb{A}$ 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 $p, q \in \mathbb{A}$ be finitely branching processes. Then $p \sim q \Leftrightarrow HML(p) = HML(q)$.
PROOF: In HENNESSY & MILNER [18]. □

As observed by PARK [29], 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 $p \leftrightarrow q$, if there exists a bisimulation R with pRq.

The relation $\underline{\leftrightarrow}$ 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 $p \underline{\leftrightarrow} q \Leftrightarrow p \sim q$.
PROOF: "$\Rightarrow$": Straightforward with induction. "$\Leftarrow$" follows from Theorem 5.6 in MILNER [25]. □

For infinitely branching processes $\sim$ is coarser then $\underline{\leftrightarrow}$ 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 [3]. 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 $\mathcal{B}(p) = \{<a, \mathcal{B}(q)> \mid p \xrightarrow{a} q\}$ 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 [3] 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 [6], 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 [13], 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 $p \underline{\leftrightarrow} q \Leftrightarrow B(p) = B(q)$.
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. Of course the second requirement for B being a bisimulation can be proved likewise.
"$\Rightarrow$". Let B^* denote the unique solution of $\mathcal{B}^*(p) = \{<a, \mathcal{B}^*(r')> \mid \exists r: r \underline{\leftrightarrow} p \;\&\; r \xrightarrow{a} r'\}$. As for B it follows from the anti-foundation axiom that such a unique solution exists. From the symmetry and transitivity of $\underline{\leftrightarrow}$ it follows that

$$p \underline{\leftrightarrow} q \Rightarrow B^*(p) = B^*(q). \qquad (*)$$

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 $\underline{\leftrightarrow}$. For "$\subseteq$", suppose $<a, X> \in B^*(p)$. Then $\exists r: r \underline{\leftrightarrow} p,\; r \xrightarrow{a} r'$ and $X = B^*(r')$. Since $\underline{\leftrightarrow}$ is a bisimulation, $\exists p': p \xrightarrow{a} p'$ and $r' \underline{\leftrightarrow} p'$. 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 $RT(p)$.

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

The remaining statements are (also) trivial. □

Theorem 2.1 is illustrated in Figure 1. There, however, completed simulation 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 [9] (see Subsection 1.4), R^- denotes acceptance-refusal semantics [33] (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 $\mathbb{S},\mathbb{T}$ be any two of the semantics mentioned above. Then $\mathbb{S} \preccurlyeq \mathbb{T}$ 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 $\mathbb{S} \preccurlyeq \mathbb{T}$ 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.

