The Linear Time - Branching Time Spectrum I*

R.J. van Glabbeek
Computer Science Department, Stanford University
Stanford, CA 94305-9045, USA
rvg@cs.stanford.edu

In this paper various semantics in the linear time - branching time spectrum are presented in a uniform, model-independent way. Restricted to the class 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.

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 by mathematical structures or by expressions in a system description language. Verification is the activity of proving statements about processes, for instance that the actual behaviour of a system is equal to its intended behaviour. Of course, this is only possible if a criterion has been defined, determining whether or not two processes are equal, i.e. two systems behave similarly. Such a criterion constitutes the *semantics* of a process theory. (To be precise, it constitutes the semantics of the equality concept employed in a process theory.) Which aspects of the behaviour of a system are of importance to a certain user depends on the environment in which the system will be running, and on the interests of the particular user. Therefore it is not a task of process theory to find the 'true' semantics of processes, but rather to determine which process semantics is suitable for which applications.

Comparative concurrency semantics This paper aims at the classification of process semantics.[1] The set of possible process semantics can be partially ordered by the relation 'makes

*This is an extension of [15]. The research reported in this paper has been initiated at CWI in Amsterdam, continued at the Technical University of Munich, and finalized at Stanford University. It has been supported by Sonderforschungsbereich 342 of the TU München and by ONR under grant number N00014-92-J-1974. Part of it was carried out in the preparation of a course Comparative Concurrency Semantics, given at the University of Amsterdam, spring 1988.

[1] This field of research is called *comparative concurrency semantics*, a terminology first used by Meyer in [30].

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 unravelled 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[5]. 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*,

[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.

[5]I use the word *domain* in the sence of *universal algebra*; it can be any class of mathematical objects—typically the first component of an *algebra*; the other component being a collection of operators defined on this domain. Without further adjectives I do not refer to the more restrictive domains employed in *domain theory*.

- *explicit domains*, in which a process is represented as a mathematically coded set of its properties,
- *projective limit domains*, which are obtained as projective limits of series of finite term domains,
- and *term domains*, in which a process is represented as a term in a system description language.

Action relations Write $p \xrightarrow{a} q$ if the process p can evolve into the process q, while performing the action a. The binary predicates $\xrightarrow{a}$ are called *action relations*. The semantic equivalences which are treated in this paper will be defined entirely in terms of action relations. Hence these definitions apply to any process domain on which action relations are defined. Such a domain is called a *labelled transition system*. 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. The material pertaining to infinitely branching processes has been coloured brown, and can easily be omitted in first reading. 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 [27]). 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 [26]. 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 [33]. Bisimulation semantics is the standard semantics for the system description language CCS (Milner [31]). The notion of bisimulation was introduced in Park [35]. Bisimulation equivalence is a refinement of *observational equivalence*, as introduced by Hennessy & Milner in [23]. 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 ten 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*. *Failures semantics* is introduced in Brookes, Hoare & Roscoe [10], and used in the construction of a model for the system description language CSP (Hoare [25, 27]). It is finer than completed trace semantics. The semantics based on *testing equivalences*, as developed in De Nicola & Hennessy [14], coincides with failures semantics on the domain of finitely branching, concrete, sequential processes, as do the semantics of Kennaway [28] and Darondeau [12]. This has been established in De Nicola [13]. In Olderog & Hoare [34] *readiness semantics* is presented, which is slightly finer than failures

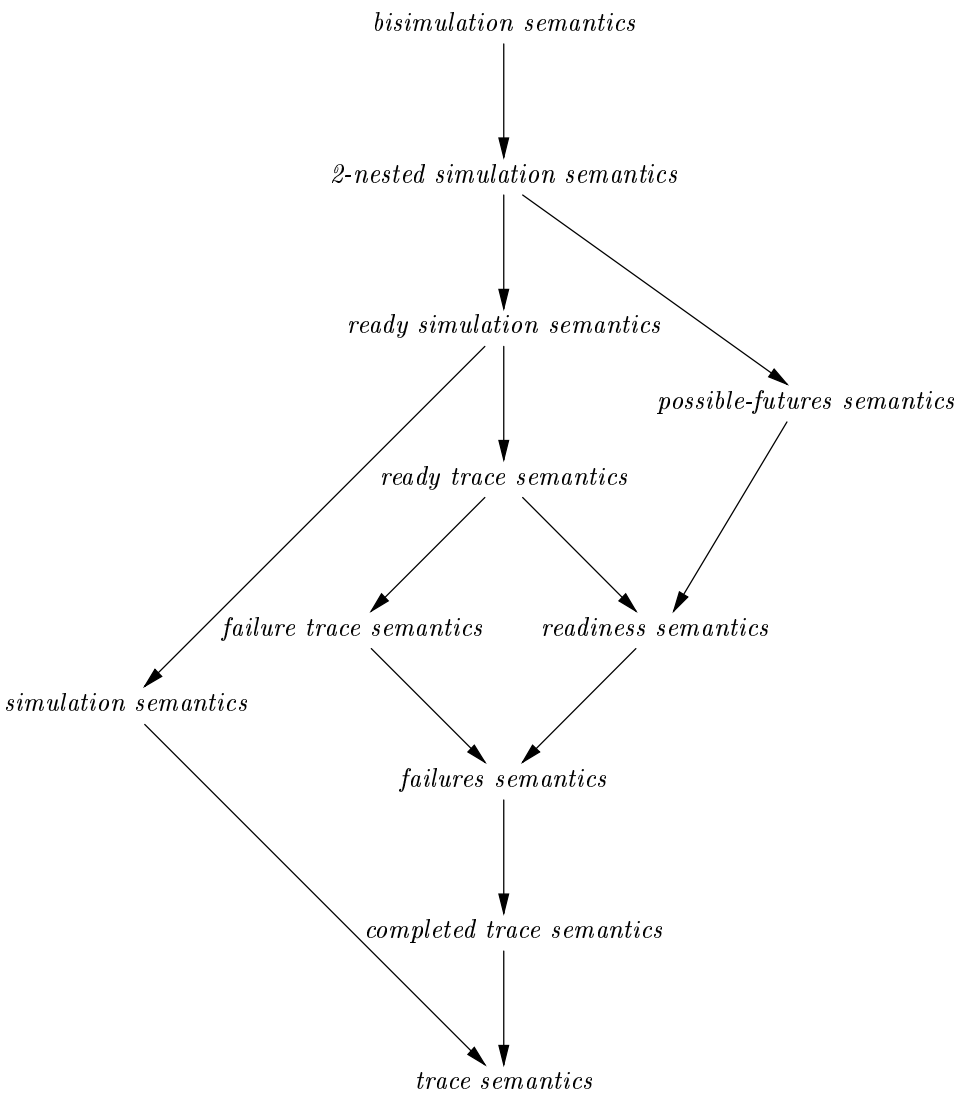


Figure 1: The linear time - branching time spectrum

semantics. Between readiness and bisimulation semantics one finds *ready trace semantics*, as introduced independently in Pnueli [37] (there called *barbed semantics*), Baeten, Bergstra & Klop [4] and Pomello [38] (under the name *exhibited behaviour semantics*). The natural completion of the square, suggested by failures, readiness and ready trace semantics yields *failure trace semantics*. For finitely branching processes this is the same as *refusal semantics*, introduced in Phillips [36]. *Simulation semantics*, based on the classical notion of *simulation* (see e.g. Park [35]), is independent of the last five semantics. *Ready simulation semantics* was introduced in Bloom, Istrail & Meyer [9] under the name *GSOS trace congruence*. It is finer than ready trace as well as simulation semantics. In Larsen & Skou [29] a more operational characterization of this equivalence was given under the name $\frac{2}{3}$*-bisimulation equivalence*. This characterization resembles the one used in this paper. Finally *2-nested simulation semantics*, introduced in Groote & Vaandrager [21], is located between ready simulation and bisimulation semantics, and *possible-futures semantics*, as

proposed in ROUNDS & BROOKES [40], can be positioned between 2-nested simulation and readiness semantics. Among the semantics which are not definable in terms of action relations and thus fall outside the scope of this paper, one finds semantics that take stochastic properties of processes into account, as in VAN GLABBEEK, SMOLKA, STEFFEN & TOFTS [19] and semantics that make almost no identifications and are hardly used for system verification.

About the contents The first section of this paper introduces labelled transition systems and process graphs. A labelled transition system is any process domain that is equipped with action relations. The domain of *process graphs* or *state transition diagrams* is one of the most popular labelled transition systems. In Sections 2–12 all semantic equivalences mentioned above are defined on arbitrary labelled transition systems. In particular these definitions apply to the domain of process graphs. Most of the 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 to certain experiments.) I will try to capture these motivations in terms of *button pushing experiments* (cf. MILNER [31], pp. 10-12). Furthermore the semantics will be partially ordered by the relation 'makes at least as many identifications as'. This yields the linear time – branching time spectrum. Counterexamples are provided, showing that on the graph domain this ordering cannot be further expanded. However, for deterministic processes the spectrum collapses, as was first observed by PARK [35]. Finally, in Section 15, ten of the semantics are applied to a simple language for finite, concrete, sequential, nondeterministic processes, and for each of them a complete axiomatization is provided.

1 Labelled transition systems and process graphs

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 $\mathbb{P}$ on which infix written binary predicates $\xrightarrow{a}$ are defined for each action $a \in Act$. The elements of $\mathbb{P}$ 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 q and r. 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 finite sequences and Σ^∞ the set of infinite sequences over Σ. $\Sigma^\omega := \Sigma^* \cup \Sigma^\infty$. Write ϵ for the empty sequence, $\sigma\rho$ for the concatenation of $\sigma \in \Sigma^*$ and $\rho \in \Sigma^\omega$, and a for the sequence consisting of the single symbol $a \in \Sigma$.

Definition 1.1 A *labelled transition system* is a pair $(\mathbb{P}, \rightarrow)$ with $\mathbb{P}$ a class and $\rightarrow \subseteq \mathbb{P} \times Act \times \mathbb{P}$, such that for $p \in \mathbb{P}$ and $a \in Act$ the class $\{q \in \mathbb{P} \mid p \xrightarrow{a} q\}$ is a set.

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

Definition 1.2 (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 recursively 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 $p \in \mathbb{P}$ is *finite* if the set $\{(\sigma, q) \in (Act^* \times \mathbb{P}) \mid p \xrightarrow{\sigma} q\}$ is finite.
- p is *image finite* if for each $\sigma \in Act^*$ the set $\{q \in \mathbb{P} \mid p \xrightarrow{\sigma} q\}$ is finite.
- p is *well-founded* if there is no infinite sequence $p \xrightarrow{a_1} p_1 \xrightarrow{a_2} p_2 \xrightarrow{a_3} \cdots$.
- p is *finitely branching* if for each $q \in \mathbb{P}$ with $p \xrightarrow{\sigma} q$ for some $\sigma \in Act^*$, the set $\{(a, r) \in Act \times \mathbb{P} \mid q \xrightarrow{a} r\}$ is finite.

Note that a process $p \in \mathbb{P}$ is image finite iff for each $q \in \mathbb{P}$ with $p \xrightarrow{\sigma} q$ for some $\sigma \in Act^*$ and each $a \in Act$, the set $\{r \in \mathbb{P} \mid q \xrightarrow{a} r\}$ is finite. Hence finitely branching processes are image finite. Moreover, by König's lemma a process is finite iff it is well-founded and finitely branching.

1.2 Process graphs

Definition 1.3 A *process graph* over an alphabet Act is a rooted, directed graph whose edges are labelled by elements of Act. Formally, a process graph g is a triple $(\text{NODES}(g), \text{EDGES}(g), \text{ROOT}(g))$, where

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

If $e = (s, a, t) \in \text{EDGES}(g)$, one says that e *goes from* s *to* t. A (finite) *path* π in a process graph is an alternating sequence of nodes and edges, starting and ending with a node, such that each edge goes from the node before it to the node after it. If $\pi = s_0(s_0, a_1, s_1)s_1(s_1, a_2, s_2) \cdots (s_{n-1}, a_n, s_n)s_n$, also denoted as $\pi : s_0 \xrightarrow{a_1} s_1 \xrightarrow{a_2} \cdots \xrightarrow{a_n} s_n$, one says that π *goes from* s_0 *to* s_n; it *starts* in s_0 and *ends* in $end(\pi) = s_n$. Let $\text{PATHS}(g)$ be the set of paths in g starting from the root. If s and t

are nodes in a process graph then *t can be reached from s* if there is a path going from s to t. A process graph is said to be *connected* if all its nodes can be reached from the root; it is a *tree* if each node can be reached from the root by exactly one path. Let $\mathbb{G}$ be the domain of connected process graphs over a given alphabet *Act*.

Definition 1.4 Let $g, h \in \mathbb{G}$. A *graph isomorphism* between g and h is a bijective function $f : \text{NODES}(g) \to \text{NODES}(h)$ satisfying

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

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 relation on $\mathbb{G}$.

Connected process graphs can be pictured by using open dots ($\circ$) to denote nodes, and labelled arrows to denote edges, as can be seen further on. 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.

Definition 1.5 For $g \in \mathbb{G}$ and $s \in \text{NODES}(g)$, let g_s be the process graph defined by

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

Of course $g_s \in \mathbb{G}$. Note that $g_{\text{ROOT}(g)} = g$. Now on $\mathbb{G}$ action relations $\xrightarrow{a}$ for $a \in Act$ are defined by $g \xrightarrow{a} h$ iff $(\text{ROOT}(g), a, s) \in \text{EDGES}(g)$ and $h = g_s$. This makes $\mathbb{G}$ into a labelled transition system.

1.3 Embedding labelled transition systems in $\mathbb{G}$

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

- $\text{NODES}(G(p)) = \{q \in \mathbb{P} \mid \exists \sigma \in A^* : \ p \xrightarrow{\sigma} q\}$,
- $\text{ROOT}(G(p)) = p \in \text{NODES}(G(p))$,
- 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 $\mathbb{P}$ to $\mathbb{G}$.

Proposition 1.1 $G : \mathbb{P} \to \mathbb{G}$ is a injective and satisfies, for $a \in Act$: $G(p) \xrightarrow{a} G(q) \Leftrightarrow p \xrightarrow{a} q$.
Proof: Trivial. □

Proposition 1.1 says that G is an *embedding* of $\mathbb{P}$ in $\mathbb{G}$. It implies that any labelled transition system over *Act* can be represented as a subclass $G(\mathbb{P}) = \{G(p) \in \mathbb{G} \mid p \in \mathbb{P}\}$ 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 $G : \mathbb{G} \to \mathbb{G}$ leaves its arguments intact up to graph isomorphism.

Proposition 1.2 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 $f : \text{NODES}(G(g)) \to \text{NODES}(g)$ defined by $f(g_s) = s$ is a graph isomorphism. □

1.4 Equivalences relations on labelled transition systems

This paper studies semantics $\mathcal{S}$ on labelled transition systems. As the definitions of these semantics are given entirely in terms of action relations, they apply to any labelled transition system $\mathbb{P}$. A semantics $\mathcal{S}$ on $\mathbb{P}$ is given by an equivalence relation $=_{\mathcal{S}}$ on $\mathbb{P}$. Write $\mathcal{S} \preceq_{\mathbb{P}} \mathcal{T}$ if semantics $\mathcal{S}$ makes at least as much identifications as semantics $\mathcal{T}$. This is the case if the equivalence corresponding with $\mathcal{S}$ is equal to or coarser than the one corresponding with $\mathcal{T}$, i.e. if $p =_{\mathcal{T}} q \Rightarrow p =_{\mathcal{S}} q$ for all $p, q \in \mathbb{P}$. Let $\preceq$ abbreviate $\preceq_{\mathbb{G}}$. The following is then immediate by Proposition 1.1.

Corollary 1.1 $\mathcal{S} \preceq \mathcal{T}$ iff $\mathcal{S} \preceq_{\mathbb{P}} \mathcal{T}$ for *each* labelled transition relation $\mathbb{P}$.
On the other hand, $\mathcal{S} \not\preceq \mathcal{T}$ iff $\mathcal{S} \not\preceq_{\mathbb{P}} \mathcal{T}$ for *some* labelled transition relation $\mathbb{P}$.

2 Trace semantics

Definition 2 $\sigma \in Act^*$ 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*, notation $p =_T q$, if $T(p) = T(q)$. In *trace semantics* (T) two processes are identified iff they are trace equivalent.

Testing scenario 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.

Modal Characterization

Definition 2.1 The set $\mathcal{L}_T$ of *trace formulas* over Act is defined recursively by:

- $\top \in \mathcal{L}_T$.
- If $\varphi \in \mathcal{L}_T$ and $a \in Act$ then $a\varphi \in \mathcal{L}_T$.

The *satisfaction relation* $\models \subseteq \mathbb{P} \times \mathcal{L}_T$ is defined recursively by:

- $p \models \top$ for all $p \in \mathbb{P}$.
- $p \models a\varphi$ if for some $q \in \mathbb{P}$: $p \xrightarrow{a} q$ and $q \models \varphi$.

Note that a trace formula satisfied by a process p represents nothing more or less than a trace of p. Hence one has

Proposition 2.1 $p =_T q \;\Leftrightarrow\; \forall \varphi \in \mathcal{L}_T (p \models \varphi \Leftrightarrow q \models \varphi)$. □

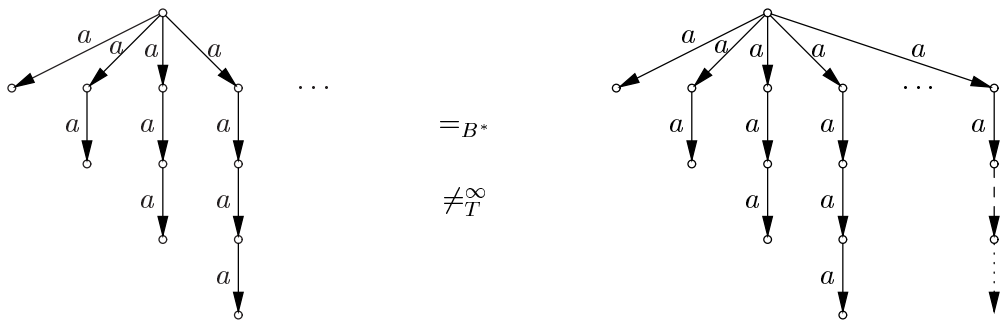
Process Graph Characterization Let $g \in \mathbb{G}$ and $\pi : s_0 \xrightarrow{a_1} s_1 \xrightarrow{a_2} \cdots \xrightarrow{a_n} s_n \in \text{PATHS}(g)$. Then $T(\pi) := a_1 a_2 \cdots a_n \in Act^*$ is the *trace* of π. As $\mathbb{G}$ is a labelled transition system, $T(g)$ is defined above. Alternatively, it could be defined as the set of traces of paths of g. It is easy to see that these definitions are equivalent:

Proposition 2.2 $T(g) = \{T(\pi) \mid \pi \in \text{PATHS}(g)\}$. □

Infinite processes For infinite processes one distinguishes two variants of trace semantics: *(finitary) trace semantics* as defined above, and *infinitary trace semantics* (T^∞), obtained by taking infinite runs into account.

