Ensuring Liveness Properties of Distributed Systems: Open Problems

Rob van Glabbeek
Data61, CSIRO, Sydney, Australia
School of Computer Science and Engineering, University of New South Wales, Sydney, Australia
rvg@cs.stanford.edu

Often fairness assumptions need to be made in order to establish liveness properties of distributed systems, but in many situations they lead to false conclusions.

This document presents a research agenda aiming at laying the foundations of a theory of concurrency that is equipped to ensure liveness properties of distributed systems without making fairness assumptions. This theory will encompass process algebra, temporal logic and semantic models. The agenda also includes the development of a methodology and tools that allow successful application of this theory to the specification, analysis and verification of realistic distributed systems.

Contemporary process algebras and temporal logics fail to make distinctions between systems of which one has a crucial liveness property and the other does not, at least when assuming *justness*, a strong progress property, but not assuming fairness. Setting up an alternative framework involves giving up on identifying strongly bisimilar systems, inventing new induction principles, developing new axiomatic bases for process algebras and new congruence formats for operational semantics, and creating matching treatments of time and probability.

Even simple systems like fair schedulers or mutual exclusion protocols cannot be accurately specified in standard process algebras (or Petri nets) in the absence of fairness assumptions. Hence the work involves the study of adequate language or model extensions, and their expressive power.

1 State-of-the-art

1.1 Specification, analysis and verification of distributed systems

At an increasing rate, humanity is creating distributed systems through hardware and software—systems consisting of multiple components that interact with each other through message passing or other synchronisation mechanisms. Examples are distributed databases, communication networks, operating systems, industrial control systems, etc. Many of these systems are hard to understand, yet vitally important. Therefore, significant effort needs to be made to ensure their correct working.

Formal methods are an indispensable tool towards that end. They consist of specification formalisms to unambiguously capture the intended requirements and behaviour of a system under consideration, tools and analysis methods to study and reason about vital properties of the system, and mathematically rigorous methods to verify that (a) a system specification ensures the required properties, and (b) an implementation meets the specification.

The standard alternative to formal specification formalisms are descriptions in English, or other natural languages, that try to specify the requirements and intended workings of a system. History has shown, almost without exception, that such descriptions are riddled with ambiguities, contradictions and under-specification. Formalisation of such a description—regardless in which formalism—is the key to elimination of these holes.

A formal specification of a distributed system typically comes in (at least) two parts.

July 2019. To appear in Ilaria Castellani, Pedro D'Argenio, Mohammad Reza Mousavi & Ana Sokolova, eds.:
"Open Problems in Concurrency Theory", a special issue of the Journal of Logical and Algebraic Methods in Programming.

One part formulates the *requirements* imposed on the system as a list of properties the system should have. Amongst the formalisms to specify such requirements are temporal logics like Linear-time Temporal Logic (LTL) [Pnu77] or Computation Tree Logic (CTL) [EC82]. Amongst others, they can specify *safety properties*, saying that something bad will never happen, and *liveness properties*, saying that something good will happen eventually [Lam77].

The other part is a formal description of how the system ought to work on an *operational* (= step by step) basis, but abstracting from implementation details. For distributed systems such accounts typically consist of descriptions of each of the parallel components, as well as of the communication interfaces that specify how different components interact with each other. Languages for giving such formal descriptions are *system description languages*. When a system description language features constants to specify elementary system activities, and operators (like parallel or sequential composition) to create more complex systems out of simpler ones, it is sometimes called a *process algebra*. Alternatively, operational system descriptions can be rendered in a model of concurrency, such as Petri nets or labelled transition systems. Such models are also used to describe the meaning of system description languages.

Once such a two-tiered formalisation of a system has been provided, there are two obvious tasks to ensure the correct working of implementations: (a) guaranteeing that the operational system description meets the requirements imposed on the system, and (b) ensuring that an implementation satisfies the specification. The latter task additionally requires a definition of what it means for an implementation to satisfy a specification, and this definition should ensure that any relevant correctness properties that are shown to hold for the specification also hold for the implementation.

A third type of task is the study of other properties of the implementation, not implied by the specification. Examples are measuring its execution times, when these are not part of the specification, or its success rate, for operations for which success cannot be guaranteed and only a best effort is made. Potentially, these tasks call for applications of probability theory.

Traditional approaches to ensure the correct working of distributed systems are simulation and test-bed experiments. While these are important and valid methods for system evaluation, in particular for quantitative performance evaluation, they have limitations in regards to the evaluation of basic correctness properties. Experimental evaluation is resource-intensive and time-consuming, and, even after a very long time of evaluation, only a finite set of operational scenarios can be considered—no general guarantee can be given about correct system behaviour for a wide range of unpredictable deployment scenarios. I believe that formal methods help in this regard; they complement simulation and test-bed experiments as methods for system evaluation and verification, and provide stronger and more general assurances about system properties and behaviour.