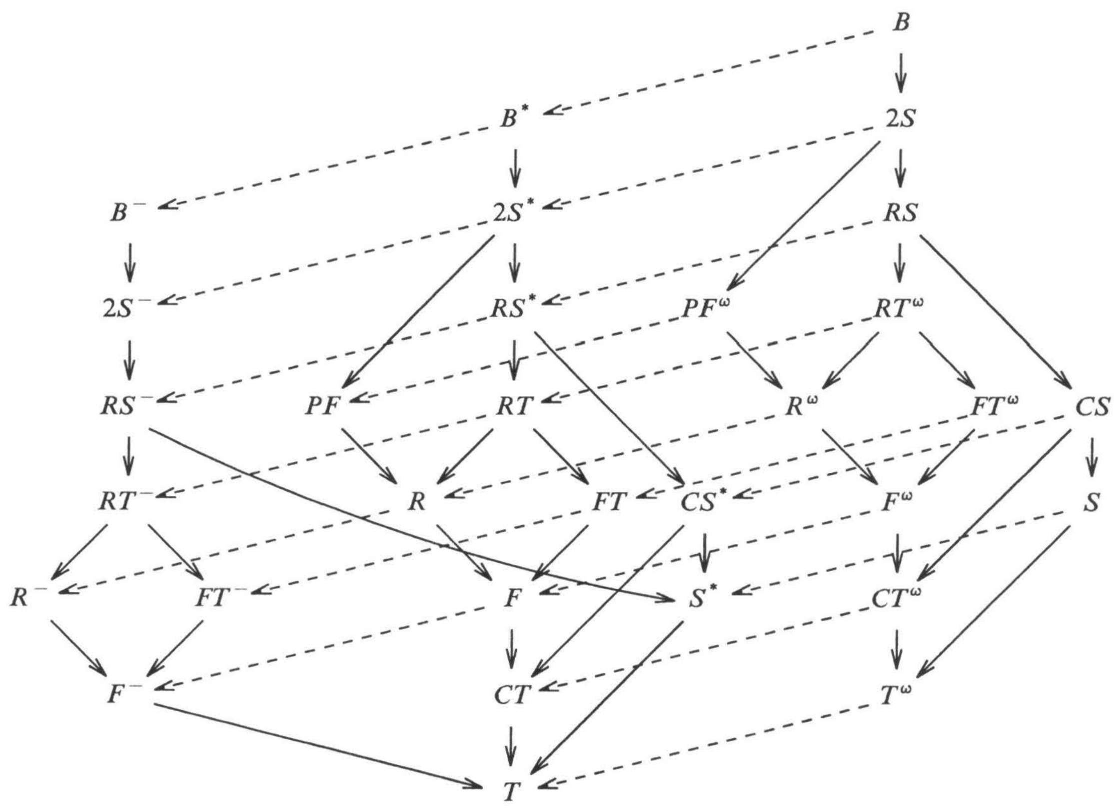


FIGURE 7. *The infinitary linear time - branching time spectrum*

2.3. Image finite processes.

DEFINITION: A process $p \in \mathbb{A}$ is *image finite* if for each $\sigma \in Act^*$ the set $\{q \in \mathbb{A} \mid p \xrightarrow{\sigma} q\}$ 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 [18]. 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 König's 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 $q \in \mathbb{A}$, $F^-(p)=F^-(q) \Rightarrow F(p)=F(q)$, 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 $q \in \mathbb{A}$ 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 [29]): *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 (NODES(g), EDGES(g), ROOT(g)), where

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

If $e = (s,a,t) \in$ 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 π: $s_0 \xrightarrow{a_1} s_1 \xrightarrow{a_2} \cdots \xrightarrow{a_n} s_n$, one says that π *goes from* s_0 *to* s_n; it *starts* in s_0 and *ends* in $end(\pi)=s_n$. Let 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 $s \in$ NODES(g), let g_s be the process graph defined by
- NODES$(g_s)=\{t \in$ NODES$(g) \mid$ *there is a path going from s to t*$\}$,
- ROOT$(g_s)=s \in$ NODES(g_s),
- and $(t,a,u) \in$ EDGES(g_s) iff $t,u \in$ NODES(g_s) and $(t,a,u) \in$ EDGES(g).

Of course $g_s \in \mathbb{G}$. Remark that $g_{\mathrm{ROOT}(g)}=g$. Now on $\mathbb{G}$ action relations $\xrightarrow{a}$ for $a \in Act$ are defined by $g \xrightarrow{a} h$ iff (ROOT$(g),a,s) \in$ 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 $O \in \{T, CT, R, F, RT, FT\}$ and $g \in \mathbb{G}$, are characterized in terms of the paths of g, rather than the generalized action relations between graphs.