Definition 2.2 $a_1a_2\cdots \in Act^\infty$ is an *infinite trace* of a process $p \in \mathbb{P}$, if there are processes $p_1, p_2, ...$ such that $p \xrightarrow{a_1} p_1 \xrightarrow{a_2} \cdots$. Let $T^\infty(p)$ denote the set of infinite traces of p. Two processes p and q are *infinitary trace equivalent*, notation $p =_T^\infty q$, if $T(p) = T(q)$ and $T^\infty(p) = T^\infty(q)$.

Clearly $p =_T^\infty q \Rightarrow p =_T q$. That on $\mathbb{G}$ the reverse does not hold follows from Counterexample 1:



Counterexample 1: Finitary equivalent but not infinitary equivalent

one has $T(left) = T(right) = \{a^n \mid n \in \mathbb{N}\}$, but $T^\infty(left) \neq T^\infty(right)$, as only the graph at the right has an infinite trace.

However, with König's lemma one easily proves that for image finite processes finitary and infinitary trace equivalence coincide:

Proposition 2.3 Let p and q be image finite processes with $p =_T q$. Then $p =_T^\infty q$.
Proof: It is sufficient to show that $T^\infty(p)$ can be expressed in terms of $T(p)$ for any image finite process p. In fact, $T^\infty(p)$ consists of all those infinite traces for which all finite prefixes are in $T(p)$. One direction of this statement is trivial: if $\sigma \in T^\infty(p)$, all finite prefixes of σ must be in $T(p)$. For the other direction suppose that, for $i \in \mathbb{N}$, $a_i \in Act$ and $a_1a_2\cdots a_i \in T(p)$. With induction on $i \in \mathbb{N}$ one can show that there exists processes p_i such that $i = 0$ and $p_0 = p$, or $p_{i-1} \xrightarrow{a_i} p_i$, and for every $j \geq i$ one has $a_{i+1}a_{i+2}\cdots a_j \in T(p_i)$. The existence of these p_i's immediately entails that $a_1a_2a_3\cdots \in T^\infty(p)$. The base case ($i = 0$) is trivial. Suppose the claim holds for certain i. For every $j \geq i+1$ there must be a process q with $p_i \xrightarrow{a_{i+1}} q$ and $a_{i+2}a_{i+3}\cdots a_j \in T(q)$. As there are only finitely many processes q with $p_i \xrightarrow{a_{i+1}} q$, there must be one choice of q for which $a_{i+2}a_{i+3}\cdots a_j \in T(q)$ for infinitely many values of j. Take this q to be p_{i+1}. As $T(p_{i+1})$ is prefix-closed, one has $a_{i+2}a_{i+3}\cdots a_j \in T(p_{i+1})$ for *all* $j \geq i+1$. □

3 Completed trace semantics

Definition 3 $\sigma \in Act^*$ is a *complete trace* of a process p, if there is a process q, such that $p \xrightarrow{\sigma} q$ and $I(q) = \emptyset$. Let $CT(p)$ denote the set of complete traces of p. Two processes p and q are *completed trace equivalent*, notation $p =_{CT} q$, if $T(p) = T(q)$ and $CT(p) = CT(q)$. In *completed trace semantics* (CT) two processes are identified iff they are completed trace equivalent.

Testing scenario Completed trace semantics can be explained with the following (rather trivial) *completed trace machine*. The process is modelled as a black box that contains as its interface to

Figure 2: The completed trace machine

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 $(\mathbb{P}, \rightarrow)$. 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. Hence the observer records either a sequence of actions performed in succession—a trace of the process—or such a sequence followed by deadlock—a completed trace. 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.

Modal Characterization

Definition 3.1 The set $\mathcal{L}_{CT}$ of *completed trace formulas* over Act is defined recursively by:

- $\top \in \mathcal{L}_{CT}$.
- $0 \in \mathcal{L}_{CT}$.
- If $\varphi \in \mathcal{L}_{CT}$ and $a \in Act$ then $a\varphi \in \mathcal{L}_{CT}$.

The *satisfaction relation* $\models \subseteq \mathbb{P} \times \mathcal{L}_{CT}$ is defined recursively by:

- $p \models \top$ for all $p \in \mathbb{P}$.
- $p \models 0$ if $I(p) = \emptyset$.
- $p \models a\varphi$ if for some $q \in \mathbb{P}$: $p \stackrel{a}{\longrightarrow} q$ and $q \models \varphi$.

Note that a completed trace formula satisfied by a process p represents either a trace (if it has the form $a_1 a_2 \cdots a_n \top$) or a completed trace (if it has the form $a_1 a_2 \cdots a_n 0$). Hence one has

Proposition 3.1 $p =_{CT} q \;\Leftrightarrow\; \forall \varphi \in \mathcal{L}_{CT}(p \models \varphi \Leftrightarrow q \models \varphi)$. □

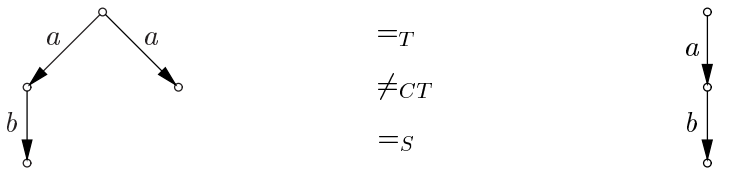
Also note the close link between the constructors of the modal formulas (corresponding to the three clauses in Definition 3.1) and the types of observations according to the testing scenario: $\top$

represents the act of the observer of terminating the observation, regardless of whether the observed process has terminated, 0 represents the observation of deadlock (the display becomes empty), and $a\varphi$ represents the observation of a being displayed, followed by the observation φ.

Process Graph Characterization Let $g \in \mathbb{G}$ and $s \in \text{NODES}(g)$. Then $I(s) := \{a \in Act \mid \exists t : (s, a, t) \in \text{EDGES}(g)\}$ is the *menu* of s. $CT(g)$ can now be characterized as follows.

Proposition 3.2 $CT(g) = \{T(\pi) \mid \pi \in \text{PATHS}(g) \wedge I(end(\pi)) = \emptyset\}$. □

Classification Trivially $T \preceq CT$ (as in Figure 1). Counterexample 2 shows that the reverse



Counterexample 2: Trace and simulation equivalent, but not completed trace equivalent

does not hold: one has $T(left) = T(right) = \{\epsilon,\ a,\ ab\}$, whereas $CT(left) \neq CT(right)$ (since $a \in CT(left) - CT(right)$). Hence the two process graphs are identified in trace semantics but distinguished in completed trace semantics. Thus $T \prec CT$: on $\mathbb{G}$ completed trace semantics makes strictly less identifications than trace semantics.

Infinite processes Also for completed trace semantics one can distinguish a finitary and an infinitary variant. In terms of the testing scenario, the latter (CT^∞) postulates that observations may take an infinite amount of time.

Definition 3.2 Two processes p and q are *infinitary completed trace equivalent*, notation $p =^\infty_{CT} q$, if $CT(p) = CT(q)$ and $T^\infty(p) = T^\infty(q)$. Note that in this case also $T(p) = T(q)$.

Proposition 2.3 implies that for image finite processes CT and CT^∞ coincide, whereas Counterexample 1 shows that in general the two are different. In fact, $T \prec T^\infty \prec CT^\infty$ and $T \prec CT \prec CT^\infty$, and the two preceding counterexamples show that there are no further inclusions.

4 Failures semantics

Testing scenario The *failures 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 chooses an execution path that fits with its position in $(\mathbb{P}, \rightarrow)$, 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,

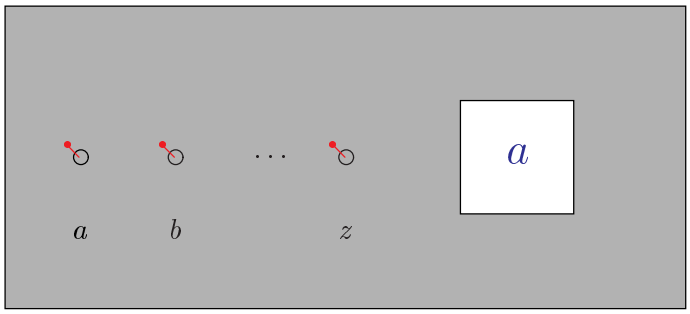


Figure 3: The failure trace machine

which can be recognized from the empty display. In this case the observer may record that after a certain sequence of actions σ, the set X of free actions is refused by the process. X is therefore called a *refusal set* and $\langle \sigma, X \rangle$ a *failure pair*. The set of all failure pairs of a process is called its *failure set*, and constitutes its observable behaviour.

Definition 4 $\langle \sigma, X \rangle \in Act^* \times \mathcal{P}(Act)$ is a *failure pair* of a process p, if there is a process q, such that $p \xrightarrow{\sigma} q$ and $I(q) \cap X = \emptyset$. Let $F(p)$ denote the set of failure pairs of p. Two processes p and q are *failures equivalent*, notation $p =_F q$, if $F(p) = F(q)$. In *failures semantics* (F) two processes are identified iff they are failures equivalent.

Note that $T(p)$ can be expressed in terms of $F(p)$: $T(p) = \{\sigma \in Act^* \mid \langle \sigma, \emptyset \rangle \in F(p)\}$; hence $p =_F q$ implies $T(p) = T(q)$.

Definition 4.1 For $p \in \mathbb{P}$ and $\sigma \in T(p)$, let $Cont_p(\sigma) = \{a \in Act \mid \sigma a \in T(p)\}$, the set of possible *continuations* of σ.

The following proposition says that the failure set $F(p)$ of a process p is completely determined by the set of failure pairs $\langle \sigma, X \rangle$ with $X \subseteq Cont_p(\sigma)$.

Proposition 4.1 Let $p \in \mathbb{P}$, $\sigma \in T(p)$ and $X \subseteq Act$. Then $\langle \sigma, X \rangle \in F(p) \Leftrightarrow \langle \sigma, X \cap Cont_p(\sigma) \rangle \in F(p)$.
Proof: If $p \xrightarrow{\sigma} q$ then $I(q) \subseteq Cont_p(\sigma)$. □

Modal Characterization

Definition 4.2 The set $\mathcal{L}_F$ of *failure formulas* over Act is defined recursively by:

- $\top \in \mathcal{L}_F$.
- $\widetilde{X} \in \mathcal{L}_F$ for $X \subseteq Act$.
- If $\varphi \in \mathcal{L}_F$ and $a \in Act$ then $a\varphi \in \mathcal{L}_F$.

The *satisfaction relation* $\models \subseteq \mathbb{P} \times \mathcal{L}_F$ is defined recursively by:

- $p \models \top$ for all $p \in \mathbb{P}$.
- $p \models \widetilde{X}$ if $I(p) \cap X = \emptyset$.
- $p \models a\varphi$ if for some $q \in \mathbb{P}$: $p \xrightarrow{a} q$ and $q \models \varphi$.

$\widetilde{X}$ represents the observation that the process refuses the set of actions X, i.e. that stagnation occurs in a situation where X is the set of actions allowed by the environment. Note that a failure formula satisfied by a process p represents either a trace (if it has the form $a_1a_2\cdots a_n\top$) or a failure pair (if it has the form $a_1a_2\cdots a_n\widetilde{X}$). Hence one has

Proposition 4.2 $p =_F q \;\Leftrightarrow\; \forall\varphi\in\mathcal{L}_F(p\models\varphi \Leftrightarrow q\models\varphi)$. □

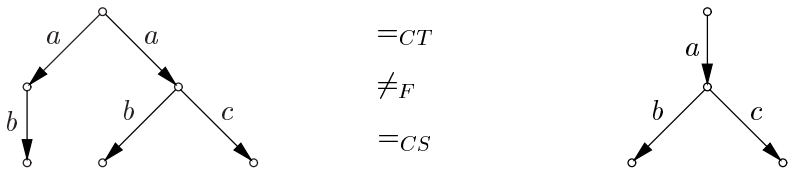
Process Graph Characterization Let $g\in\mathbb{G}$ and $\pi\in$ PATHS(g). Then $F(\pi) := \{\langle T(\pi), X\rangle \mid I(end(\pi))\cap X=\emptyset\}$ is the *failure set* of π. $F(g)$ can now be characterized as follows.

Proposition 4.3 $F(g) = \bigcup_{\pi\in\text{PATHS}(g)} F(\pi)$. □

Classification $CT \prec F$.
Proof: For "$CT \preceq F$" it suffices to show that also $CT(p)$ can be expressed in terms of $F(p)$:

$$CT(p) = \{\sigma\in Act^* \mid \langle\sigma, Act\rangle\in F(p)\}.$$

It also suffices to show that the modal language $\mathcal{L}_{CT}$ is a sublanguage of $\mathcal{L}_F$: $p\models 0 \Leftrightarrow p\models\widetilde{Act}$.
"$CT \not\succeq F$" follows from Counterexample 3: one has $CT(left) = CT(right) = \{ab,\ ac\}$, whereas $F(left)\neq F(right)$ (since $\langle a,\{b\}\rangle\in F(left) - F(right)$). □



$ab + a(b+c)$ $\qquad$ $a(b+c)$

Counterexample 3: Completed trace and completed simulation equivalent, but not failures equivalent

Alternative characterizations In De Nicola [13] several equivalences, that were proposed in Kennaway [28], Darondeau [12] and De Nicola & Hennessy [14], are shown to coincide with failures semantics on the domain of finitely branching transition systems without internal moves. For this purpose he uses the following alternative characterization of failures equivalence.

Definition 4.3 Write p *after* σ *MUST* X if for each $q\in\mathbb{P}$ with $p\xrightarrow{\sigma} q$ there is an $a\in I(q)$ with $a\in X$. 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.

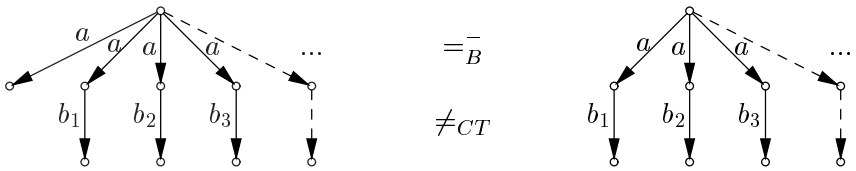
Proposition 4.4 Let $p,q\in\mathbb{P}$. Then $p\simeq q \;\Leftrightarrow\; p =_F q$.
Proof: p *after* σ *MUST* $X \;\Leftrightarrow\; \langle\sigma, X\rangle\notin F(p)$ [13]. □

In Hennessy [22], 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 $\langle\sigma, X\rangle\in Act^*\times\mathcal{P}(Act)$ for which there is a process q, such that $p\xrightarrow{\sigma} q$ and $I(q)\subseteq X\subseteq Cont_p(\sigma)$. It follows that for such processes *acceptance tree equivalence* coincides with failures equivalence.

Infinite processes For infinite processes, three versions of failures semantics can be distinguished.

Definition 4.4 Two processes p and q are *(finitary) failures equivalent* if $F(p) = F(q)$. p and q are *infinitary failures equivalent*, notation $p =_F^\infty q$, if $F(p) = F(q)$ and $T^\infty(p) = T^\infty(q)$. They are *finite-failures equivalent*, notation $p =_F^- q$, if $F^-(p) = F^-(q)$, where $F^-(p)$ denotes the set of failure pairs $\langle \sigma, X\rangle$ of p with X finite.

The original failures semantics of Brookes, Hoare & Roscoe [10] is F^-, i.e. what I call *finite-failures semantics*. They "adopt this view of distinguishability because [they] consider a *realistic* environment to be one that is at any time capable of performing only a finite number of events." In terms of the failures machine this means that at any time only finitely many switches can be set on free. Finitary failures semantics is the default version introduced at the beginning of this section. This can be regarded to be the semantics employed in Brookes & Roscoe [11] and Hoare [27]. Infinitary failures semantics was first discussed in Bergstra, Klop & Olderog [7]; it was proposed as a semantics for CSP in Roscoe [39]. The difference between the testing scenarios for F and F^∞ is that only the latter allows observations of infinite duration. Obviously, $F^- \preceq F \preceq F^\infty$. That the latter inclusion is strict follows from Counterexample 1; Counterexample 4 shows that also the former is strict: one has $F^-(\mathit{left}) = F^-(\mathit{right})$, whereas $F(\mathit{left}) \neq F(\mathit{right})$. In fact even



Counterexample 4: HML- and finite-failures equivalent, but not completed trace equivalent

$CT(\mathit{left}) \neq CT(\mathit{right})$, as $a \in CT(\mathit{left}) - CT(\mathit{right})$. Thus, although $T \prec F^-$, $CT \prec F$ and $CT^\infty \prec F^\infty$, CT and F^- are independent, as are CT^∞ and F.

In addition to the three variants of Definition 4.4 one could also define a version of failures semantics based on infinite traces and finite refusal sets. Such a semantics would distinguish the two graphs of Counterexample 1, but identify the ones of Counterexample 4. As this semantics does not occur in the literature, and has no clear advantages over the other variants, I will not further consider it here.

Proposition 4.5 Let p en q be image finite processes. Then $p =_F^- q \Leftrightarrow p =_F q \Leftrightarrow p =_F^\infty q$.
Proof: "$\Leftarrow$" has been established for all processes, and the second "$\Rightarrow$" follows immediately from Proposition 2.3 (as $p =_F q \Rightarrow p =_T q \Rightarrow p =_T^\infty q$). So it remains to show that $p \neq_F q \Rightarrow p \neq_F^- q$. Suppose $F(p) \neq F(q)$, say there is a failure pair $\langle \sigma, X\rangle \in F(p) - F(q)$. By the image finiteness of q there are only finitely many processes r_i with $q \xrightarrow{\sigma} r_i$, and for each of them there is an action $a_i \in I(r_i) \cap X$ (as otherwise $\langle \sigma, X\rangle$ 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 $\langle \sigma, Y\rangle \in F^-(p)$. On the other hand, $a_i \in I(r_i) \cap Y$ for all r_i, so $\langle \sigma, Y\rangle \notin F^-(q)$. □

Exercise 1 Give an example of two processes p and q, one of which is image finite, such that $p =_F^- q$ but $p \neq_F q$.

Thus, in the first statement of Proposition 4.5 it is necessary that both processes are image finite. For the subclass of finitely branching processes a stronger result can be obtained.