1.2 Achievements of process algebra and related formalisms

Process algebra is a family of approaches to the specification, analysis and verification of distributed systems. Its tools encompass algebraic languages for the specification of systems (mentioned above), algebraic laws to reason about system descriptions, and induction principles to derive behaviours of infinite systems from those of their finite approximations.

Many industrial size distributed systems have been successfully specified, analysed and verified in frameworks based on process algebra–examples can be found through the following links. Major toolsets primarily based on process algebra include FDR [GABR14], CADP [GLMS11], mCRL2 [GM14] and the Psi-Calculi Workbench [BGRV15, BJPV11]. Key methods employed are *model checking* [BK08] to ensure that an operational system description meets system requirements, and *equivalence checking* [CS01] or *refinement* [Mor94], to show that an operational description of an implementation is equivalent to or at least as suitable as an operational system specification, in the sense that it inherits all its relevant good properties.

Additional toolsets primarily based on model checking or other mathematical techniques that explore the state spaces of distributed systems include SPIN [Hol04], UPPAAL [BDL04], LTSmin [KLM$^+$15], PRISM [KNP10] and TLA [Lam02].

1.3 Liveness and fairness assumptions

One of the crucial tasks in the analysis of distributed systems is the verification of liveness properties, saying that something good will happen eventually. A typical example is the verification of a communication protocol—such as the *alternating bit protocol* [Lyn68, BSW69]—that ensures that a stream of messages is relayed correctly, without loss or reordering, from a sender to a receiver, while using an unreliable communication channel. The protocol works by means of acknowledgements, and resending of messages for which no acknowledgement is received.

Naturally, no protocol is able to ensure such a thing, unless one assumes that attempts to transmit a message over the unreliable channel will not fail in perpetuity. Such an assumption, essentially saying that if one keeps trying something forever it will eventually succeed, is often called a *fairness* assumption. See [GH18] for a formal definition of fairness, and an overview of notions of fairness from the literature.

Making a fairness assumption is indispensable when verifying the alternating bit protocol. If one refuses to make such an assumption, no such protocol can be found correct, and one misses a chance to check the protocol logic against possible flaws that have nothing to do with a perpetual failure of the communication channel.

For this reason, fairness assumptions are made in many process-algebraic verification methods, and are deeply ingrained in their methodology [BBK87]. This applies, amongst others, to the default incarnations of the process algebras CCS [Mil89], ACP [BW90, Fok00a, BBR10], the π-calculus [Mil99, SW01] and mCRL2 [GM14].

1.4 Fairness assumptions in process algebra

This section explains the last claim, and can be skipped by anyone unfamiliar with these process algebras. A typical process-algebraic verification of—say—the alternating bit protocol [BK86b] starts from process algebraic descriptions of the specification as well as the implementation. Each comes as an expression in a suitable process-algebraic specification language, and is interpreted as a state in a labelled transition system. The specification describes the intended behaviour of the protocol, and does not model message loss due to unreliable communication channels. The implementation describes the parallel composition of the sender, the receiver, and the unreliable channels for messages and acknowledgements. All unsuccessful communication attempts manifest themselves as cycles of internal transitions (labelled with the unobservable action τ) in the labelled transition system that forms the semantic model of the composed implementation. The verification consists of showing that the implementation is semantically equivalent to the specification, using an equivalence such as *branching bisimulation equivalence* [GW96] or *weak bisimulation equivalence* [Mil89]. These semantic equivalences abstract from cycles of internal actions, and this is essential for a successful verification. The technique of proving liveness properties through abstraction from τ-cycles is called *fair abstraction* [BK86b, BRV96]. It amounts to applying a particular strong fairness assumption, called *full fairness* in [GH18]. In [BK86b] the equivalence of specification and implementation is obtained by algebraic manipulation of process-algebraic expressions, and here the abstraction from τ-cycles is captured by *Koomen's Fair Abstraction Rule*. The soundness of this proof rule w.r.t. weak bisimulation equivalence is established in [BBK87].

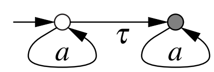
A popular alternative to bisimulation equivalence for relating specifications and implementations is the *must testing equivalence* of [DH84], which coincides with the *failures equivalence* of CSP [BHR84, Hoa85, Ros97]. It does not embody fairness assumptions. However, a variant that incorporates fair abstraction has been proposed in [Vog92, BRV95, NC95].