DEFINITION: Let $g \in \mathbb{G}$ and let π: $s_0 \xrightarrow{a_1} s_1 \xrightarrow{a_2} \cdots \xrightarrow{a_n} s_n \in$ PATHS(g). Consider the following notions:
- the *trace* associated to π: $T(\pi)=a_1a_2 \cdots a_n \in Act^*$;
- the *menu* of a node $s \in$ NODES(g): $I(s)=\{a \in Act \mid \exists t\colon (s,a,t) \in$ 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
  - $I(s_0)a_1I(s_1)a_2 \cdots a_nI(s_n) \in RT(\pi)$,
  - $\sigma X\rho \in RT(\pi) \Rightarrow \sigma\rho \in RT(\pi)$,
  - $\sigma X\rho \in RT(\pi) \Rightarrow \sigma XX\rho \in RT(\pi)$;
- and the *failure trace set* of π: $FT(\pi)$ is the smallest subset of $(Act \cup \mathcal{P}(Act))^*$ satisfying
  - $(A-I(s_0))a_1(A-I(s_1))a_2 \cdots a_n(A-I(s_n)) \in FT(\pi)$,
  - $\sigma X\rho \in FT(\pi) \Rightarrow \sigma\rho \in FT(\pi)$,
  - $\sigma X\rho \in FT(\pi) \Rightarrow \sigma XX\rho \in FT(\pi)$,
  - $\sigma X\rho \in FT(\pi) \wedge Y \subseteq X \Rightarrow \sigma Y\rho \in FT(\pi)$.

PROPOSITION 2.5:

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

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

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

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

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

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

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 $g,h \in \mathbb{G}$. 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.

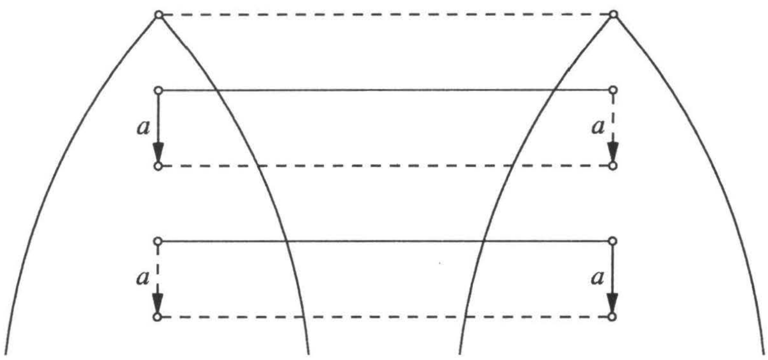


FIGURE 8

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 $g,h \in \mathbb{G}$, π: $s_0 \xrightarrow{a_1} s_1 \xrightarrow{a_2} \cdots \xrightarrow{a_n} s_n \in \text{PATHS}(g)$ and π': $t_0 \xrightarrow{a_1} t_1 \xrightarrow{a_2} \cdots \xrightarrow{a_n} t_m \in \text{PATHS}(h)$. 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 $g,h \in \mathbb{G}$ are ready trace equivalent iff*
- *for any path $\pi \in \text{PATHS}(g)$ in g there is a $\pi' \in \text{PATHS}(h)$ such that $\pi =_{FT} \pi'$*
- *and for any path $\pi \in \text{PATHS}(g)$ in h there is a $\pi' \in \text{PATHS}(g)$ such that $\pi =_{FT} \pi'$.*

They are failure trace equivalent iff
- *for any path $\pi \in \text{PATHS}(g)$ in g there is a $\pi' \in \text{PATHS}(h)$ such that $\pi \leqslant_{FT} \pi'$*
- *and for any path $\pi \in \text{PATHS}(g)$ in h there is a $\pi' \in \text{PATHS}(g)$ 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 $g,h \in \mathbb{G}$. A *graph isomorphism* between g and h is a bijective function $f: \text{NODES}(g) \rightarrow \text{NODES}(h)$ satisfying
- $f(\text{ROOT}(g)) = \text{ROOT}(g)$ and
- $(s,a,t) \in \text{EDGES}(g) \Leftrightarrow (f(s),a,f(t)) \in \text{EDGES}(h)$.