Proposition 4.6 Let $p, q \in \mathbb{P}$ and p is finitely branching. Then $p =_F^- q \Leftrightarrow p =_F q$.
Proof: Suppose $p =_F^- q$. As p is finitely branching, $Cont_p(\sigma)$ is finite for all $\sigma \in T(p)$. And as $T(q) = T(p)$, $Cont_q(\sigma) = Cont_p(\sigma)$, which is finite, for all $\sigma \in T(q)$. Now for processes p with this property, $F(p)$ is completely determined by $F^-(p)$, as follows from Proposition 4.1. □

The second statement of Proposition 4.5 does not allow such a strengthening:

Exercise 2 Give an example of two processes p and q, one of which is finitely branching, such that $p =_F q$ but $p \neq_F^\infty q$.

5 Failure trace semantics

Testing scenario The *failure trace machine* has the same layout as the failures 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* (FT).

For image finite processes failure trace semantics is exactly the equivalence that originates from Phillips notion of *refusal testing* [36]. (Image infinite processes are not considered in [36].) There it is called *refusal equivalence*.

Definition 5

- The *refusal relations* $\xrightarrow{X}$ for $X \subseteq Act$ are defined by: $p \xrightarrow{X} q$ iff $p = q$ and $I(p) \cap X = \emptyset$.
  $p \xrightarrow{X} q$ 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* $\stackrel{\sigma}{\longrightarrow}$ 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 $p \stackrel{\sigma}{\longrightarrow} q$. Let $FT(p)$ denote the set of failure traces of p. Two processes p and q are *failure trace equivalent*, notation $p =_{FT} q$, if $FT(p) = FT(q)$.

Exercises

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

Modal Characterization

Definition 5.1 The set $\mathcal{L}_{FT}$ of *failure trace formulas* over Act is defined recursively by:

- $\top \in \mathcal{L}_{FT}$.
- If $\varphi \in \mathcal{L}_{FT}$ and $X \subseteq Act$ then $\widetilde{X}\varphi \in \mathcal{L}_{FT}$.
- If $\varphi \in \mathcal{L}_{FT}$ and $a \in Act$ then $a\varphi \in \mathcal{L}_{FT}$.

The *satisfaction relation* $\models \subseteq \mathbb{P} \times \mathcal{L}_{FT}$ is defined recursively by:

- $p \models \top$ for all $p \in \mathbb{P}$.
- $p \models \widetilde{X}\varphi$ if $I(p) \cap X = \emptyset$ and $p \models \varphi$.
- $p \models a\varphi$ if for some $q \in \mathbb{P}$: $p \stackrel{a}{\longrightarrow} q$ and $q \models \varphi$.

$\widetilde{X}\varphi$ represents the observation that the process refuses the set of actions X, followed by the observation φ. A modal failure trace formula satisfied by a process p represents exactly a failure trace as defined above. Hence one has

Proposition 5.1 $p =_{FT} q \;\Leftrightarrow\; \forall \varphi \in \mathcal{L}_{FT}(p \models \varphi \Leftrightarrow q \models \varphi)$. □

Process Graph Characterization Let $g \in \mathbb{G}$ and $\pi : s_0 \stackrel{a_1}{\longrightarrow} s_1 \stackrel{a_2}{\longrightarrow} \cdots \stackrel{a_n}{\longrightarrow} s_n \in \textsc{paths}(g)$. Then the *failure trace set* of π, $FT(\pi)$, is the smallest subset of $(Act \cup \mathcal{P}(Act))^*$ satisfying

- $(Act - I(s_0))a_1(Act - I(s_1))a_2 \cdots a_n(Act - I(s_n)) \in FT(\pi)$,
- $\sigma X \rho \in FT(\pi) \Rightarrow \sigma\rho \in FT(\pi)$,
- $\sigma X \rho \in FT(\pi) \Rightarrow \sigma X X \rho \in FT(\pi)$,
- $\sigma X \rho \in FT(\pi) \wedge Y \subset X \Rightarrow \sigma Y \rho \in FT(\pi)$.

$FT(g)$ can now be characterized as follows.

Proposition 5.2 $FT(g) = \bigcup_{\pi \in \textsc{paths}(g)} FT(\pi)$. □

Proposition 5.2 yields a technique for deciding that two process graphs are failure trace equivalent, without calculating their entire failure trace set.
Let $g, h \in \mathbb{G}$, $\pi : s_0 \xrightarrow{a_1} s_1 \xrightarrow{a_2} \cdots \xrightarrow{a_n} s_n \in \textsc{paths}(g)$ and $\pi' : t_0 \xrightarrow{b_1} t_1 \xrightarrow{b_2} \cdots \xrightarrow{b_m} t_m \in \textsc{paths}(h)$. Path π' is a *failure trace augmentation* of π, notation $\pi \leq_{FT} \pi'$, if $FT(\pi) \subseteq FT(\pi')$. This is the case exactly when $n = m$, $a_i = b_i$ and $I(t_i) \subseteq I(s_i)$ for $i = 1, ..., n$. From this the following can be concluded.

Corollary 5.1 Two process graphs $g, h \in \mathbb{G}$ are failure trace equivalent iff

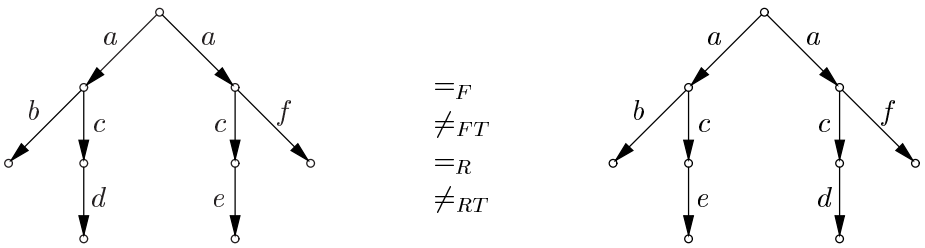
- for any path $\pi \in \textsc{paths}(g)$ in g there is a $\pi' \in \textsc{paths}(h)$ such that $\pi \leq_{FT} \pi'$
- and for any path $\pi \in \textsc{paths}(g)$ in h there is a $\pi' \in \textsc{paths}(g)$ such that $\pi \leq_{FT} \pi'$.

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

Classification $F \prec FT$.
Proof: For "$F \preceq FT$" it suffices to show that $F(p)$ can be expressed in terms of $FT(p)$:

$$\langle \sigma, X \rangle \in F(p) \quad \Leftrightarrow \quad \sigma X \in FT(p).$$

"$F \not\succeq FT$" follows from Counterexample 5; see Section 7 for details. □



$a(b + cd) + a(f + ce)$ $\qquad$ $a(b + ce) + a(f + cd)$

Counterexample 5: Failures and ready equivalent, but not failure trace or ready trace equivalent

Infinite processes As for failures semantics, three variants of failure trace semantics for infinite processes can be defined. Besides the default version (FT) there is an infinitary version (FT^∞), motivated by observations that may last forever, and a finite version (FT^-), motivated by an observer that may only set finitely many switches on free at any time.

Definition 5.2 $\sigma_1\sigma_2 \cdots \in (Act \cup \mathcal{P}(Act))^\infty$ is an *infinite failure trace* of a process $p \in \mathbb{P}$, if there are processes $p_1, p_2, ...$ such that $p \xrightarrow{\sigma_1} p_1 \xrightarrow{\sigma_2} \cdots$. Let $FT^\infty(p)$ denote the set of infinite failure traces of p. Two processes p and q are *infinitary failure trace equivalent*, notation $p =^\infty_{FT} q$, if $FT^\infty(p) = FT^\infty(q)$ and $FT(p) = FT(q)$. They are *finite-failure trace equivalent*, notation $p =^-_{FT} q$, if $FT^-(p) = FT^-(q)$, where $FT^-(p)$ denotes the set of failure traces of p in which all refusal sets are finite.

Clearly, $FT^- \prec FT \prec FT^\infty$; Counterexamples 1 and 4 show that the inclusions and strict. One also has $F^- \prec FT^-$, $F \prec FT$ and $F^\infty \prec FT^\infty$; here strictness follows from Counterexample 5.

Proposition 5.3 Let p en q be image finite processes. Then $p =^{-}_{FT} q \Leftrightarrow p =_{FT} q \Leftrightarrow p =^{\infty}_{FT} q$.
Proof: "$p =^{-}_{FT} q \Leftarrow p =_{FT} q \Leftarrow p =^{\infty}_{FT} q$" holds for all processes.
Note that the definition of $FT(p)$ is exactly like the definition of $T(p)$, except that the failure trace relations are used instead of the generalized action relations; the same relation exists between $FT^{\infty}(p)$ and $T^{\infty}(p)$. Moreover, a process $p \in \mathbb{P}$ is image finite in terms of the failure trace relations on $\mathbb{P}$ iff it is image finite in terms of terms of the (generalized) action relations on $\mathbb{P}$, as defined in Definition 1.2. Hence "$p =_{FT} q \Rightarrow p =^{\infty}_{FT} q$" follows immediately from Proposition 2.3.
"$p =^{-}_{FT} q \Rightarrow p =_{FT} q$": Suppose $FT(p) \neq FT(q)$, say $FT(p) - FT(q) \neq \emptyset$. Let σ be a failure trace in $FT(p) - FT(q)$ with at least one infinite refusal set. I will show that there must be a failure trace in $FT(p) - FT(q)$ with strictly fewer infinite refusal sets than σ. By applying this result a finite number of times, a failure trace $\rho \in FT(p) - FT(q)$ is found without infinite refusal sets, showing that $FT^{-}(p) \neq FT^{-}(q)$.

So let $\sigma = \sigma_1 X \sigma_2 \in FT(p) - FT(q)$ with X an infinite refusal set. Clearly $\sigma_1\sigma_2 \in FT(p)$. By the image finiteness of q there are only finitely many pairs of processes r_i, s_i with $q \xrightarrow{\sigma_1} r_i \xrightarrow{\sigma_2} s_i$, and for each of them there is an action $a_i \in I(r_i) \cap X$ (as otherwise $\sigma_1 X \sigma_2$ would be a failure trace of q). Let Y be the set of all those a_i's. Then Y is finite. As Y is a subset of X, one has $\sigma_1 Y \sigma_2 \in FT(p)$. On the other hand, $a_i \in I(r_i) \cap Y$ for all r_i, so $\sigma_1 Y \sigma_2 \notin FT(q)$. □

Unlike the situation for failures semantics, in the first statement of Proposition 5.3 it is not necessary that *both* processes are image finite.

Proposition 5.4 Let $p, q \in \mathbb{P}$ and p is image finite. Then $p =^{-}_{FT} q \Leftrightarrow p =_{FT} q$.
Proof: More difficult, and omitted here. □

The second statement of Proposition 5.3 does not allow such a strengthening; Exercise 2 in Section 4 easily lifts to failure trace semantics.

6 Ready trace semantics

Testing scenario 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

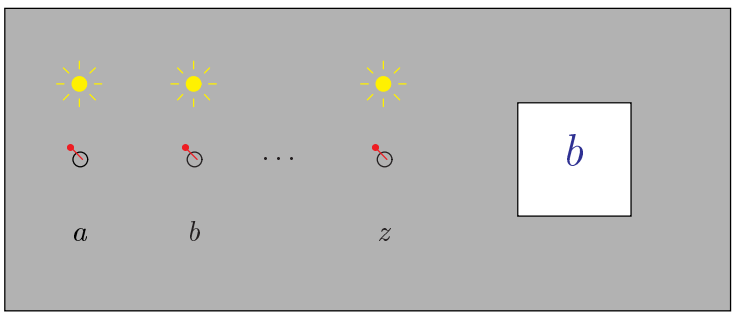


Figure 4: The ready trace machine

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 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.

Definition 6

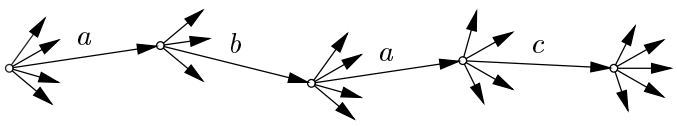
- The *ready trace relations* $\stackrel{\sigma}{\ast\!\!\longrightarrow}$ for $\sigma \in (Act \cup \mathcal{P}(Act))^*$ are defined recursively by:

  1. $p \stackrel{\epsilon}{\ast\!\!\longrightarrow} p$, for any process p.
  2. $p \stackrel{a}{\longrightarrow} q$ implies $p \stackrel{a}{\ast\!\!\longrightarrow} q$.
  3. $p \stackrel{X}{\ast\!\!\longrightarrow} q$ with $X \subseteq Act$ whenever $p = q$ and $I(p) = X$.
  4. $p \stackrel{\sigma}{\ast\!\!\longrightarrow} q \stackrel{\rho}{\ast\!\!\longrightarrow} r$ implies $p \stackrel{\sigma\rho}{\ast\!\!\longrightarrow} r$.

  The special arrow $\stackrel{\sigma}{\ast\!\!\longrightarrow}$ had to be used, since further overloading of $\stackrel{\sigma}{\longrightarrow}$ 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 $p \stackrel{\sigma}{\ast\!\!\longrightarrow} q$. Let $RT(p)$ denote the set of ready traces of p. Two processes p and q are *ready trace equivalent*, notation $p =_{RT} q$, if $RT(p) = RT(q)$. In *ready trace semantics* (RT) two processes are identified iff they are ready trace equivalent.

In Baeten, Bergstra & Klop [4], Pnueli [37] and Pomello [38] ready trace semantics was defined slightly differently. By Proposition 6.1 below, their definition yields the same equivalence as mine.



Definition 6.1 $X_0 a_1 X_1 a_2 \cdots a_n X_n \in \mathcal{P}(Act) \times (Act \times \mathcal{P}(Act))^*$ is a *normal ready trace* of a process p, if there are processes $p_1, ..., p_n$ such that $p \stackrel{a_1}{\longrightarrow} p_1 \stackrel{a_2}{\longrightarrow} \cdots \stackrel{a_n}{\longrightarrow} p_n$ and $I(p_i) = X_i$ for $i = 1, ..., n$. 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, 37, 38] if $RT_N(p) = RT_N(q)$.

Proposition 6.1 Let $p, q \in \mathbb{P}$. 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. □

Modal Characterization

Definition 6.2 The set $\mathcal{L}_{RT}$ of *ready trace formulas* over Act is defined recursively by:

- $\top \in \mathcal{L}_{RT}$.
- If $\varphi \in \mathcal{L}_{RT}$ and $X \subseteq Act$ then $X\varphi \in \mathcal{L}_{RT}$.
- If $\varphi \in \mathcal{L}_{RT}$ and $a \in Act$ then $a\varphi \in \mathcal{L}_{RT}$.

The *satisfaction relation* $\models \subseteq \mathbb{P} \times \mathcal{L}_{RT}$ is defined recursively by:

- $p \models \top$ for all $p \in \mathbb{P}$.
- $p \models X\varphi$ if $I(p) = X$ and $p \models \varphi$.
- $p \models a\varphi$ if for some $q \in \mathbb{P}$: $p \xrightarrow{a} q$ and $q \models \varphi$.

$X\varphi$ represents the observation of a menu, followed by the observation φ. A ready trace formula satisfied by a process p represents exactly a ready trace in Definition 6. Hence one has

Proposition 6.2 $p =_{RT} q \;\Leftrightarrow\; \forall \varphi \in \mathcal{L}_{RT}(p \models \varphi \Leftrightarrow q \models \varphi)$. □

Process Graph Characterization Let $g \in \mathbb{G}$ and $\pi : s_0 \xrightarrow{a_1} s_1 \xrightarrow{a_2} \cdots \xrightarrow{a_n} s_n \in \textsc{paths}(g)$. Then the *ready trace set* of π, $RT(\pi)$, is the smallest subset of $(Act \cup \mathcal{P}(Act))^*$ satisfying

- $I(s_0)a_1I(s_1)a_2 \cdots a_nI(s_n) \in RT(\pi)$,
- $\sigma X \rho \in RT(\pi) \Rightarrow \sigma\rho \in RT(\pi)$,
- $\sigma X \rho \in RT(\pi) \Rightarrow \sigma X X \rho \in RT(\pi)$.

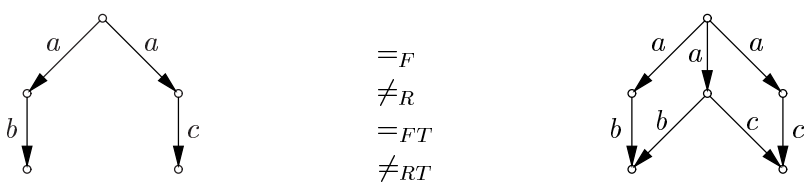
$RT(g)$ can now be characterized as follows.

Proposition 6.3 $RT(g) = \bigcup_{\pi \in \textsc{paths}(g)} RT(\pi)$. □

Classification $FT \prec RT$.
Proof: For "$FT \preceq RT$" it suffices to show that $FT(p)$ can be expressed in terms of $RT(p)$:

$\sigma = \sigma_1\sigma_2 \cdots \sigma_n \in FT(p)$ $(\sigma_i \in Act \cup \mathcal{P}(Act)) \;\Leftrightarrow$
$\exists \rho = \rho_1\rho_2 \cdots \rho_n \in RT(p)$ $(\rho_i \in Act \cup \mathcal{P}(Act))$ such that for $i = 1, ..., n$ either
$\sigma_i = \rho_i \in Act$ or $\sigma_i, \rho_i \subseteq Act$ and $\sigma_i \cap \rho_i = \emptyset$.

"$FT \not\succeq RT$" follows from Counterexample 6; see Section 7 for details. □



$ab + ac$ $\qquad$ $ab + a(b + c) + ac$

Counterexample 6: Failures and failure trace equivalent, but not ready or ready trace equivalent

Infinite processes An infinitary version of ready trace semantics (RT^∞) is defined analogously to infinitary failure trace semantics. A finite version is not so straightforward; a definition will be proposed in the next section.

Definition 6.3 $\sigma_1\sigma_2\cdots \in (Act \cup \mathcal{P}(Act))^\infty$ is an *infinite ready trace* of a process $p \in \mathbb{P}$, if there are processes $p_1, p_2, \ldots$ such that $p \stackrel{\sigma_1}{\nrightarrow} p_1 \stackrel{\sigma_2}{\nrightarrow} \cdots$. Let $RT^\infty(p)$ denote the set of infinite ready traces of p. Two processes p and q are *infinitary ready trace equivalent*, notation $p =^\infty_{RT} q$, if $RT^\infty(p) = RT^\infty(q)$ and $RT(p) = RT(q)$.

Clearly, $RT \prec RT^\infty$; Counterexample 1 shows that the inclusion is strict. Moreover $FT^\infty \prec RT^\infty$.