2 The dangers of assuming fairness

Using a fairness assumption, however, needs to be done with care. Making a fairness assumption can lead to patently false results. This applies for instance to the alternating bit protocol in cases where one of the possible behaviours of the unreliable channel is to perpetually lose all messages.

In the study of routing protocols for wireless networks, a desirable liveness property is *packet delivery* [FGH$^+$12, FGH$^+$13]. It says that a data packet injected at a source node will eventually be delivered at its destination node, provided (a) the source node is connected to the target node through a sequence of 1-hop connections where each two adjacent nodes are within transmission range of each other, and (b) no 1-hop connection in the entire network breaks down before the data packet is delivered. In a reasonable model of a wireless network all connections between nodes can nondeterministically appear or disappear in any state. As a consequence, transitions leading to successful delivery of a packet remain possible as long as the packet is not delivered, and for this reason the assumption of full fairness is sufficient to establish packet delivery even without the side conditions (a) and (b).[1] Nevertheless, such a result has little bearing on reality in wireless networks.

I quote an example from [GH18] as another illustration of this phenomenon. Consider the CCS process $Q := (X|b)\backslash b$ where $X \stackrel{def}{=} a.X + \bar{b}.X$. The syntax and semantics of the process algebra CCS, in which this example is specified, can be found in Appendix A. The transition system of this process is depicted on the right.[2] The process X models the behaviour of relentless Alice, who either picks up her phone when Bob is calling ($\bar{b}$), or performs another activity (a), such as eating an apple. In parallel, b models Bob, constantly trying to call Alice; the action b models the call that takes place when Alice answers the phone. A desired (by Bob) liveness property $\mathscr{G}$ is to achieve a phone connection with Alice, i.e. to reach the rightmost state in the above transition system. Under each of the notions of strong and weak fairness reviewed in [GH18], Q satisfies $\mathscr{G}$. Yet, it is perfectly reasonable that the connection is never established: Alice could never pick up her phone, as she is not in the mood of talking to Bob; maybe she totally broke up with him.



My last example is taken from [Gla19a]. Suppose Bart stands behind a bar and wants to order a beer. But by lack of any formal queueing protocol many other customers get their beer before Bart does. This situation can be modelled as a transition system where in each state in which Bart is not served yet there is an outgoing transition modelling that Bart gets served, but there are also outgoing transitions modelling that someone else gets served instead. The essence of fairness is the assumption that Bart will get his beer eventually. Fairness rules out as unfair any execution in which Bart could have gotten a beer any time, but never will. Yet, this possibility can not be ruled out in reality.

These examples are not anomalies; they describe a default situation. A fairness assumption says, in essence, that if you try something often enough, you will eventually succeed. There is nothing in our understanding of the physical universe that supports such a belief. Fairness assumptions are justified in exceptional situations, the verification of the alternating bit protocol being a good example. However, by default they are unwarranted.

2.1 Probabilistic arguments in favour of fairness

In defence of making a fairness assumption it is sometimes argued that whenever at some point the probability of success is 0, the success possibility should not be part of our model of reality. When

[1] All relevant transitions up to the final delivery can be relabelled τ, and this cloud of τ-transitions has only one exit, successful delivery, which remains reachable throughout. Full fairness allows abstraction from this cloud of τ-transition.

[2] States are depicted by circles and transitions by arrows between them. An initial state is indicated by a short arrow without a source state. When a liveness property modelled as a set of goal states is involved, these states are indicated by shading.

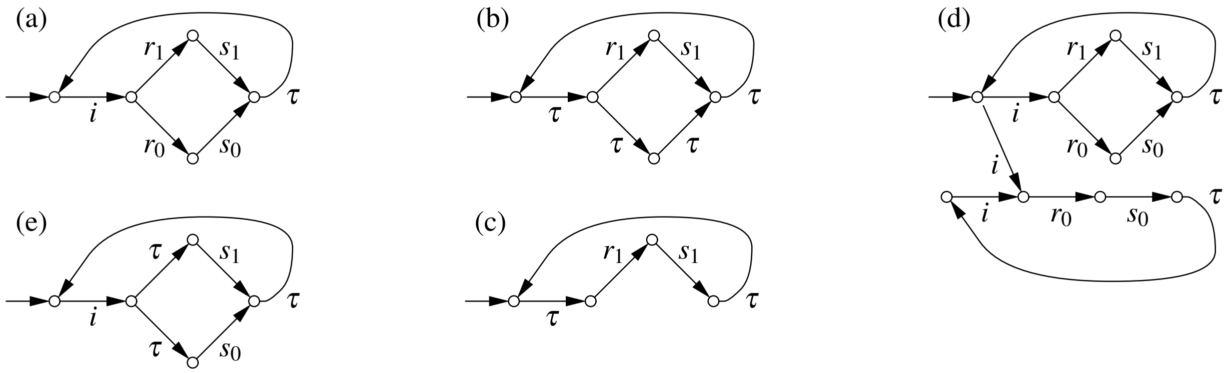


Figure 1: The meaning of choice in reactive systems

infinitely many choices remain that allow success with a fixed positive probability, with probability 1 success will be achieved eventually. This argument rests on assumptions on relative probabilities of certain choices, but is applied to models that abstract from those probabilities.