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 $f:\text{NODES}(g) \rightarrow \text{NODES}(h)$ be a graph isomorphism. Define $R \subseteq \text{NODES}(g) \times \text{NODES}(h)$ 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 $f:\text{NODES}(g) \rightarrow \text{NODES}(h)$ 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 $\mathbb{A}$ be an arbitrary labelled transition system and let $p \in \mathbb{A}$. The *canonical graph* $G(p)$ of p is defined as follows:

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

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

PROPOSITION 2.7: *$G:\mathbb{A} \rightarrow \mathbb{G}$ is an injective function, satisfying, for $a \in Act$: $G(p) \xrightarrow{a} G(q) \Leftrightarrow p \xrightarrow{a} q$.*
PROOF: Trivial. □
COROLLARY: *For $p \in \mathbb{A}$ 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 $\mathfrak{S}$ and $\mathfrak{T}$ be semantics on $\mathbb{H}$ from the series T, CT, F, R, FT, RT, S, CS, RS, PF, 2S, B. Then $\mathfrak{S} \preccurlyeq \mathfrak{T}$ 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 $\mathfrak{S} \preccurlyeq \mathfrak{T}$, 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 $\mathfrak{S}$.

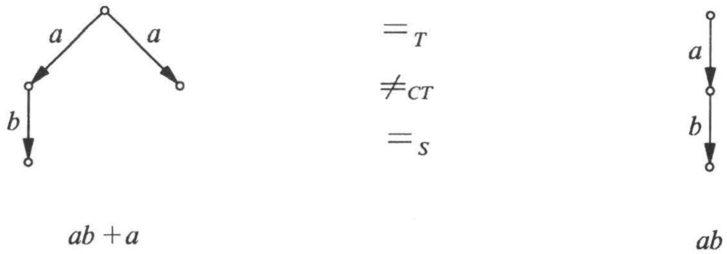


FIGURE 9

1. $T \not\succcurlyeq CT$. For the graphs of Figure 9, $T(left)=T(right)=\{\epsilon, a, ab\}$, whereas $CT(left) \neq CT(right)$ (since $a \in CT(left)-CT(right)$). 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 $\mathfrak{S} \not\preccurlyeq \mathfrak{T}$ for $\mathfrak{S} \in \{CT, CS, F, R, FT, RT, RS, PF, 2S, B\}$ and $\mathfrak{T} \in \{T, S\}$.

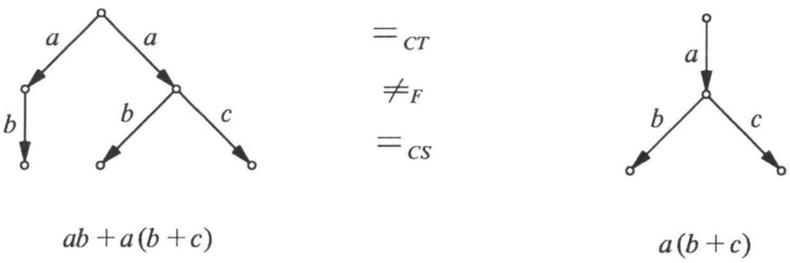


FIGURE 10

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 $\mathfrak{S} \not\preccurlyeq \mathfrak{T}$ for $\mathfrak{S} \in \{F, R, FT, RT, RS, PF, 2S, B\}$ and $\mathfrak{T} \in \{CT, CS\}$.

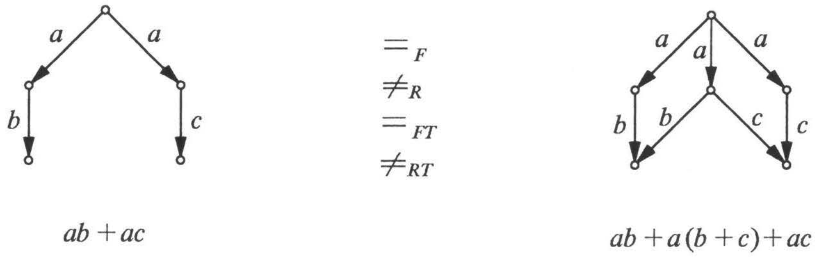


FIGURE 11

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 $\mathbb{S} \not\succcurlyeq \mathbb{T}$ for any $\mathbb{S} \preccurlyeq FT$ and $\mathbb{T} \succcurlyeq R$, so in particular $F \not\succcurlyeq R$ and $FT \not\succcurlyeq RT$.