Proposition 6.4 Let p en q be image finite processes. Then $p =_{RT} q \Leftrightarrow p =^\infty_{RT} q$.
Proof: Exactly as the corresponding part of Proposition 5.3. □

7 Readiness semantics

Testing scenario The *readiness machine* has the same layout as the ready trace machine, but, like the failures 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 7 $\langle \sigma, X\rangle \in Act^* \times \mathcal{P}(Act)$ is a *ready pair* of a process p, if there is a process q, such that $p \stackrel{\sigma}{\longrightarrow} q$ and $I(q) = X$. Let $R(p)$ denote the set of ready pairs of p. Two processes p and q are *ready equivalent*, notation $p =_R q$, if $R(p) = R(q)$. In *readiness semantics* (R) two processes are identified iff they are ready equivalent.

Modal Characterization

Definition 7.1 The set $\mathcal{L}_R$ of *readiness formulas* over Act is defined recursively by:

- $\top \in \mathcal{L}_R$.
- $X \in \mathcal{L}_R$ for $X \subseteq Act$.
- If $\varphi \in \mathcal{L}_R$ and $a \in Act$ then $a\varphi \in \mathcal{L}_R$.

The *satisfaction relation* $\models \subseteq \mathbb{P} \times \mathcal{L}_R$ is defined recursively by:

- $p \models \top$ for all $p \in \mathbb{P}$.
- $p \models X$ if $I(p) = X$.
- $p \models a\varphi$ if for some $q \in \mathbb{P}$: $p \stackrel{a}{\longrightarrow} q$ and $q \models \varphi$.

X represents the observation of a menu. A readiness formula satisfied by a process p represents either a trace (if it has the form $a_1a_2\cdots a_n\top$) or a ready pair (if it has the form $a_1a_2\cdots a_nX$). Hence one has

Proposition 7.1 $p =_R q \;\Leftrightarrow\; \forall\varphi \in \mathcal{L}_R(p \models \varphi \Leftrightarrow q \models \varphi)$. □

Process Graph Characterization Let $g \in \mathbb{G}$ and $\pi \in \text{PATHS}(g)$. The *ready pair* of π is given by $R(\pi) := \langle T(\pi), I(end(\pi))\rangle$. $R(g)$ can now be characterized by:

Proposition 7.2 $R(g) = \{R(\pi) \mid \pi \in \text{PATHS}(g)\}$. □

Classification $F \prec R \prec RT$, but R and FT are independent.
Proof: : For "$F \preceq R$" it suffices to show that $F(p)$ can be expressed in terms of $R(p)$:

$$\langle \sigma, X\rangle \in F(p) \quad \Leftrightarrow \quad \exists Y \subseteq Act : \langle \sigma, Y\rangle \in R(p) \ \wedge \ X \cap Y = \emptyset.$$

For "$R \preceq RT$" it suffices to show that $R(p)$ can be expressed in terms of $RT(p)$:

$$\langle \sigma, X\rangle \in R(p) \quad \Leftrightarrow \quad \sigma X \in RT(p).$$

"$R \not\preceq FT$" (and hence "$R \not\preceq RT$" and "$F \not\preceq FT$") follows from Counterexample 5, in which $R(left) = R(right)$ but $FT(left) \neq FT(right)$. The first statement follows with Proposition 7.2. Both graphs have 9 paths starting from the root, and hence 9 ready pairs. These are easily seen to be the same at both sides; in the second graph only 4 ready pairs swopped places. The second statement follows since $a\{b\}ce \in FT(left) - FT(right)$.

"$R \not\succeq FT$" (and hence "$R \not\succeq F$" and "$RT \not\succeq FT$") follows from Counterexample 6, in which $FT(left) = FT(right)$ but $R(left) \neq R(right)$. The first statement follows from Corollary 5.1, 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 $\langle a, \{b, c\}\rangle \in R(right) - R(left)$. □

Possible-futures and acceptance-refusal semantics Readiness semantics was proposed by Olderog & Hoare [34]. Two preliminary versions stem from Rounds & Brookes [40]: in *possible-futures semantics* (PF) the menu consists of the entire trace set of the 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 Section 4 are observable.

Definition 7.2 $\langle \sigma, X\rangle \in Act^* \times \mathcal{P}(Act^*)$ 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*, notation $p =_{PF} q$, if $PF(p) = PF(q)$.

The modal and process graph characterizations of possible-future semantics are straightforward, but a plausible testing scenario has not been proposed. Trivially $R \preceq PF$. That the reverse does not hold, and even that $PF \not\preceq RT$, will follow from Counterexample 9.

Exercise 6 Provide a counterexample showing that $FT \not\preceq PF$.

Thus possible-future semantics is incomparable with failure trace and ready trace semantics.

Definition 7.3 $\langle \sigma, X, Y\rangle \in Act^* \times \mathcal{P}(Act) \times \mathcal{P}(Act)$ 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 $Y \cap I(q) = \emptyset$. Let $AR(p)$ denote the set of acceptance-refusal triples of p. Two processes p and q are *acceptance-refusal equivalent*, notation $p =_{AR} q$, if $AR(p) = AR(q)$.

The modal and process graph characterizations are again straightforward. A motivating testing scenario would be the same as for readiness semantics, except that at any time only finitely many switches can be set on free, and only finitely many lamps can be investigated in a finite amount of time. Clearly $p =_R q \Rightarrow p =_{AR} q$, for

$$AR(p) = \{\langle \sigma, X, Y\rangle \mid \exists \langle \sigma, Z\rangle \in R(p) \mid X, Y \text{ finite } \wedge X \subseteq Z \wedge Y \cap Z = \emptyset\}.$$

That this implication is strict follows from Counterexample 4. It is not difficult to see that for finitely branching processes acceptance-refusal equivalence coincides with ready equivalence: $\langle \sigma, X\rangle$ is a ready pair of a process p iff p has an acceptance-refusal triple $\langle \sigma, X, Y\rangle$ with $X \cup Y = Cont(\sigma)$ (cf. Definition 4.1).

Infinite processes Note that if in Definition 7.3 the sets X and Y are allowed to be infinite the resulting equivalence would we ready equivalence again. Namely $\langle \sigma, X\rangle$ is a ready pair of a process p iff p has such an acceptance-refusal triple $\langle \sigma, X, Act - Y\rangle$. Thus acceptance-refusal semantics can be regarded as the finite variant of readiness semantics, and will therefore be denoted R^-. The infinitary variant of readiness semantics (R^∞), motivated by observations that may last forever, is defined analogously to F^∞:

Definition 7.4 p and q are *infinitary ready equivalent* if $R(p) = R(q)$ and $T^\infty(p) = T^\infty(q)$.

Clearly, $R \prec R^\infty$; by Counterexample 1 the inclusion is strict. Moreover, $F^\infty \prec R^\infty \prec RT^\infty$.

Proposition 7.3 Let p en q be image finite processes. Then $p =_R q \Leftrightarrow p =_R^\infty q$.
Proof: "$\Leftarrow$" has been established for all processes, and the second "$\Rightarrow$" follows immediately from Proposition 2.3 (as $p =_R q \Rightarrow p =_T q \Rightarrow p =_T^\infty q$). □

Proposition 7.4 Let $p, q \in \mathbb{P}$ and p is image finite. Then $p =_{AR} q \Leftrightarrow p =_R q$.
Proof: "$\Leftarrow$" holds for all process. I will prove "$\Rightarrow$" assuming that p has the property that for any $\sigma \in Act^*$ there are only finitely many ready pairs $\langle \sigma, X\rangle \in R(p)$. This property (call it *RIF*) is clearly implied by image finiteness. So suppose p has the RIF property and $AR(p) = AR(q)$. I will show that $R(p) = R(q)$.

Suppose $\langle \sigma, Y\rangle \notin R(p)$. By RIF there are only finitely many ready pairs $\langle \sigma, X_i\rangle \in R(p)$. For each of them choose an action $a_i \in Y - X_i$ or $b_i \in X_i - Y$. Let U be the set of all those a_i's, and V the set of the b_i's. Then $\langle \sigma, U, V\rangle \notin AR(p) = AR(q)$ and hence $\langle \sigma, Y\rangle \notin R(q)$.

It follows that $R(q) \subseteq R(p)$, and thus q has the property RIF as well. Now the same argument applies in the other direction, yielding $R(p) \subseteq R(q)$. □

Inspired by the definition of R^-, a finite version of ready trace semantics (RT^-) can be defined likewise. Here I will just give its modal characterization.

Definition 7.5 The set $\mathcal{L}_{RT}^-$ of *finite ready trace formulas* over Act is given by:

- $\top \in \mathcal{L}_{RT}^-$.
- If $\varphi \in \mathcal{L}_{RT}^-$ and $X \subseteq_{fin} Act$ then $X\varphi \in \mathcal{L}_{RT}^-$ and $\widetilde{X}\varphi \in \mathcal{L}_{RT}^-$.
- If $\varphi \in \mathcal{L}_{RT}^-$ and $a \in Act$ then $a\varphi \in \mathcal{L}_{RT}^-$.

The *satisfaction relation* $\models \subseteq \mathbb{P} \times \mathcal{L}_{RT}^-$ is defined recursively by:

- $p \models \top$ for all $p \in \mathbb{P}$.
- $p \models X\varphi$ if $X \subseteq I(p)$ and $p \models \varphi$.
- $p \models \widetilde{X}\varphi$ if $I(p) \cap X = \emptyset$ and $p \models \varphi$.
- $p \models a\varphi$ if for some $q \in \mathbb{P}$: $p \xrightarrow{a} q$ and $q \models \varphi$.

Processes p and q are *finite-ready trace equivalent*, notation $p =^{-}_{RT} q$, if $\forall \varphi \in \mathcal{L}^{-}_{RT}(p \models \varphi \Leftrightarrow q \models \varphi)$.

As these formulas are expressible in terms of the ones of Definition 6.2, one has $RT^{-} \prec RT$; Counterexample 4 shows that the inclusion is strict. Also $FT^{-} \prec RT^{-}$ and $F^{-} \prec R^{-} \prec RT^{-}$.

Proposition 7.5 Let $p, q \in \mathbb{P}$ and p is image finite. Then $p =^{-}_{RT} q \Leftrightarrow p =_{RT} q$.
Proof: "$\Leftarrow$" holds for all process. "$\Rightarrow$" follows just as in Proposition 7.4, using the property that for any $a_1 a_2 \cdots a_n \in Act^{\omega}$ there are only finitely many normal ready traces $X_0 a_1 X_1 a_2 \cdots a_n X_n \in R(p)$.

Unlike the semantics T to RT, possible-futures semantics distinguishes between the two processes of Counterexample 1: $\langle a, a^* \rangle \in PF(right) - PF(left)$. Still, $T^{\infty} \not\prec PF$, as can be seen from the variant of Counterexample 1 in which the left-hand process is appended to the endnodes of both processes. The so obtained systems have the same possible futures, namely $\{\langle a^n, a^* \rangle \mid n \in \mathbb{N}\}$, but only the right-hand side has an infinite trace.

For the sake of completeness I include a definition of infinitary possible-futures semantics (PF^{∞}), such that $PF \prec PF^{\infty}$ and $R^{\infty} \prec PF^{\infty}$. A finite variant of PF has not been explored.

Definition 7.6 $\langle \sigma, X \rangle \in Act^* \times \mathcal{P}(Act^*)$ is an *infinitary possible future* of a process p, if there is a process q, such that $p \xrightarrow{\sigma} q$ and $T(q) \cup T^{\infty}(q) = X$. Let $PF^{\infty}(p)$ denote the set of infinitary possible futures of p. Two processes p and q are *infinitary possible-futures equivalent*, notation $p =^{\infty}_{PF} q$, if $PF^{\infty}(p) = PF^{\infty}(q)$.

8 Simulation semantics

The following concept of *simulation*, occurs frequently in the literature (see e.g. Park [35]).

Definition 8 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 $p' R q'$.

Process p *can be simulated by* q, notation $p \subseteq\!\!\rightarrow q$, if there is a simulation R with pRq.
p and q are *similar*, notation $p \rightleftarrows q$, if $p \subseteq\!\!\rightarrow q$ and $q \subseteq\!\!\rightarrow p$.

Proposition 8.1 Similarity is an equivalence relation on the domain of processes.
Proof: Symmetry is immediate, so it has to be checked that $p \subseteq\!\!\rightarrow p$, and $p \subseteq\!\!\rightarrow q \wedge q \subseteq\!\!\rightarrow r \Rightarrow p \subseteq\!\!\rightarrow r$.

- 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 $R;S$, defined by $x(R;S)z$ iff $\exists y :\ xRy \ \wedge\ ySz$, is a simulation with $p(R;S)r$. □

Hence the relation will be called *simulation equivalence*. In *simulation semantics* (S) two processes are identified iff they are simulation equivalent.

Testing scenario and modal characterization The testing scenario for 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 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.

Definition 8.1 The class $\mathcal{L}_S$ of *simulation formulas* over Act is defined recursively by:

- If I is a set and $\varphi_i \in \mathcal{L}_S$ for $i \in I$ then $\bigwedge_{i\in I} \varphi_i \in \mathcal{L}_S$.
- If $\varphi \in \mathcal{L}_S$ and $a \in Act$ then $a\varphi \in \mathcal{L}_S$.

The *satisfaction relation* $\models \subseteq \mathbb{P} \times \mathcal{L}_S$ is defined recursively by:

- $p \models \bigwedge_{i\in I} \varphi_i$ if $p \models \varphi_i$ for all $i \in I$.
- $p \models a\varphi$ if for some $q \in \mathbb{P}$: $p \xrightarrow{a} q$ and $q \models \varphi$.

Let $S(p)$ denote the class of simulation formulas satisfied by the process p: $S(p) = \{\varphi \in \mathcal{L}_S \mid p \models \varphi\}$. Write $p \sqsubseteq_S q$ if $S(p) \subseteq S(q)$ and $p =_S q$ if $S(p) = S(q)$.

Write $\top$ for $\bigwedge_{i\in\emptyset} \varphi_i$, and $\varphi_1 \wedge \varphi_2$ for $\bigwedge_{i\in\{1,2\}} \varphi_i$. It turns out that $\mathcal{L}_T$ is a sublanguage of $\mathcal{L}_S$.

Proposition 8.2 $p \subseteq\!\!\!\rightarrow q \Leftrightarrow p \sqsubseteq_S q$. Hence $p \rightleftarrows q \Leftrightarrow p =_S q$.
Proof: For "$\Rightarrow$" I have to prove that for any simulation R and for all $\varphi \in \mathcal{L}_S$ one has

$$pRq \Rightarrow (p \models \varphi \Rightarrow q \models \varphi).$$

I will do so with structural induction on φ. Suppose pRq.

- Let $p \models a\varphi$. Then there is a $p' \in \mathbb{P}$ with $p \xrightarrow{a} p'$ and $p' \models \varphi$. As R is a simulation, there must be a $q' \in \mathbb{P}$ with $q \xrightarrow{a} q'$ and $p'Rq'$. So by induction $q' \models \varphi$, and hence $q \models a\varphi$.
- $p \models \bigwedge_{i\in I} \varphi_i \Leftrightarrow \forall i \in I (p \models \varphi_i) \stackrel{\text{ind.}}{\Longrightarrow} \forall i \in I (q \models \varphi_i) \Leftrightarrow q \models \bigwedge_{i\in I} \varphi_i$.

For "$\Leftarrow$" it suffices to establish that $\sqsubseteq_S$ is a simulation.

- Suppose $p \sqsubseteq_S q$ and $p \xrightarrow{a} p'$. I have to show that $\exists q' \in \mathbb{P}$ with $q \xrightarrow{a} q'$ and $p' \sqsubseteq_S q'$. Let Q' be

  $$\{q' \in \mathbb{P} \mid q \xrightarrow{a} q' \wedge p' \not\sqsubseteq_S q'\}.$$

  By Definition 1.1 Q' is a set. For every $q' \in Q'$ there is a formula $\varphi_{q'} \in S(p') - S(q')$. Now

  $$a \bigwedge_{q'\in Q'} \varphi_{q'} \in S(p) \subseteq S(q),$$

  so there must be a $q' \in \mathbb{P}$ with $q \xrightarrow{a} q'$ and $q' \notin Q'$, which had to be shown. □

Process Graph Characterization Simulation equivalence can also be characterized by means of relations between the nodes of two process graphs, rather than between process graphs themselves.

Definition 8.2 Let $g, h \in \mathbb{G}$. A *simulation* of g by h is a binary relation $R \subseteq \text{NODES}(g) \times \text{NODES}(h)$, satisfying:

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

This definition is illustrated in Figure 5. Solid lines indicates what is assumed, dashed lines what is required. It follows easily that $g \subseteq\!\!\!\rightarrow h$ iff there exists a simulation of g by h.

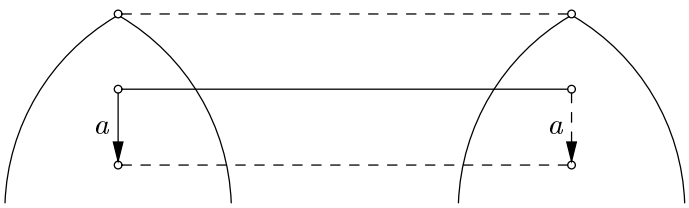
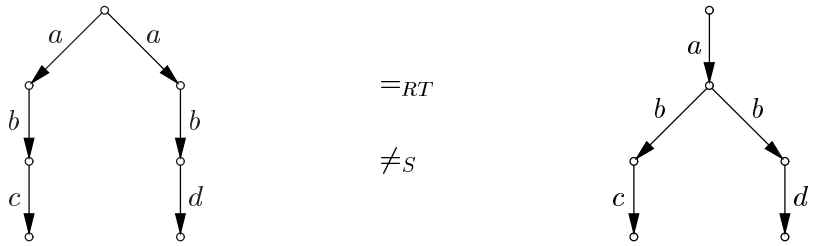


Figure 5: A simulation

Classification Simulation semantics (S) is finer than trace semantics ($T \prec S$), but independent of CT, F, R, FT and RT.
Proof: "$T \preceq S$" follows since $\mathcal{L}_T$ is a sublanguage of $\mathcal{L}_S$.
"$S \not\succeq CT$" (and hence "$S \not\succeq RT$" etc.) follows from Counterexample 2. There $left \not=_{CT} right$, although $left \rightleftarrows right$; the construction of the two simulations is left to the reader.
"$S \not\preceq RT$" (and hence "$S \not\preceq T$" etc.) follows from Counterexample 7 below. There $RT(left) = RT(left)$, but $S(left) \neq S(left)$. The first statement follows from Proposition 6.3 and the insight that it suffices to check the two ready traces contributed by the maximal paths; these are the same for both graphs. The second statement follows since $a(bc\top \wedge bd\top) \in S(right) - S(left)$. □



Counterexample 7: Ready trace equivalent, but not simulation equivalent

Infinite processes In order to make the testing scenario match its formalization in terms of the modal language $\mathcal{L}_S$ even for infinite processes, one has to assume that the amount of copies one can make at any time is infinite. Moreover, although no single copy can be tested forever, due to its infinite branching there may be no upperbound upon the duration of an observation.