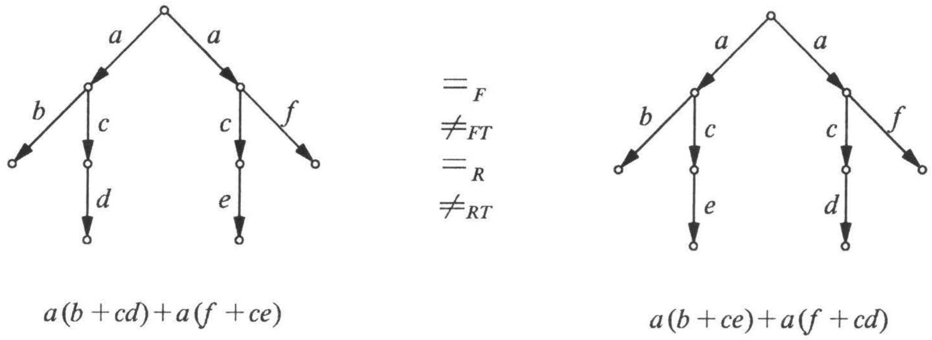
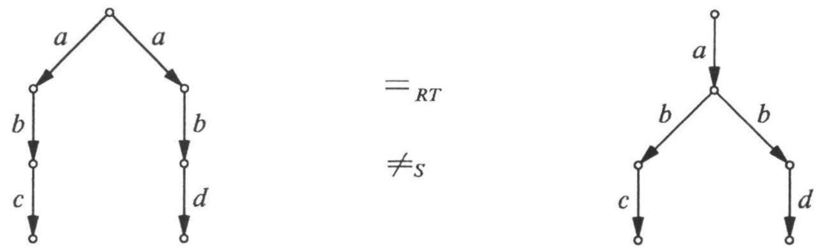


FIGURE 12

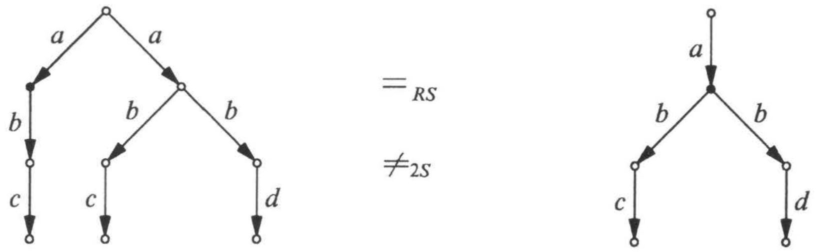
4. $R \not\succcurlyeq FT$. For the graphs of Figure 12, $R(left) = R(right)$, whereas $FT(left) \neq FT(right)$. The first statement follows since in the second graph only 4 ready pairs swopped places. The second one follows since $a\{b\}ce \in FT(left) - FT(right)$. Hence these processes are identified in readiness semantics but distinguished in failure trace semantics. Since $\preccurlyeq$ is a partial order, the same example shows that $\mathbb{S} \not\succcurlyeq \mathbb{T}$ for any $\mathbb{S} \preccurlyeq R$ and $\mathbb{T} \succcurlyeq FT$, 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.



$abc + abd$ $\qquad$ $a(bc + bd)$

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 $\mathcal{S} \not\succcurlyeq \mathcal{T}$ for any $\mathcal{S} \preccurlyeq RT$ and $\mathcal{T} \succcurlyeq S$, so in particular $T \not\succcurlyeq S$, $CT \not\succcurlyeq CS$ and $RT \not\succcurlyeq RS$.



$abc + a(bc + bd)$ $\qquad$ $a(bc + bd)$

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 $\mathcal{S} \not\succcurlyeq PF$ for any $\mathcal{S} \preccurlyeq RS$.

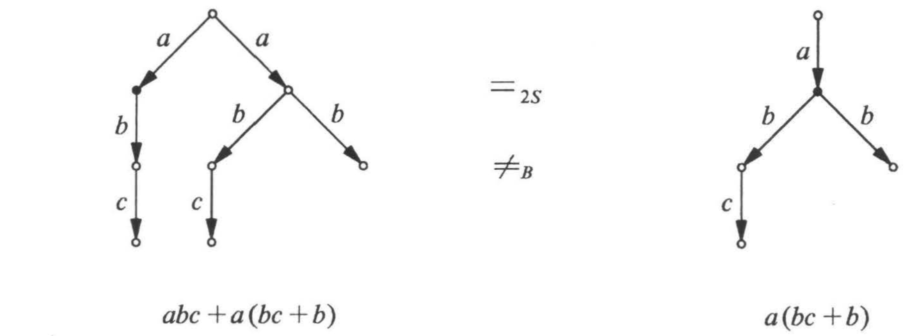


FIGURE 15

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, $a\neg b\neg cT \in HML(left) - HML(right)$. □

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

PROOF: The following counterexamples, together with the ones used in the previous proof, provide for any statement $\mathbb{S} \preccurlyeq \mathcal{T}$, 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 $\mathbb{S}$.

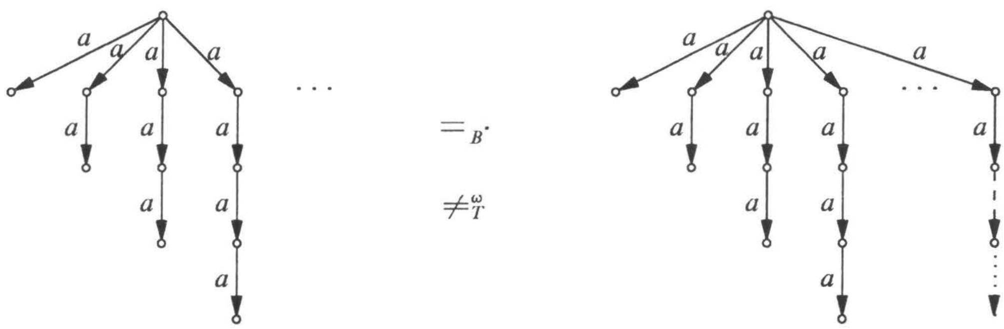


FIGURE 16

8. $B^* \not\succcurlyeq T^\omega$. The graphs of Figure 16 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 $\mathbb{S} \not\succcurlyeq \mathcal{T}$ for $\mathbb{S} \preccurlyeq B^*$ and $\mathcal{T} \succcurlyeq T^\omega$.

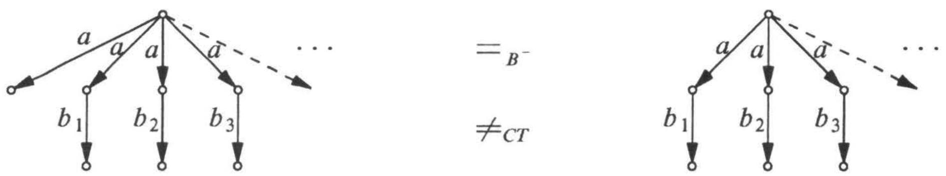


FIGURE 17

9. $B^- \not\succcurlyeq CT$. For the graphs of Figure 17, $HML(left) = HML(right)$, whereas $CT(left) \neq CT(right)$. 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 $a \in CT(left) - CT(right)$. Since $\preccurlyeq$ is a partial order it follows that $\mathbb{S} \not\succcurlyeq \mathbb{T}$ for $\mathbb{S} \preccurlyeq B^-$ and $\mathbb{T} \succcurlyeq CT$. □

One could say that a semantics $\mathbb{S}$ *respects deadlock behaviour* iff $\mathbb{S} \succcurlyeq CT$. 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 17 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 $\mathbb{P}$.