One might consider an even more infinitary testing scenario by allowing observations to go on forever on some or all of the copies. However, this would not give rise to a more discriminating equivalence; ordinary simulation equivalences already preserves infinite traces.

Proposition 8.3 If $p \subsetneq q$ then $T^\infty(p) \subseteq T^\infty(q)$. Hence $T^\infty \prec S$.
Proof: Suppose R is a simulation with $p_0 R q_0$ and $a_1 a_2 \cdots \in T^\infty(p_0)$. Then there are $p_1, p_2, \ldots$ such that $p_0 \xrightarrow{a_1} p_1 \xrightarrow{a_2} \cdots$. With induction on $i \in \mathbb{N}$ it follows that there are processes q_{i+1} such that $q_i \xrightarrow{a_{i+1}} q_{i+1}$ and $p_{i+1} R q_{i+1}$. Hence $a_1 a_2 \cdots \in T^\infty(q_0)$. □

The most radical way to make the testing scenario finitary, is to allow only finitely many copies to be made in any state of the process. This also puts an upperbound on the duration of any

observation. Observations can now be modelled with simulation formulas in which the index sets I of the first clause of Definition 8.1 are always finite. The modal language containing such simulation formulas can equivalently be defined by splitting the construction $\bigwedge_{i\in I}$ into $\top$ and $\wedge$.

Definition 8.3 The set $\mathcal{L}_S^*$ of *finitary simulation formulas* over Act is defined recursively by:

- $\top \in \mathcal{L}_S^*$.
- If $\varphi, \psi \in \mathcal{L}_S^*$ then $\varphi \wedge \psi \in \mathcal{L}_S^*$.
- If $\varphi \in \mathcal{L}_S^*$ and $a \in Act$ then $a\varphi \in \mathcal{L}_S^*$.

The *satisfaction relation* $\models \subseteq \mathbb{P} \times \mathcal{L}_S^*$ is defined recursively by:

- $p \models \top$ for all $p \in \mathbb{P}$.
- $p \models \varphi \wedge \psi$ if $p \models \varphi$ and $p \models \psi$.
- $p \models a\varphi$ if for some $q \in \mathbb{P}$: $p \xrightarrow{a} q$ and $q \models \varphi$.

Let $S^*(p)$ denote the set of all finitary simulation formulas that are satisfied by the process p: $S^*(p) = \{\varphi \in \mathcal{L}_S^* \mid p \models \varphi\}$. Two processes p and q are *finitary simulation equivalent*, notation $p =_S^* q$, if $S^*(p) = S^*(q)$.

In contrast, the equivalence $\rightleftarrows$ of Definition 8 is then *infinitary simulation equivalence*. Note however, that contrary to the previous equivalences surveyed, the default version (the one meant when leaving out the adjective "finitary" or "infinitary") is the infinitary one. In general, I use the superscript $*$ for finitary versions and ∞ for infinitary versions. However, for the trace oriented equivalences (Sections 2–7) I leave out the $*$, and for the simulation oriented equivalences (Sections 8–12) I leave out the ∞.

The next proposition states that for image finite processes finitary and infinitary simulation equivalence coincide.

Proposition 8.4 Let $p, q \in \mathbb{P}$ be image finite processes. Then $p \rightleftarrows q \Leftrightarrow p =_S^* q$.
Proof: Exactly as the proof of Proposition 8.2, but for "$\Leftarrow$" one shows that the relation $\sqsubseteq_S^{i.f.}$ given by $p \sqsubseteq_S^{i.f.} q$ iff $p \sqsubseteq_S q$ and q is image finite is a simulation, using that, as q is image finite, Q' must be finite. □

In fact, this proposition is a special case of the following one, which is proved likewise.

Proposition 8.5 Let $S_\kappa(p)$ denote the set of all simulation formulas satisfied by p in which all index sets have cardinality less than κ. Let $p, q \in \mathbb{P}$ and assume $|\{q' \mid q \xrightarrow{\sigma} q'\}| < \kappa$ for each $\sigma \in Act^*$. Then $p \overset{\subseteq}{\hookrightarrow} q \Leftrightarrow S_\kappa(p) \subseteq S_\kappa(q)$. □

A less radical way to finitise the testing scenario for simulation semantics is to allow infinitely many copies to be made in any state of the process, but put a finite upperbound on the duration of any observation. Observations can then be modelled with simulation formulas in which the index sets can be arbitrary, but there is a finite upperbound on the nesting of the construction $a\varphi$ of the second clause of Definition 8.1.

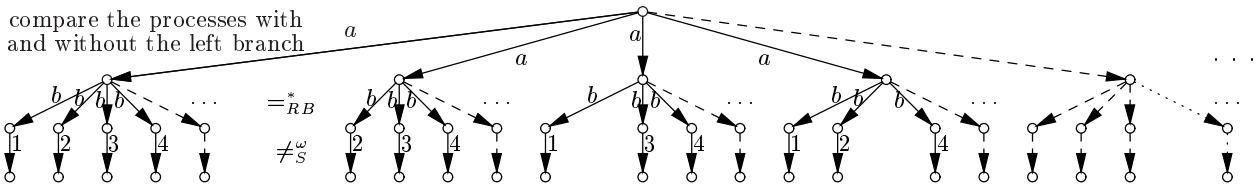
Definition 8.4 Let $\mathcal{L}_S^\omega = \bigcup_{n=0}^\infty \mathcal{L}_S^n$, where $\mathcal{L}_S^n$ is given by:

- If I is a set and $\varphi_i \in \mathcal{L}_S^n$ for $i \in I$ then $\bigwedge_{i\in I} \varphi_i \in \mathcal{L}_S^n$.

- If $\varphi \in \mathcal{L}_S^n$ and $a \in Act$ then $a\varphi \in \mathcal{L}_S^{n+1}$.

Let $S^\omega(p) = \{\varphi \in \mathcal{L}_S^\omega \mid p \models \varphi\}$ and write $p =_S^\omega q$ if $S^\omega(p) = S^\omega(q)$.

Now $p =_S q \Rightarrow p =_S^\omega q \Rightarrow p =_S^* q$, and for image finite processes all three equivalences coincide. For image infinite processes both implications are strict, as illustrated by Counterexamples 8 and 1.



Counterexample 8: Finitary equivalent, but not S^ω-equivalent

In Counterexample 8 $S^*(with) = S^*(without)$, yet $a\bigwedge_{i=1}^\infty bi \in S^\omega(with) - S^\omega(without)$.
In Counterexample 1 $S^\omega(left) = S^\omega(right)$, yet $right \overset{C}{\not\simeq} left$. For the first statement, let $\varphi \in \mathcal{L}_S^\omega$. Then there is an n such that $\varphi \in \mathcal{L}_S^n$. Now parts of trees that are further than n edges away from the root play no rôle in the satisfaction relation for φ. Thus, the validity of φ remains unchanged if in both trees all paths are cut off after n steps. However, the cut versions of both trees are isomorphic, and hence satisfy the same formulas (cf. Corollary 12.1). The second statement follows immediately from Proposition 8.3.

It follows that $T \prec S^* \prec S^\omega \prec S$ and $T \prec T^\infty \prec S$, whereas T^∞ is incomparable with S^* and S^ω. Moreover, S^*, S^ω and S are incomparable with the semantics ranging from CT or F^- to RT^∞.

9 Ready simulation semantics

Testing scenario 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 machines from the Sections 3 to 7 together with a *replicator*, an ingenious device by which one can replicate the machine whenever and as often as one wants. In order to represent observations, the modal languages from Sections 3 to 7 need to be combined with the one from Section 8.

Definition 9 The language $\mathcal{L}_{CS}$ and the corresponding satisfaction relation is defined recursively by combining the clauses of Definition 3.1 (for $\mathcal{L}_{CT}$) with those of Definition 8.1 (for $\mathcal{L}_S$). Likewise, $\mathcal{L}_{FS}$ is obtained by combining $\mathcal{L}_F$ and $\mathcal{L}_S$; $\mathcal{L}_{FTS}$ by combining $\mathcal{L}_{FT}$ and $\mathcal{L}_S$; $\mathcal{L}_{RS}$ by combining $\mathcal{L}_R$ and $\mathcal{L}_S$; and $\mathcal{L}_{RTS}$ by combining $\mathcal{L}_{RT}$ and $\mathcal{L}_S$. For $p \in \mathbb{P}$ and $X \in \{CS, FS, FTS, RS, RTS\}$ let $X(p) = \{\varphi \in \mathcal{L}_X \mid p \models \varphi\}$. Two processes $p, q \in \mathbb{P}$ are

- *completed simulation equivalent*, notation $p =_{CS} q$, if $CS(p) = CS(q)$;
- *failure simulation equivalent*, notation $p =_{FS} q$, if $FS(p) = FS(q)$;
- *failure trace simulation equivalent*, notation $p =_{FTS} q$, if $FTS(p) = FTS(q)$;
- *ready simulation equivalent*, notation $p =_{RS} q$, if $RS(p) = RS(q)$;
- *ready trace simulation equivalent*, notation $p =_{RTS} q$, if $RTS(p) = RTS(q)$.

It is obvious that failure trace simulation equivalence coincides with failure simulation equivalence and ready trace simulation equivalence with ready simulation equivalence ($p \models X\varphi \Leftrightarrow p \models X \wedge \varphi$).

Also it is not difficult to see that failure simulation equivalence and ready simulation equivalence coincide ($p \models X \Leftrightarrow p \models \widetilde{Y} \wedge \bigwedge_{a \in X} a\top$, where $Y = Act - X$). So one has

Proposition 9.1 $p =_{FS} q \Leftrightarrow p =_{FTS} q \Leftrightarrow p =_{RTS} q \Leftrightarrow p =_{RS} q$. □

Relational Characterizations The two remaining equivalences can be characterized as follows:

Definition 9.1 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 $I(p) = \emptyset \Leftrightarrow I(q) = \emptyset$.

Proposition 9.2 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.
Proof: A trivial modification of the proof of Proposition 8.2. □

Definition 9.2 A *ready simulation* is a binary relation R on processes, satisfying, for $a \in Act$:

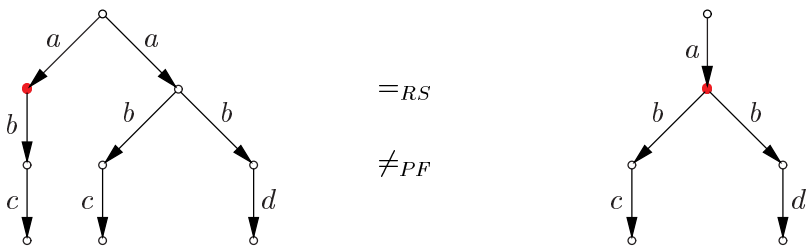
- if pRq and $p \xrightarrow{a} p'$, then $\exists q' :\ q \xrightarrow{a} q'$ and $p'Rq'$;
- if pRq then $I(p) = I(q)$.

Proposition 9.3 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.
Proof: A trivial modification of the proof of Proposition 8.2. □

A variant of ready simulation equivalence was originally proposed by Bloom, Istrail & Meyer [9] under the name *GSOS trace congruence*; they provided a modal characterization, to be discussed in Section 10. A relational characterization was first given by Larsen & Skou [29] under the name $\frac{2}{3}$*-bisimulation equivalence*. A $\frac{2}{3}$-bisimulation is defined just like a ready simulation, except that the second clause reads "if pRq and $\exists q' : q \xrightarrow{a} q'$ then $\exists p' : p \xrightarrow{a} p'$". This is clearly equivalent.

Classification $RT \prec RS$, $CT \prec CS$ and $S \prec CS \prec RS$. CS is independent of F to RT.
Proof: "$RT \preceq RS$" follows since $\mathcal{L}_{RT}$ is a sublanguage of $\mathcal{L}_{RTS}$, using Proposition 9.1.
"$CT \preceq CS$" and "$S \preceq CS \preceq RS$" follow since $\mathcal{L}_{CT}$ and $\mathcal{L}_{S}$ are sublanguages of $\mathcal{L}_{CS}$, which is a sublanguage of $\mathcal{L}_{FS}$.
"$RT \not\succeq RS$" follows from Counterexample 7, using "$RS \succeq S$"; similarly $RT \not\succeq CS$ and $CT \not\succeq CS$.
"$S \not\succeq CS$" follows from Counterexample 2, using "$CS \succeq CT$".
"$CS \not\succeq F$" (and hence "$CS \not\succeq RS$") follows from Counterexample 3, in which $F(left) \neq F(right)$ but $left =_{CS} right$; the construction of the two complete simulations is left to the reader. □

Proposition 9.4 RS and PF are incomparable semantics.
Proof: That "$RS \not\preceq PF$" follows from the counterexample requested in Exercise 6 in Section 7.
"$RS \not\succeq PF$" follows from Counterexample 9, which shows two graphs that are ready simulation equivalent but not possible-futures equivalent. Concerning the first claim, note that there exists exactly one simulation of *right* by *left*, namely the one mapping *right* on the right-hand side of *left*. There also exists exactly one simulation of *left* by *right*, which relates the red node on the *left* to the red node on the *right*. Both simulations are ready simulations, as related nodes have the same menu of initial actions. The second claim follows since $\langle a, \{\epsilon,\ b,\ bc\}\rangle \in PF(left) - PF(right)$. □

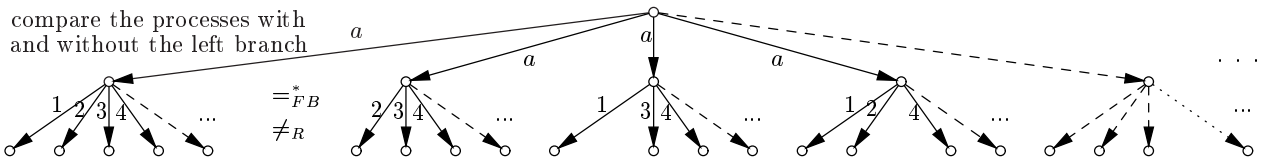


Counterexample 9: Ready simulation equivalent, but not possible-futures equivalent

Infinite processes For each of the semantics *CS, FS, FTS, RS* and *RTS* a finitary variant (superscripted with a *), motivated by allowing finite replication only, is defined by combining the modal languages $\mathcal{L}_{CT}$, $\mathcal{L}_F$, $\mathcal{L}_{FT}$, $\mathcal{L}_R$ and $\mathcal{L}_{RT}$, respectively, with $\mathcal{L}^*_S$. Likewise, an intermediate variant (superscripted with an ω), motivated by requiring any observation to be over within a finite amount of time, is defined by combining these languages with $\mathcal{L}^\omega_S$. Finally, a finite variant (superscripted with a $-$), motivated by observers that can only engage in finite replication, can only set finitely many switches on free, and can only inspect finitely many lamps in a finite time, is obtained by combining the (obvious) modal languages $\mathcal{L}^-_F$, $\mathcal{L}^-_{FT}$, $\mathcal{L}^-_R$ and $\mathcal{L}^-_{RT}$ with $\mathcal{L}^*_S$ (there is no CS^-). Exactly as in the case of Proposition 9.1 one finds:

Proposition 9.5 $FS^\omega = FTS^\omega = RTS^\omega = RS^\omega$ and $FS^- = FTS^- = RTS^- = RS^-$. Moreover, $FS^* = FTS^*$ and $RTS^* = RS^*$. □

However, as pointed out in Schnoebelen [41], FS^* and RS^* are different: in Counterexample 10



Counterexample 10: Finitary failure simulation equivalent, but not ready equivalent

one has $FS^*(with) = FS^*(without)$, but $\langle a, \{1, 2, ...\}\rangle \in R(with) - R(without)$.

Clearly one has $CS^* \prec CS^\omega \prec CS$ and $RS^- \prec FS^* \prec RS^* \prec RS^\omega \prec RS$. The strictness of these inclusions is given by Counterexamples 4, 10, 8 and 1. In addition one has $RT^- \prec RS^-$, $S^* \prec RS^-$, $RT \prec RS^*$, $FT \prec FS^*$, $CT \prec CS^*$ and $S^* \prec CS^* \prec FS^*$; as well as $RT^\infty \prec RS$, $CT^\infty \prec CS$, $S^\omega \prec CS^\omega \prec RS^\omega$ and $S \prec CS \prec RS$. Counterexamples against further inclusions have already been provided.

Proposition 9.6 Let $p, q \in \mathbb{P}$ be image finite. Then $p =^*_{CS} q \Leftrightarrow p =_{CS} q$ and $p =^-_{RS} q \Leftrightarrow p =_{RS} q$.
Proof: Two trivial modifications of the proof of Proposition 8.4. In the second one, one uses that if $\forall \varphi \in \mathcal{L}^-_{RS}(p \models \varphi \Rightarrow q \models \varphi)$ then surely $I(p) = I(q)$. □

For general (non-image-finite) processes, no relational characterisations of the finite, finitary and intermediate equivalences are known.

Testing scenario An alternative and maybe more natural testing scenario for finitary ready simulation semantics (or simulation semantics) can be obtained by exchanging the replicator for an *undo*-button on the (ready) trace machine (Figure 6). It is assumed that all intermediate states

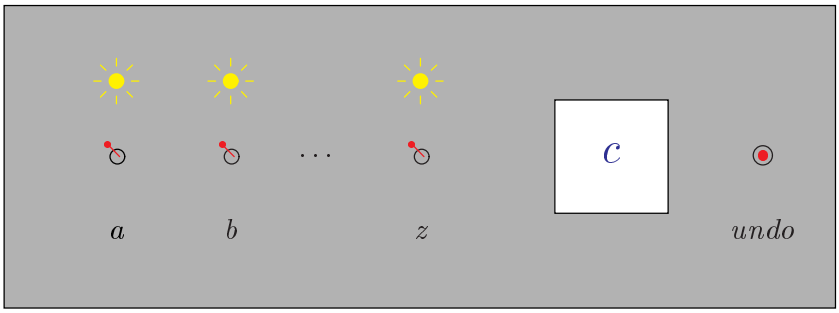


Figure 6: The ready simulation machine

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 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.

10 Reactive versus generative testing scenarios

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. Initially the process waits patiently until the observer

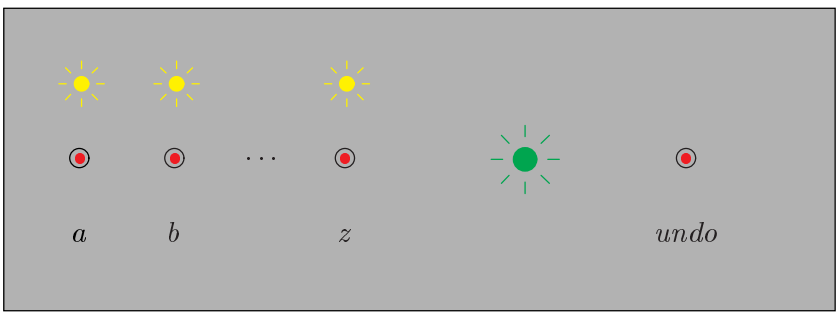


Figure 7: The reactive ready simulation machine

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 [31, 32].