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

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 for ten from the twelve semantics of this paper that differ on BSSCP can be found. 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 [7] by means of *graph transformations*. A similar proof for RT can be found in BAETEN, BERGSTRA & KLOP [4]. For the remaining semantics a proof can be given along the same lines. □

CONCLUDING REMARKS

In this paper 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 paper 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 [8] and [15].

Also the generalization of this work to a setting with silent moves and/or with parallelism is left for the future. In this case the number of equivalences that are worth classifying is much larger. However, in many papers parts of a classification can be found already (see for instance [32]).

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 [9] and for readiness semantics in [28].

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*, unpublished manuscript.

[3] P. ACZEL (1988): *Non-well-founded sets*, CSLI Lecture Notes No.14, Stanford University.

[4] 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.

[5] 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.

[6] J.W. DE BAKKER & J.I. ZUCKER (1982): *Processes and the denotational semantics of concurrency.* I&C 54(1/2), pp. 70-120.

[7] 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.

[8] 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.

[9] S.D. BROOKES, C.A.R. HOARE & A.W. ROSCOE (1984): *A theory of communicating sequential processes.* JACM 31(3), pp. 560-599.

[10] 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.

[11] R. DE NICOLA (1987): *Extensional equivalences for transition systems.* Acta Informatica 24, pp. 211-237.

[12] R. DE NICOLA & M. HENNESSY (1984): *Testing equivalences for processes.* Theoretical Computer Science 34, pp. 83-133.

[13] 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.

[14] 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 5^{th} Annual Symposium on Logic in Computer Science (LICS 90), Philadelphia, USA, IEEE Computer Society Press, Washington.

[15] 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.

[16] M. HENNESSY (1985): *Acceptance trees*. JACM 32(4), pp. 896-928.

[17] 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:.

[18] M. HENNESSY & R. MILNER (1985): *Algebraic laws for nondeterminism and concurrency*. JACM 32(1), pp. 137-161.

[19] C.A.R. HOARE (1978): *Communicating sequential processes*. Communications of the ACM 21(8), pp. 666-677.

[20] 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.

[21] C.A.R. HOARE (1985): *Communicating sequential processes*, Prentice-Hall International.

[22] J.K. KENNAWAY (1981): *Formal semantics of nondetermism and parallelism*. Ph.D. Thesis, University of Oxford.

[23] 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 16^{th} Annual ACM Symposium on Principles of Programming Languages (POPL), Austin, Texas, ACM Press, New York 1989.

[24] A.R. MEYER (1985): *Report on the 5^{th} international workshop on the semantics of programming languages in Bad Honnef*. Bulletin of the EATCS 27, pp. 83-84.

[25] R. MILNER (1980): *A calculus of communicating systems*, LNCS 92, Springer-Verlag.

[26] 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.

[27] R. MILNER (1983): *Calculi for synchrony and asynchrony*. Theoretical Computer Science 25, pp. 267-310.

[28] E.-R. OLDEROG & C.A.R. HOARE (1986): *Specification-oriented semantics for communicating processes*. Acta Informatica 23, pp. 9-66.

[29] D.M.R. PARK (1981): *Concurrency and automata on infinite sequences*. In: Proceedings 5^{th} GI Conference (P. Deussen, ed.), LNCS 104, Springer-Verlag, pp. 167-183.

[30] I.C.C. PHILLIPS (1987): *Refusal testing*. Theoretical Computer Science 50, pp. 241-284.

[31] 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.

[32] 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.

[33] W.C. ROUNDS & S.D. BROOKES (1981): *Possible futures, acceptances, refusals and communicating processes*. In: Proceedings 22^{nd} Annual Symposium on Foundations of Computer Science, Nashville, USA 1981, IEEE, New York, pp. 140-149.