One 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 preceding action (when no button can go down). When this preceding 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'; moreover an action a appearing in the display is mimicked by the a-button going down, and the disappearance of a from the display by the green light going off.

Another family of testing scenarios is obtained by allowing the user to try only one button at a time. They are equipotent with those generative testing scenarios in which at any time only one switch can be set on 'free'. Next I will discuss the equivalences that originate from these scenarios.

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, indicating which buttons went down in a sequence of trials of the user. Such a sequence can be seen as a failure trace where all refusal sets are singletons. Call the resulting semantics FT^1. Clearly, the failure trace set of any process p satisfies

$$\sigma(X \cup Y)\rho \in FT(p) \;\Leftrightarrow\; \sigma XY\rho \in FT(p).$$

Thus, any failure trace $\sigma\{a_1,\ldots,a_n\}\rho$ can be rewritten as (contains the same information as) $\sigma\{a_1\}\{a_2\}\cdots\{a_n\}\rho$. It follows that the singleton-failure trace set $FT^1(p)$ of a process p contains as much information as its finite-failure trace set $FT^-(p)$, so the semantics FT^1 coincides with FT^-.

In order to arrive at a reactive counterpart to failures semantics, one could suppose that an observer continues an experiment only as long as all buttons he tries to depress actually go down; when a button refuses to go down, he will not try another one. This testing scenario gives rise to the variant F^1 of failures semantics in which all refusal sets are singletons. Contrary to the situation for failure trace semantics, F^1 is situated strictly between T and F^- As the testing scenario is not very convincing, and F^1 did not appear in the literature before, I will not treat it formally here.

Adding the *undo*-button to the reactive failure trace machine gives a semantics FS^1 characterized by the modal language $\mathcal{L}^*_S$ to which has been added a modality "Can't(a)", with $p \models$ Can't(a) iff $a \notin I(p)$. This modality denotes a failed attempt to depress the a-button. If fact, BLOOM, ISTRAIL & MEYER studied the coarsest equivalence finer than trace equivalence that is a congruence for the class of so-called *GSOS-operators*, and characterized this *GSOS trace congruence* by the modal language above; its formulas were called *denial formulas*. As in the modal language $\mathcal{L}^-_{FS}$ one has $p \models X\widetilde{\cup}Y \Leftrightarrow p \models \widetilde{X} \wedge \widetilde{Y}$, and Can't($a$) is the same as $\widetilde{\{a\}}$, it follows that the language of denial formulas is equally expressive as $\mathcal{L}^-_{FS}$, and hence FS^1 coincides with FS^- and RS^-.

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. Likewise, adding menu-lights to the reactive failure scenario would give readiness semantics, and adding them to the reactive failure simulation machine would yield ready simulation. If the green light (as well as the menu-lights) are removed from the reactive failure trace machine, one can only test trace equivalence, since any refusal may be caused by the last action not being ready yet. Likewise, removing the green

light from the reactive failure simulation machine (with *undo*-button) yields (finitary) simulation semantics. Reactive machines seem to be unsuited for testing completed trace, completed simulation and failures equivalence.

11 2-nested simulation semantics

2-nested simulation equivalence popped up naturally in Groote & Vaandrager [21] as the coarsest congruence with respect to a large and general class of operators that is finer than completed trace equivalence.

Definition 11 A *2-nested simulation* is a simulation contained in simulation equivalence ($\rightleftarrows$). Two processes p and q are *2-nested simulation equivalent*, notation $p =_{2S} q$, if there exists a 2-nested simulation R with pRq and a 2-nested simulation S with qSp.

Modal Characterization A modal characterization of this notion is obtained by the fragment of the infinitary Hennessy-Milner logic (cf. Definition 12.1) without nested negations.

Definition 11.1 The class $\mathcal{L}_{2S}$ of *2-nested simulation formulas* over Act is defined recursively by:

- If I is a set and $\varphi_i \in \mathcal{L}_{2S}$ for $i \in I$ then $\bigwedge_{i\in I} \varphi_i \in \mathcal{L}_{2S}$.
- If $\varphi \in \mathcal{L}_{2S}$ and $a \in Act$ then $a\varphi \in \mathcal{L}_{2S}$.
- If $\varphi \in \mathcal{L}_S$ then $\neg\varphi \in \mathcal{L}_{2S}$.

Note that $\mathcal{L}_S \subseteq \mathcal{L}_{2S}$. The *satisfaction relation* $\models \subseteq \mathbb{P} \times \mathcal{L}_{2S}$ is defined recursively by:

- $p \models \bigwedge_{i\in I} \varphi_i$ if $p \models \varphi_i$ for all $i \in I$.
- $p \models a\varphi$ if for some $q \in \mathbb{P}$: $p \xrightarrow{a} q$ and $q \models \varphi$.
- $p \models \neg\varphi$ if $p \not\models \varphi$.

Proposition 11.1 $p =_{2S} q \;\Leftrightarrow\; \forall \varphi \in \mathcal{L}_{2S}(p \models \varphi \Leftrightarrow q \models \varphi)$.
Proof: A trivial modification of the proof of Proposition 8.2. □

Testing scenario In order to obtain a testing scenario for this equivalence one has to introduce the rather unnatural notion of a *lookahead* [21]: The *2-nested simulation machine* is a variant of the ready trace machine with replicator, where in an idle state the machine not only tells which actions are on the menu, but even which simulation formulas are (not) satisfied in the current state.

Classification $RS \prec 2S$ and $PF \prec 2S$.
Proof: For "$RS \preceq 2S$" it suffices to show that each 2-nested simulation is a ready simulation. This follows since $p \rightleftarrows q \Rightarrow I(p) = I(q)$. $PF \prec 2S$ is easily established using that $T \prec S$. That both inclusions are strict follows immediately from the fact that RS and PF are incomparable (Proposition 9.4). □

Infinite processes Exactly as for ready simulations semantics, 5 versions of 2-nested simulation semantics can be defined that differ for infinite processes. $2S^-$ is the semantics whose modal characterization has the constructs $\top$, $\wedge$, $a\varphi$ and $\neg\varphi'$ with $\varphi' \in \mathcal{L}^*_S$. The constructs $\widetilde{X}$ and X for $X \subseteq_{fin} Act$ are expressible in this logic. $F2S^*$ additionally has the construct $\widetilde{X}$, and $R2S^*$ the construct X, for $X \subseteq Act$. Finally $2S^\omega$ is characterized by the class of 2-nested simulation formulas with a finite upperbound on the nesting of the $a\varphi$ construct. The constructs $\widetilde{X}$ and X for $X \subseteq Act$ are expressible in $\mathcal{L}^\omega_{2S}$, and hence also in $\mathcal{L}_{2S}$.

We have $2S^- \prec F2S^* \prec R2S^* \prec 2S^\omega \prec 2S$. The strictness of these inclusions is given by Counterexamples 4, 10, 8 and 1. In addition one has $RS^- \prec 2S^-$, $FS^* \prec F2S^*$, $RS^* \prec R2S^*$, $RS^\omega \prec 2S^\omega$ and $RS \prec 2S$; as well as $PF^\infty \prec 2S$. Counterexample 1 shows that $PF \not\prec 2S^*$: $2S^*(left) = 2S^*(right)$ (cf. Proposition 12.11), but $\langle a, a^*\rangle \in PF(right) - PF(left)$.

Proposition 11.2 Let $p, q \in \mathbb{P}$ be image finite. Then $p =^-_{2S} q \Leftrightarrow p =_{2S} q$.
Proof: An easy modifications of the proof of Proposition 8.4, also using the result of Proposition 8.4. □

12 Bisimulation semantics

The concept of *bisimulation equivalence* stems from Milner [31]. It's formulation below is due to Park [35].

Definition 12 A *bisimulation* is a binary relation R on processes, satisfying, for $a \in Act$:

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

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

The relation $\leftrightarrows$ is again a bisimulation. As for similarity, one easily checks that bisimilarity is an equivalence relation on $\mathbb{P}$. Hence the relation will be called *bisimulation equivalence*. In *bisimulation semantics* (B) two processes are identified iff they are bisimulation equivalent. 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 $\xrightarrow{\sigma}$.

Modal Characterization

Definition 12.1 The class $\mathcal{L}_B$ of *infinitary Hennessy-Milner formulas* over Act is defined by:

- If I is a set and $\varphi_i \in \mathcal{L}_B$ for $i \in I$ then $\bigwedge_{i\in I} \varphi_i \in \mathcal{L}_B$.
- If $\varphi \in \mathcal{L}_B$ and $a \in Act$ then $a\varphi \in \mathcal{L}_B$.
- If $\varphi \in \mathcal{L}_B$ then $\neg\varphi \in \mathcal{L}_B$.

The *satisfaction relation* $\models \subseteq \mathbb{P} \times \mathcal{L}_B$ is defined recursively by:

- $p \models \bigwedge_{i\in I} \varphi_i$ if $p \models \varphi_i$ for all $i \in I$.
- $p \models a\varphi$ if for some $q \in \mathbb{P}$: $p \xrightarrow{a} q$ and $q \models \varphi$.
- $p \models \neg\varphi$ if $p \not\models \varphi$.

Let $B(p)$ denote the class of all infinitary Hennessy-Milner formulas satisfied by the process p: $B(p) = \{\varphi \in \mathcal{L}_B \mid p \models \varphi\}$. Write $p \sqsubseteq_B q$ if $B(p) \subseteq B(q)$ and $p =_B q$ if $B(p) = B(q)$.

Proposition 12.1 $p \sqsubseteq_B q \Leftrightarrow p =_B q$.
Proof: If $\varphi \in B(q) - B(p)$ then $\neg\varphi \in B(p) - B(q)$. □

Proposition 12.2 $p \leftrightarrows q \Leftrightarrow p =_B q$.
Proof: For "⇒" I have to prove that for any bisimulation R and for all $\varphi \in \mathcal{L}_B$ one has

$$pRq \Rightarrow (p \models \varphi \Leftrightarrow q \models \varphi).$$

I will do so with structural induction on φ. Suppose pRq.

- Let $p \models a\varphi$. Then there is a $p' \in \mathbb{P}$ with $p \xrightarrow{a} p'$ and $p' \models \varphi$. As R is a bisimulation, there must be a $q' \in \mathbb{P}$ with $q \xrightarrow{a} q'$ and $p'Rq'$. So by induction $q' \models \varphi$, and hence $q \models a\varphi$.
  By symmetry one also obtains $q \models a\varphi \Rightarrow p \models a\varphi$.
- $p \models \bigwedge_{i\in I} \varphi_i \Leftrightarrow \forall i \in I (p \models \varphi_i) \overset{\text{ind.}}{\Longleftrightarrow} \forall i \in I (q \models \varphi_i) \Leftrightarrow q \models \bigwedge_{i\in I} \varphi_i$.
- $p \models \neg\varphi \Leftrightarrow p \not\models \varphi \overset{\text{ind.}}{\Longleftrightarrow} q \not\models \varphi \Leftrightarrow q \models \neg\varphi$.

For "⇐" it suffices to establish that $\sqsubseteq_B$ is a simulation (Proposition 12.1 then implies that $=_B = \sqsubseteq_B = \sqsubseteq_B^{-1}$ is a bisimulation). This goes exactly as in the proof of Proposition 8.2. □

Testing scenario The testing scenario for bisimulation semantics, as presented in Milner [31], 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 replicator, 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 [31] implemented global testing by assuming that

"(i) It is the *weather* at any moment which determines the choice of transition (in case of ambiguity [...]);

(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 a given a-experiment will be investigated by simply performing the experiment in all possible weather conditions. Unfortunately, as remarked in Milner [32], 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 dropped, thereby losing the possibility of effective implementation of the testing scenario.

An observation in the global testing scenario can be represented as an infinitary Hennessy-Milner formula $\varphi \in \mathcal{L}_B$. This is essentially a simulation formula in which it is possible to indicate that certain branches are not present. A formula $\neg\varphi$ says that by making sufficiently many copies of the investigated process, and exposing them to all possible whether conditions, it can be observed that none of these copies permits the observation φ.

Remark: Let $[a]\varphi$ denote $\neg a \neg \varphi$. Now the negation in $\mathcal{L}_B$ can be eliminated in favour of the

modalities $[a]$ and infinitary disjunction $\bigvee_{i\in I}$. A formula $[a]\varphi$ says that in all possible weather conditions, after an a-move it is always possible to make the observation φ.

In order to justify the observations of $\mathcal{L}_B$ in a generative testing scenario no switches or menu-lights are needed; the architecture of the completed trace machine suffices. However, in order to warrant negative observations, one has to assume that actions take only a finite amount of time, and idling can be detected (either by observations that last forever, or by means of the display becoming empty). Adding switches and or menu-lights does not increase the discriminating power of the observers. It would give rise to observations that can be modelled as formulas in languages $\mathcal{L}_{FB}$, $\mathcal{L}_{RTB}$, etc., obtained by combining $\mathcal{L}_F$, $\mathcal{L}_{RT}$, etc. with $\mathcal{L}_B$. These observations can already be expressed in $\mathcal{L}_B$: $p \models \tilde{X} \Leftrightarrow p \models \bigwedge_{a\in X} \neg a\top$ and $p \models X\varphi \Leftrightarrow p \models (\bigwedge_{a\notin X} \neg a\top) \wedge (\bigwedge_{a\in X} a\top) \wedge \varphi$.

A different implementation of global testing is given in Larsen & Skou [29]. 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 [9], 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.

Process Graph Characterization Also bisimulation equivalence can be characterized by means of relations between the nodes of two process graphs.

Definition 12.2 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:

- $\text{ROOT}(g) R \text{ROOT}(h)$.
- If sRt and $(s, a, s') \in \text{EDGES}(g)$, then there is an edge $(t, a, t') \in \text{EDGES}(h)$ such that $s'Rt'$.
- 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. Solid lines indicates what is assumed, dashed lines what is required. It follows easily that $g \leftrightarroweq h$ iff there exists a bisimulation between g and h.

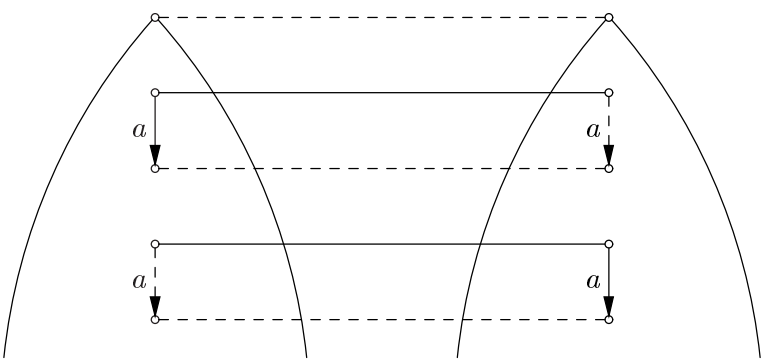
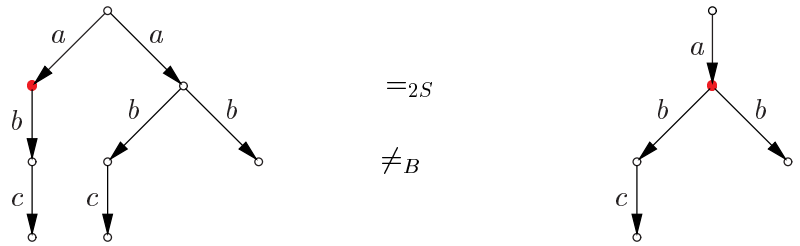


Figure 8: A bisimulation

Classification $2S \prec B$.
Proof: "$2S \preceq B$" follows since $\mathcal{L}_{2S}$ is a sublanguage of $\mathcal{L}_B$.
"$2S \not\succeq B$" follows from Counterexample 11, which shows two graphs that are 2-nested simulation equivalent, but not bisimulation equivalent. Concerning the first claim, as in Counterexample 9 there exists exactly one simulation of *left* by *right*, which relates the red node on the *left* to the red node on the *right*. Unlike in Counterexample 9, this simulation is 2-nested, for the two subgraphs originating from the two red nodes are simulation equivalent, as are the graphs *left* and *right* themselves. Likewise, the simulation mapping *right* on the right-hand side of *left* is also 2-nested. The second claim follows since $a\neg b\neg cT \in B(left) - B(right)$. □



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

Counterexample 11: 2-nested simulation equivalent, but not bisimulation equivalent

Thus bisimulation equivalence is the finest semantic equivalence treated so far. The following shows however that on $\mathbb{G}$ graph isomorphism is even finer, i.e. isomorphic graphs are always bisimilar. In fact, a graph isomorphism can be seen as a bijective bisimulation. That not all bisimilar graphs are isomorphic is trivial and left to the reader.

Proposition 12.3 For $g, h \in \mathbb{G}$, $g \cong h$ iff there exists a bisimulation R between g and h, satisfying

- If sRt and uRv then $s = u \Leftrightarrow t = v$. (*)

Proof: Suppose $g \cong h$. Let $f : \textsc{nodes}(g) \to \textsc{nodes}(h)$ be a graph isomorphism. Define $R \subseteq \textsc{nodes}(g) \times \textsc{nodes}(h)$ by sRt iff $f(s) = t$. Then it is routine to check that R satisfies all clauses of Definition 12.2 and (*). Now suppose R is a bisimulation between g and h satisfying (*). Define $f : \textsc{nodes}(g) \to \textsc{nodes}(h)$ 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 (*) implies that $f(s)$ is uniquely determined. Hence f is well-defined. Now direction "$\Leftarrow$" of (*) implies that f is injective. From the connectedness of h if follows that f is also surjective, and hence a bijection. Finally, the clauses of Definition 12.2 imply that f is a graph isomorphism. □

Corollary 12.1 If $g \cong h$ then g and h are equivalent according to all semantic equivalences encountered so far.

Non-well-founded sets 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 $x_0 \notin x_1 \notin \cdots$ terminates) is replaced by an *anti-foundation axiom*.

Definition 12.3 Let $\mathcal{B}$ denote the unique function $\mathcal{M} : \mathbb{P} \to \mathcal{V}$ satisfying

$$\mathcal{M}(p) = \{\langle a, \mathcal{M}(q)\rangle \mid p \xrightarrow{a} q\}$$

for all $p \in \mathbb{P}$. Two processes p and q are *branching equivalent* (my terminology) if $\mathcal{B}(p) = \mathcal{B}(q)$.

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 $\mathcal{B}$ 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 [18], where also a proof can be found of the next proposition, saying that bisimulation equivalence coincides with branching equivalence.

Proposition 12.4 Let $p, q \in \mathbb{P}$. Then $p \leftrightarroweq q \;\Leftrightarrow\; \mathcal{B}(p) = \mathcal{B}(q)$.
Proof: "$\Leftarrow$": Let B be the relation defined by pBq iff $\mathcal{B}(p) = \mathcal{B}(q)$; then it suffices to prove that B is a bisimulation. Suppose pBq and $p \xrightarrow{a} p'$. Then $\langle a, \mathcal{B}(p')\rangle \in \mathcal{B}(p) = \mathcal{B}(q)$. So by the definition of $\mathcal{B}(q)$ there must be a process q' with $\mathcal{B}(p') = \mathcal{B}(q')$ and $q \xrightarrow{a} q'$. Hence $p'Bq'$, which had to be proved. The second requirement for B being a bisimulation follows by symmetry.

"$\Rightarrow$": Let $\mathcal{B}^*$ denote the unique solution of $\mathcal{M}^*(p) = \{\langle a, \mathcal{M}^*(r')\rangle \mid \exists r : r \leftrightarroweq p \wedge r \xrightarrow{a} r'\}$. As for $\mathcal{B}$ it follows from the anti-foundation axiom that such a unique solution exists. From the symmetry and transitivity of $\leftrightarroweq$ it follows that

$$p \leftrightarroweq q \quad\Rightarrow\quad \mathcal{B}^*(p) = \mathcal{B}^*(q). \tag{1}$$

Hence it remains to be proven that $\mathcal{B}^* = \mathcal{B}$. This can be done by showing that $\mathcal{B}^*$ satisfies the equations $\mathcal{M}(p) = \{\langle a, \mathcal{M}(q)\rangle \mid p \xrightarrow{a} q\}$, which have $\mathcal{B}$ as unique solution. So it has to be established that $\mathcal{B}^*(p) = \{\langle a, \mathcal{B}^*(q)\rangle \mid p \xrightarrow{a} q\}$. The direction "$\supseteq$" follows directly from the reflexivity of $\leftrightarroweq$. For "$\subseteq$", suppose $\langle a, X\rangle \in \mathcal{B}^*(p)$. Then $\exists r : r \leftrightarroweq p$, $r \xrightarrow{a} r'$ and $X = \mathcal{B}^*(r')$. Since $\leftrightarroweq$ is a bisimulation, $\exists p' : p \xrightarrow{a} p'$ and $r' \leftrightarroweq p'$. From (1) it follows that $X = \mathcal{B}^*(r') = \mathcal{B}^*(p')$. Therefore $\langle a, X\rangle \in \{\langle a, \mathcal{B}^*(q)\rangle \mid p \xrightarrow{a} q\}$, which had to be established. □

Infinite processes The finite version of bisimulation equivalence is described by the *Hennessy-Milner logic* (*HML*) [23, 24].

Definition 12.4 The set $\mathcal{L}_{\rm HM}$ of *Hennessy-Milner formulas* over *Act* is defined recursively by:

- $\top \in \mathcal{L}_{\rm HM}$.
- If $\varphi, \psi \in \mathcal{L}_{\rm HM}$ then $\varphi \wedge \psi \in \mathcal{L}_{\rm HM}$.
- If $\varphi \in \mathcal{L}_{\rm HM}$ and $a \in Act$ then $a\varphi \in \mathcal{L}_{\rm HM}$.
- If $\varphi \in \mathcal{L}_{\rm HM}$ then $\neg\varphi \in \mathcal{L}_{\rm HM}$.

The *satisfaction relation* $\models \subseteq \mathbb{P} \times \mathcal{L}_{\rm HM}$ is defined recursively by:

- $p \models \top$ for all $p \in \mathbb{P}$.
- $p \models \varphi \wedge \psi$ if $p \models \varphi$ and $p \models \psi$.
- $p \models a\varphi$ if for some $q \in \mathbb{P}$: $p \xrightarrow{a} q$ and $q \models \varphi$.
- $p \models \neg\varphi$ if $p \not\models \varphi$.

Let $HM(p)$ denote the set of all Hennessy-Milner formulas that are satisfied by the process p: $HM(p) = \{\varphi \in \mathcal{L}_{\rm HM} \mid p \models \varphi\}$. Two processes p and q are *HML-equivalent*, notation $p =_B^- q$, if $HM(p) = HM(q)$.

For image finite processes Hennessy & Milner [23, 24] provided the following characterizations of this equivalence.

Definition 12.5 Let $p, q \in \mathbb{P}$. Then:

- $p \approx_0 q$ is always true.
- $p \approx_{n+1} q$ if for all $a \in Act$:
  - $p \xrightarrow{a} p'$ implies $\exists q' :\ q \xrightarrow{a} q'$ and $p' \approx_n q'$;
  - $q \xrightarrow{a} q'$ implies $\exists p' :\ p \xrightarrow{a} p'$ and $p' \approx_n q'$.
- p and q are *observationally equivalent*, notation $p \approx q$, if $p \approx_n q$ for every $n \in \mathbb{N}$ [23, 24].

Proposition 12.5 Let $p, q \in \mathbb{P}$ be image finite. Then $p \approx q \Leftrightarrow HML(p) = HML(q)$.
Proof: This is Theorem 2.2 in Hennessy & Milner [23, 24]. □

Proposition 12.6 Let $p, q \in \mathbb{P}$ be image finite. Then $p \leftrightarrows q \ \Leftrightarrow\ p \approx q$.
Proof: This follows immediately from Theorem 2.1 in Hennessy & Milner [23, 24]. □

Inspired by this insight, Park [35] proposed the concise formulation of bisimulation equivalence employed in Definition 12. Below is a modal characterization of $\approx$.

Definition 12.6 Let $\mathcal{L}_B^\omega = \bigcup_{n=0}^\infty \mathcal{L}_B^n$, where $\mathcal{L}_B^n$ is given by:

- If I is a set and $\varphi_i \in \mathcal{L}_B^n$ for $i \in I$ then $\bigwedge_{i\in I} \varphi_i \in \mathcal{L}_B^n$.
- If $\varphi \in \mathcal{L}_B^n$ and $a \in Act$ then $a\varphi \in \mathcal{L}_B^{n+1}$.
- If $\varphi \in \mathcal{L}_B^n$ then $\neg\varphi \in \mathcal{L}_B^n$.

Let $B^\omega(p) = \{\varphi \in \mathcal{L}_B^\omega \mid p \models \varphi\}$ and write $p =_B^\omega q$ if $B^\omega(p) = B^\omega(q)$.

Proposition 12.7 $p \approx_n q \Leftrightarrow \forall \varphi \in \mathcal{L}_B^n (p \models \varphi \Leftrightarrow q \models \varphi)$ for all $n \in \mathbb{N}$. Hence $p \approx q \Leftrightarrow p =_B^\omega q$.
Proof: *Induction Base:* Formulas in $\mathcal{L}_B^0$ do not contain the construct $a\varphi$. Hence for such formulas ψ the statement $p \models \psi$ is independent of p. Thus $\forall p, q \in \mathbb{P} : \forall \varphi \in \mathcal{L}_B^0 (p \models \varphi \Leftrightarrow q \models \varphi)$.
Induction Step: Suppose $p \approx_{n+1} q$. I now use structural induction on φ.

- Let $p \models a\varphi$ with $a\varphi \in \mathcal{L}_B^{n+1}$. Then there is a $p' \in \mathbb{P}$ with $p \xrightarrow{a} p'$ and $p' \models \varphi \in \mathcal{L}_B^n$. As $p \approx_{n+1} q$, there must be a $q' \in \mathbb{P}$ with $q \xrightarrow{a} q'$ and $p' \approx_n q'$. So by induction $q' \models \varphi$, and hence $q \models a\varphi$.
  By symmetry one also obtains $q \models a\varphi \Rightarrow p \models a\varphi$.
- $p \models \bigwedge_{i\in I} \varphi_i \Leftrightarrow \forall i \in I (p \models \varphi_i) \overset{\text{ind.}}{\Longleftrightarrow} \forall i \in I (q \models \varphi_i) \Leftrightarrow q \models \bigwedge_{i\in I} \varphi_i$.

– $p \models \neg\varphi \Leftrightarrow p \not\models \varphi \stackrel{\text{ind.}}{\Longleftrightarrow} q \not\models \varphi \Leftrightarrow q \models \neg\varphi$.

Now suppose $\forall \varphi \in \mathcal{L}_B^{n+1}(p \models \varphi \Leftrightarrow q \models \varphi)$ and $p \stackrel{a}{\longrightarrow} p'$. Considering the symmetry in the definitions involved, all I have to show is that $\exists q' \in \mathbb{P}$ with $q \stackrel{a}{\longrightarrow} q'$ and $p' \approx_n q'$. Let Q' be

$$\{q' \in \mathbb{P} \mid q \stackrel{a}{\longrightarrow} q' \wedge p' \not\approx_n q'\}.$$

By Definition 1.1 Q' is a set. For every $q' \in Q'$ there must, by induction, be a formula $\varphi_{q'} \in \mathcal{L}_B^n$ with $p' \models \varphi_{q'}$ but $q' \not\models \varphi_{q'}$ (use negation if necessary). Now $p \models a\bigwedge_{q' \in Q'} \varphi_{q'} \in \mathcal{L}_B^{n+1}$ and therefore $q \models a\bigwedge_{q' \in Q'} \varphi_{q'}$. So there must be a $q' \in \mathbb{P}$ with $q \stackrel{a}{\longrightarrow} q'$ and $q' \notin Q'$, which had to be shown. □

For the sake of completeness, two more variants of bisimulation equivalence can be considered. Let FB^* be characterized by the Hennessy-Milner logic augmented with formulas $\widetilde{X}$, and RB^* by the Hennessy-Milner logic augmented with formulas X, for $X \subseteq Act$.

Then $B^- \prec FB^* \prec RB^* \prec B^\omega \prec B$, and for image finite processes all five equivalences coincide. The strictness of these inclusions is given by Counterexamples 4, 10, 8 and 1:

Proposition 12.8 $CT \not\preceq B^-$, and hence $FB^* \not\preceq B^-$.
Proof: Counterexample 4 shows two process with $CT(left) \neq CT(right)$. It remains to be shown that $HM(left) = HM(right)$, i.e. that for all $\varphi \in \mathcal{L}_{\rm HM}$: $left \models \varphi \Leftrightarrow right \models \varphi$. Using Definition 12.4 it is sufficient to restrict attention to formulas φ which are of the form $a(\bigwedge_{i\in I} b_i\top \wedge \bigwedge_{j\in J} \neg b_j\top)$ with I and J finite sets of indices. It is not difficult to see that each such formula that is satisfied on one side is also satisfied on the other side. □

Proposition 12.9 $R \not\preceq FB^*$, and hence $RB^* \not\preceq FB^*$.
Proof: Counterexample 10, shows two process with $R(with) \neq R(without)$. It remains to be shown that $FB^*(with) = FB^*(without)$. The argument is the same as in the previous proof, but this time focusing on formulas of the form $a(\widetilde{X} \wedge \bigwedge_{i\in I} i\top \wedge \bigwedge_{j\in J} \neg j\top)$ with I and J finite sets of numbers and X a possibly infinite set of numbers (= actions). □

Proposition 12.10 $S^\omega \not\preceq RB^*$, and hence $RB^\omega \not\preceq RB^*$.
Proof: Counterexample 8 shows two process with $S^\omega(with) \neq S^\omega(without)$. It remains to be shown that $RB^*(with) = RB^*(without)$. The argument is the same as in the previous proofs—this time using formulas of the form $a(\{b\} \wedge \bigwedge_{i\in I} bi\top \wedge \bigwedge_{j\in J} \neg bj\top)$ with I and J finite sets of numbers. □

Proposition 12.11 $T^\infty \not\preceq B^\omega$, and hence $B \not\preceq B^\omega$. In addition, $PF \not\preceq B^\omega$.
Proof: Counterexample 1 shows two process with $T^\infty(left) \neq T^\infty(right)$. As remarked at the end of Section 7, also $PF(left) \neq PF(right)$. It remains to be shown that $left =_B^\omega right$, i.e. that for all $n \in \mathbb{N}$: $left \approx_n right$. In order to establish $p \approx_n q$ for two trees p and q, the parts of p and q that are further than n edges away from the root play no rôle, and can just as well be omitted. As the cut versions of *left* and *right* are isomorphic, by Corollary 12.1 surely $left \approx_n right$. □

In addition one has $2S^- \prec B^-$, $F2S^* \prec FB^*$, $R2S^* \prec RB^*$, $2S^\omega \prec B^\omega$ and $2S \prec B$.

13 Summary

In Sections 2–12 twelve semantics were defined that are different for finitely branching processes. These are abbreviated by *T*, *CT*, *F*, *R*, *FT*, *RT*, *S*, *CS*, *RS*, *PF*, *2S* and *B*. Write $\mathcal{S} \preceq \mathcal{T}$ if semantics $\mathcal{S}$ makes at least as much identifications as semantics $\mathcal{T}$. This is the case if $=_{\mathcal{S}} \supseteq =_{\mathcal{T}}$.

Theorem 1 $T \preceq CT \preceq F \preceq R \preceq RT$, $F \preceq FT \preceq RT \preceq RS \preceq 2S \preceq B$, $T \preceq S \preceq CS \preceq RS$, $CT \preceq CS$ and $R \preceq PF \preceq 2S$.

Theorem 1 is illustrated in Figure 1. There, however, completed simulation semantics is missing, since it did not occur in the literature. The theorem applies to any labelled transition system $(\mathbb{P}, \rightarrow)$. It has been proven in the various subsections entitled "classification". Whether the inclusions are strict depends on the choice of $(\mathbb{P}, \rightarrow)$. In the same subsections (and through Exercise 6) a number of counterexamples have been presented, showing that on $\mathbb{G}$ all semantic notions mentioned in Theorem 1 are different and $\mathcal{S} \preceq \mathcal{T}$ holds only if this follows from that theorem. Moreover, all relevant examples use finite processes only.

Let $\mathbb{H}$ be the set of finite connected process graphs. Here a process graph $g \in \mathbb{G}$ is *finite* iff PATHS(g) is finite. Finite graphs are acyclic and have only finitely many nodes and edges. They represent finite processes. Now the next theorem follows.

Theorem 2 Let $\mathcal{S}$ and $\mathcal{T}$ be semantics on $\mathbb{H}$ from the series *T, CT, F, R, FT, RT, S, CS, RS, PF, 2S, B*. Then $\mathcal{S} \preceq \mathcal{T}$ only if this follows from Theorem 1 (and the fact that $\preceq$ is a partial order).

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 are notationally distinguished by means of superscripts "*" and "∞" respectively; the unsubscripted abbreviation will, for historical reasons, refer to the infinitary versions in case of simulation-like semantics (treated in Sections 8–12) and to the finitary versions for the trace-like semantics (treated in Section 2–7). For the simulation-like semantics also an intermediate variant is considered—superscripted with "ω"—based on the assumption that observers can investigate arbitrary many copies of a process in parallel, but have only a finite amount of time to do so. For the semantics that incorporate refusal sets, the finitary versions come in two variants, depending on whether the refusal sets are required to be finite (superscript "$-$") or not (the default assumption). A similar distinction is made for semantics where menus of actions can be observed. Finally, finitary failure simulation semantics (FS^*) can be distinguished from finitary ready simulation semantics (RS^*), although FS^ω and RS^ω, as well as FS and RS, and FS^- and RS^-, coincide. A similar situation occurs for 2-nested simulation and for bisimulation.

All semantics encountered are displayed in Figure 9, in which the $\preceq$-relation is represented by solid and dotted arrows.

Theorem 3 For all semantics $\mathcal{S}$ and $\mathcal{T}$ defined so far, the formula $\mathcal{S} \preceq \mathcal{T}$ holds iff there is a path $\mathcal{S} \rightarrow \cdots \rightarrow \mathcal{T}$ (consisting of solid and dotted arrows alike) in Figure 9. Furthermore, semantics connected by dotted arrows coincide for image finite processes.

Proof: All the implications corresponding to the arrows in Figure 9 have been encountered before. The latter statement has been established in Propositions 2.3, 4.5, 5.3, 6.4, 7.3, 7.4, 7.5, 8.4, 9.6, 11.2, 12.5 and 12.6 (except that the case of possible-futures semantics is left to the reader). In order to show that on $\mathbb{G}$ there are no inclusions that are not indicated in Figure 9, is suffices, in view of Theorem 2, the already established parts of Theorem 3, and the fact that $\preceq$ is a partial order, to show that $CT \not\preceq B^-$, $R \not\preceq FB^*$, $S^\omega \not\preceq RB^*$, $T^\infty \not\preceq B^\omega$, $PF \not\preceq B^\omega$ and $T^\infty \not\preceq PF$. This has been done in Propositions 12.8, 12.9, 12.10 and 12.11, and at the end of Section 7. □

One could say that a semantics $\mathcal{S}$ *respects deadlock behaviour* iff $\mathcal{S} \succeq CT$. Counterexample 4 then shows that none of the semantics on the left in Figure 9 respects deadlock behaviour; only the left-hand process of Counterexample 4 can deadlock after an a-move.

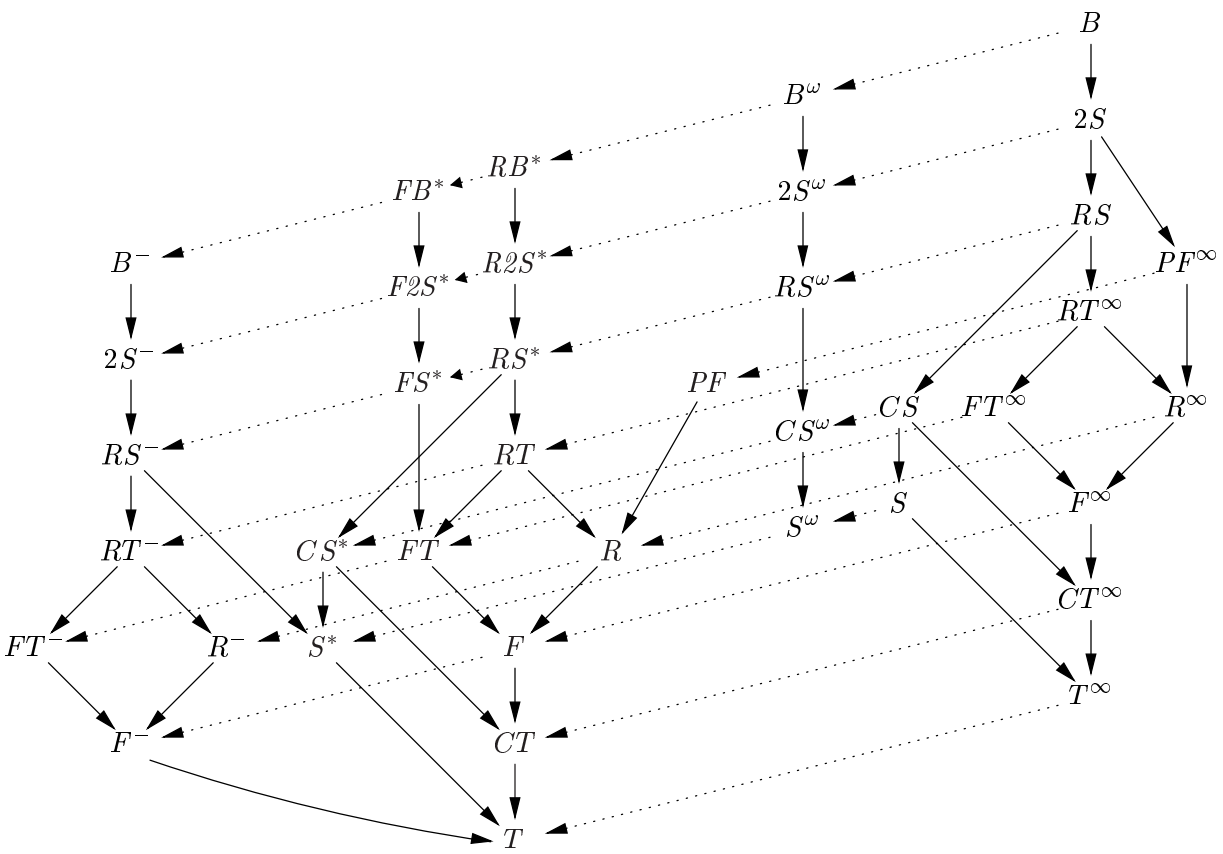


Figure 9: The infinitary linear time - branching time spectrum

14 Deterministic processes

If the labelled transition system $\mathbb{P}$ on which the semantic equivalences of Section 13 are defined is large enough, then they are all different and $\mathcal{S} \preceq_{\mathbb{P}} \mathcal{T}$ holds only if this is indicated in Figure 9. However, for certain labelled transition systems much more identifications can be made. Is has been remarked already that for image finite processes all semantics that are connected by dotted arrows coincide. In this section a class of processes will be defined on which all the semantics coincide.

Definition 14.1 A process p is *deterministic* if $p \xrightarrow{\sigma} q \wedge p \xrightarrow{\sigma} r \Rightarrow q = r$.

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 4 (Park [35]) On a domain of deterministic processes all semantics in the infinitary linear time – branching time spectrum coincide.

Proof: Because of Theorem 3 it suffices to show that on a domain $\mathbb{P}$ of deterministic processes one has $B \preceq_{\mathbb{P}} T$. This is the case if $T(p) = T(q) \Rightarrow p \leftrightarrows q$ for any two deterministic processes

$p, q \in \mathbb{P}$. Let R be the binary relation on $\mathbb{P}$ 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 $q' \in \mathbb{P}$ 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. □

15 Complete axiomatizations

A language for finite, concrete, sequential processes Consider the following basic CCS- and CSP-like language BCCSP for finite, concrete, sequential processes over a given 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 $\mathsf{T}(\mathrm{BCCSP})$ of (closed) *process expressions* or *terms* over this language is defined as usual:

- $0 \in \mathsf{T}(\mathrm{BCCSP})$,
- $ap \in \mathsf{T}(\mathrm{BCCSP})$ for any $a \in Act$ and $p \in \mathsf{T}(\mathrm{BCCSP})$,
- $p + q \in \mathsf{T}(\mathrm{BCCSP})$ for any $p, q \in \mathsf{T}(\mathrm{BCCSP})$.

Subterms $a0$ may be abbreviated by a. Brackets are used for disambiguation only, assuming associativity of $+$, and letting a bind stronger than $+$.

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

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

Table 1: Action rules for BCCSP

Now all semantic equivalences of Sections 2–12 are well-defined on $\mathsf{T}(\mathrm{BCCSP})$, and for each of the semantics it is determined when two process expressions denote the same process.

Axioms In Table 2, complete axiomatizations can be found for ten of the twelve semantics of this paper that differ on BCCSP. 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 5 For each of the semantics $O \in \{T,\ S,\ CT,\ CS,\ F,\ R,\ FT,\ RT,\ RS,\ B\}$ two process expressions $p, q \in \mathsf{T}(\mathrm{BCCSP})$ 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' or 'ω' are valid in O-semantics but not needed for the proof.

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

Table 2: Complete axiomatizations

Proof: For F, R and B the proof is given in Bergstra, Klop & Olderog [8] 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. □

Theorem 6 Suppose Act is infinite. For each of the semantics $O \in \{T, S, CT, F, R, FT, RT, RS, B\}$ two BCCSP expressions with variables are O-equivalent iff they can be proved equal from the axioms marked with '+' or 'ω' in the column for O in Table 2. It follows that the axioms marked with 'v' are derivable.

Proof: For $O \in \{T,\ CT,\ F,\ R,\ FT,\ RT,\ B\}$ this has been established by Jan Friso Groote [20]. His proof for F, R, FT and RT can be applied to S and RS as well. □

Groote also showed that if Act is finite, Theorem 3.2 does not hold for F, R, FT and RT. But for B and CT it suffices to assume that Act is nonempty, and for T it suffices to assume that Act has at least two elements. I don't know which cardinality restriction on Act is needed in the cases of S and RS. A complete axiomatization for open terms for completed simulation semantics has so far not been provided.

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 rôle 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. They also introduce *acceptance semantics*, which can be obtained from acceptance-refusal semantics (Section 7) by dropping the refusals, and analogously *acceptance trace semantics*. I'm not aware of any reasonable testing scenario for these notions.

In Section 9 I remarked that a testing scenario for simulation and ready simulation semantics can be obtained by adding an *undo*-button to the scenario's for trace and ready trace semantics. Likewise SCHNOEBELEN [41] investigates the addition of an *undo*-button to the testing scenarios for completed trace, readiness, failures and failure trace semantics, thereby obtaining 3 new equivalences $CT_\#$, $R_\#$ and $F_\#$. *Undo*-failure trace equivalence coincides with finitary failure simulation equivalence, just like *undo*-trace and *undo*-ready trace equivalence coincide with finitary simulation and finitary ready simulation equivalence. For image finite processes $R_\#$ coincides with $F_\#$. Furthermore $R \preceq R_\# \preceq RS^*$, $F \preceq F_\# \preceq FS^*$, $CT \preceq CT_\# \preceq CS^*$ and $S^* \preceq CT_\# \preceq F_\# \preceq R_\#$.

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 [9] and [21]. More research in this direction will be reported in [17].

An interesting topic is the generalization of this work to a setting with silent moves and/or with parallelism. In both cases there turn out to be many interesting variations. The generalization to a setting with invisible actions will be tackled in [17]. Some work towards generalizing the spectrum to a setting with parallelism can be found for instance in [38] and [16].

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 classify them as well. Also this will be done in [17].

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 failures semantics in [10] and for readiness semantics in [34].

References

[1] S. ABRAMSKY (1987): *Observation equivalence as a testing equivalence. Theoretical Computer Science* 53, pp. 225–241.

[2] S. ABRAMSKY & S. VICKERS (1993): *Quantales, observational logic and process semantics. Mathematical Structures in Computer Science* 3, pp. 161–227.

[3] P. ACZEL (1988): *Non-well-founded Sets, CSLI Lecture Notes* 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. 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 J.W. de Bakker, W.P. de Roever & G. Rozenberg, editors: *Current Trends in Concurrency*, LNCS 224, Springer, pp. 51–121.

[6] J.W. de Bakker & J.I. Zucker (1982): *Processes and the denotational semantics of concurrency. Information and Computation* 54(1/2), pp. 70–120.

[7] J.A. Bergstra, J.W. Klop & E.-R. Olderog (1986): *Failure semantics with fair abstraction.* Report CS-R8609, CWI, Amsterdam.

[8] 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.

[9] B. Bloom, S. Istrail & A.R. Meyer (1995): *Bisimulation can't be traced. Journal of the ACM* 42(1), pp. 232–268.

[10] S.D. Brookes, C.A.R. Hoare & A.W. Roscoe (1984): *A theory of communicating sequential processes. Journal of the ACM* 31(3), pp. 560–599.

[11] S.D. Brookes & A.W. Roscoe (1985): *An improved failures model for communicating processes.* In S.D. Brookes, A.W. Roscoe & G. Winskel, editors: *Seminar on Concurrency*, LNCS 197, Springer, pp. 281–305.

[12] P. Darondeau (1982): *An enlarged definition and complete axiomatisation of observational congruence of finite processes.* In M. Dezani-Ciancaglini & U. Montanari, editors: *Proceedings international symposium on programming: 5^{th} colloquium, Aarhus*, LNCS 137, Springer, pp. 47–62.

[13] R. De Nicola (1987): *Extensional equivalences for transition systems. Acta Informatica* 24, pp. 211–237.

[14] R. De Nicola & M. Hennessy (1984): *Testing equivalences for processes. Theoretical Computer Science* 34, pp. 83–133.

[15] R.J. van Glabbeek (1990): *The linear time – branching time spectrum.* Report CS-R9029, CWI, Amsterdam. Extended abstract in J.C.M. Baeten & J.W. Klop, editors: Proceedings *CONCUR '90, Theories of Concurrency: Unification and Extension*, Amsterdam, August 1990, LNCS 458, Springer-Verlag, 1990, pp. 278–297.

[16] R.J. van Glabbeek (1990): *The refinement theorem for ST-bisimulation semantics.* In M. Broy & C.B. Jones, editors: Proceedings IFIP TC2 Working Conference on *Programming Concepts and Methods*, Sea of Gallilee, Israel, North-Holland, pp. 27–52.

[17] R.J. van Glabbeek (1993): *The linear time – branching time spectrum II; the semantics of sequential systems with silent moves.* Preliminary version available from `boole.stanford.edu`. Extended abstract in *Proceedings CONCUR 93*, Hildesheim, Germany, LNCS 715, Springer-Verlag, pp. 66-81.

[18] 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*, CWI, Amsterdam, pp. 243–246.

[19] R.J. van Glabbeek, S.A. Smolka, B. Steffen & C.M.N. Tofts (1990): *Reactive, generative, and stratified models of probabilistic processes.* In *Proceedings 5^{th} Annual Symposium on Logic in Computer Science*, Philadelphia, USA, IEEE Computer Society Press, pp. 130–141.

[20] J.F. Groote (1990): *A new strategy for proving ω–completeness with applications in process algebra.* In J.C.M. Baeten & J.W. Klop, editors: *Proceedings CONCUR 90*, Amsterdam, LNCS 458, Springer, pp. 314–331.

[21] J.F. Groote & F.W. Vaandrager (1992): *Structured operational semantics and bisimulation as a congruence.* *Information and Computation* 100(2), pp. 202–260.

[22] M. Hennessy (1985): *Acceptance trees.* *Journal of the ACM* 32(4), pp. 896–928.

[23] M. Hennessy & R. Milner (1980): *On observing nondeterminism and concurrency.* In J.W. de Bakker & J. van Leeuwen, editors: *Proceedings 7^{th} ICALP, Noorwijkerhout*, LNCS 85, Springer, pp. 299–309. This is a preliminary version of: Algebraic laws for nondeterminism and concurrency. *JACM*, 32(1):137–161, 1985.

[24] M. Hennessy & R. Milner (1985): *Algebraic laws for nondeterminism and concurrency.* *Journal of the ACM* 32(1), pp. 137–161.

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

[26] C.A.R. Hoare (1980): *Communicating sequential processes.* In R.M. McKeag & A.M. Macnaghten, editors: *On the construction of programs – an advanced course*, Cambridge University Press, pp. 229–254.

[27] C.A.R. Hoare (1985): *Communicating Sequential Processes.* Prentice Hall, Englewood Cliffs.

[28] J.K. Kennaway (1981): *Formal semantics of nondetermism and parallelism.* PhD thesis, University of Oxford.

[29] K.G. Larsen & A. Skou (1991): *Bisimulation through probabilistic testing.* *Information and Computation*, 94.

[30] A.R. Meyer (1985): *Report on the 5^{th} international workshop on the semantics of programming languages in Bad Honnef.* *Bulletin of the European Association for Theoretical Computer Science* 27, pp. 83–84.

[31] R. Milner (1980): *A Calculus of Communicating Systems*, LNCS 92. Springer.

[32] R. Milner (1981): *Modal characterisation of observable machine behaviour.* In G. Astesiano & C. Bohm, editors: *Proceedings CAAP 81*, LNCS 112, Springer, pp. 25–34.

[33] R. Milner (1983): *Calculi for synchrony and asynchrony.* *Theoretical Computer Science* 25, pp. 267–310.

[34] E.-R. Olderog & C.A.R. Hoare (1986): *Specification-oriented semantics for communicating processes. Acta Informatica* 23, pp. 9–66.

[35] D.M.R. Park (1981): *Concurrency and automata on infinite sequences.* In P. Deussen, editor: 5^{th} *GI Conference*, LNCS 104, Springer, pp. 167–183.

[36] I.C.C. Phillips (1987): *Refusal testing. Theoretical Computer Science* 50, pp. 241–284.

[37] A. Pnueli (1985): *Linear and branching structures in the semantics and logics of reactive systems.* In W. Brauer, editor: *Proceedings* 12^{th} *ICALP, Nafplion*, LNCS 194, Springer, pp. 15–32.

[38] L. Pomello (1986): *Some equivalence notions for concurrent systems – An overview.* In G. Rozenberg, editor: *Advances in Petri Nets 1985*, LNCS 222, Springer, pp. 381–400.

[39] A.W. Roscoe (1993): *Unbounded non-determinism in CSP. Journal of Logic and Computation* 3(2), pp. 131–172.

[40] W.C. Rounds & S.D. Brookes (1981): *Possible futures, acceptances, refusals and communicating processes.* In 22^{th} *Annual Symposium on Foundations of Computer Science*, Nashville, Tennessee, IEEE, New York, pp. 140–149.

[41] Ph. Schnoebelen (1991): *Experiments on processes with backtracking.* In J.C.M. Baeten & J.F. Groote, editors: *Proceedings CONCUR 91, Amsterdam*, LNCS 527, Springer, pp. 80–94.

Index

$\frac{2}{3}$-bisimulation equivalence, 4, 29
2-nested simulation, 33
2-nested simulation equivalence, 33
2-nested simulation formulas, 33
2-nested simulation machine, 33
2-nested simulation semantics, 4, 33

acceptance semantics, 45
acceptance trace semantics, 45
acceptance tree, 13
acceptance tree equivalence, 13
acceptance-refusal equivalent, 22
acceptance-refusal semantics, 22
acceptance-refusal triple, 22
action, 5
action relations, 3, 6
action rules, 43
algebra, 2
anti-foundation axiom, 37

barbed semantics, 4
bisimilar, 34
bisimulation, 34, 36
bisimulation equivalence, 34
bisimulation semantics, 3, 34
blocked, 11
branching, 5
branching equivalent, 38
button pushing experiments, 5

canonical graph, 7
comparative concurrency semantics, 1
complete simulation, 29
complete trace, 9
completed simulation equivalent, 28
completed trace equivalent, 9
completed trace formulas, 10
completed trace machine, 10
completed trace semantics, 3, 9
concrete, 3
connected, 7
continuations, 12

deadlock behaviour, 41
denial formulas, 32
deterministic, 42
domain, 2
domain theory, 2

edges, 6
embedding, 7
event structure, 2
event structure domains, 2
exhibited behaviour semantics, 4
explicit domains, 3

failure formulas, 12
failure pair, 12
failure set, 12, 13
failure simulation equivalent, 28
failure trace, 15, 16
failure trace augmentation, 17
failure trace equivalent, 16
failure trace formulas, 16
failure trace machine, 15
failure trace relations, 16
failure trace semantics, 4, 15
failure trace set, 15, 16
failure trace simulation equivalent, 28
failures equivalent, 12
failures machine, 11
failures semantics, 3, 12
finitary simulation equivalent, 27
finitary simulation formulas, 27
finite, 6, 41
finite ready trace formulas, 23
finite-failure trace equivalent, 17
finite-failures equivalent, 14
finite-failures semantics, 14
finite-ready trace equivalent, 24
finitely branching, 3, 6
free, 11

generalized action relations, 6
generative processes, 31
global testing, 35
graph domains, 2
graph isomorphism, 7
graph transformations, 44

GSOS trace congruence, 4, 29, 32

Hennessy-Milner formulas, 38
Hennessy-Milner logic (HML), 38
HML-equivalent, 39

image finite, 6
infinitary completed trace equivalent, 11
infinitary failure trace equivalent, 17
infinitary failures equivalent, 14
infinitary Hennessy-Milner formulas, 34
infinitary possible future, 24
infinitary possible-futures equivalent, 24
infinitary ready equivalent, 23
infinitary ready trace equivalent, 21
infinitary simulation equivalence, 27
infinitary trace equivalent, 9
infinitary trace semantics, 9
infinite failure trace, 17
infinite ready trace, 21
infinite trace, 9
initial actions, 6
initial state, 6
isometry, 38
isomorphic, 7

labelled transition system, 3, 6
linear time – branching time, 3
lookahead, 33

menu, 11, 19
modelling, 1

net domains, 2
nodes, 6
normal ready trace, 19

observational equivalence, 3, 39

partial traces, 3
path, 6
Petri net, 2
possible future, 22
possible-futures equivalent, 22
possible-futures semantics, 4, 22
process, 1
process domain, 2
process expressions, 43
process graph, 6
process theory, 1
projective limit domains, 3

reactive machines, 31
readiness formulas, 21
readiness machine, 21
readiness semantics, 3, 21
ready equivalent, 21
ready pair, 21, 22
ready set, 21
ready simulation, 29
ready simulation equivalent, 28
ready simulation machine, 31
ready simulation semantics, 4, 28
ready trace, 19
ready trace equivalent, 19
ready trace formulas, 20
ready trace machine, 18
ready trace relations, 19
ready trace semantics, 4, 19
ready trace set, 19, 20
ready trace simulation equivalent, 28
refusal equivalence, 15
refusal relations, 15
refusal semantics, 4
refusal set, 12, 15
refusal testing, 15
replicator, 28
root, 6

satisfaction relation, 8, 10, 12, 16, 20, 21, 23, 25, 27, 33, 34, 38
semantic equivalence, 2
semantics, 1
sequential, 3, 5
similar, 24
simulation, 4, 24, 25
simulation equivalence, 24
simulation formulas, 25
simulation semantics, 4, 24
state transition diagram, 2
states, 6

term domains, 3
terms, 43

testing equivalences, 3
trace, 8
trace equivalent, 8
trace formulas, 8
trace machine, 10
trace semantics, 3, 8
transitions, 6
tree, 7

uniform concurrency, 2
uniformly, 3
universal algebra, 2

verification, 1

weather, 35
well-founded, 6