My counter-arguments are that (1) when abstracting from probabilities it is quite well possible that a success probability is always positive, yet quickly diminishing, so that the cumulative success probability is less than 1, and (2) that in many applications one does not know whether certain behaviours have a chance of occurring or not, but they are included in the model nevertheless.

2.2 Demonic choice in reactive systems

The issue is further illustrated by the labelled transition system of Figure 1(a). It depicts a system B that, after an initialisation action i, reads a Boolean value on a certain input channel. The action r_b indicates the reading of $b \in \{0,1\}$. Subsequently it forwards the received value b on a different channel by performing the action s_b. Then, via an internal action τ, B returns to its initial state. Thus B functions as a one-bit buffer.

Using *fair abstraction*, e.g., standard process algebraic reasoning as described in Section 1.4, one can prove that this system will eventually receive and forward the Boolean value 1. Namely, one renames all actions one is not interested in into the internal action τ, thereby obtaining the system of Figure 1(b). This system is weakly (and branching) bisimulation equivalent with the system depicted in Figure 1(c), and the latter clearly receives and forwards value 1 eventually.

In my view, this is another example showing that the assumption of fairness is by default unwarranted. For the reactive system B could well be placed in an environment that will never provide the value 1 on the modelled input channel.

A counterargument has been brought to my attention by one of the referees of this paper. Namely a transition system as in Figure 1(a) denotes a system with the property that each execution of the action i reaches a state from which it is actually possible that action r_1 will occur. This is at odds with my "placement" of B in an environment where this is not going to ever happen. If one wants to model the possible placement of B in such an environment, one might need a transition system that looks more like Figure 1(d). Here one sees the possibility that during the execution of the action i, system B falls into an environment that will never provide the input 1.

In this point of view, my argument against the use of fairness loses its force. For the system of Figure 1(d) surely is not guaranteed to ever receive and forward a 1, regardless whether one employs fairness of not. The system of Figure 1(a) on the other hand is one that will not be placed in such an environment, and here the probabilistic argument could be used to argue that it is very unlikely that one will never observe r_1.

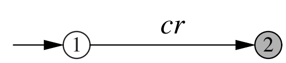
In my opinion the modelling of a reactive system ought to be independent of the environment in which it is going to operate. Figure 1(d) shows a bad model of B, as it may reach a state where it no longer accepts the value 1. The better representation is Figure 1(a), and its meaning should allow an environment as discussed above.

The example above involves a choice between r_0 and r_1 that is entirely triggered by the environment. This is often called an *external choice* [Ros97]. One may wonder if the same reasoning applies to an *internal* or *nondeterministic* choice. Figure 1(e) shows a system that after the action i nondeterministically chooses to either send 0 or 1 on its output channel. Here one may argue that Figure 1(e) models a system where after each occurrence of i both choices are open, so that there must be a positive probability that s_1 will occur. My argument (1) above that even this does not guarantee that s_1 will ever occur, amounts to saying that the probability of doing s_1 could diminish quickly after each occurrence of i. My referee answers that in this scenario "the Markovian assumption is being violated, meaning that another state, with a different choice semantics, is being entered after each choice". This could presumably be modelled by an infinite-state transition system, but not by one such as Figure 1(e).

My point of view is that a nondeterministic choice such as depicted in Figure 1(e) is really an external choice triggered by an aspect of the environment that is outside our grasp. Either one has chosen to abstract from this aspect, or one does not know what truly causes the decision. Thus, Figure 1(e) can be seen as representing the system from Figure 1(a) but with the actions r_b depicted as τ. Consequently, this system might run in environments were always the choice leading to r_0 is taken. It might also run in environments where an unknown adversary rolls an increasingly unfair dice at each choice point. This view of nondeterminism is sometimes called *demonic choice* [MM01].

3 Progress and justness

Having reached the point where I advocate proving liveness properties of distributed system without resorting to fairness assumption, the question arises whether any weaker assumptions in lieu of fairness are appropriate. My answer is a resounding *yes*. The least that should always be assumed when proving liveness properties is what I call *progress*. Without a progress assumption no meaningful liveness properties can be established.



To illustrate this concept, consider the transition system on the right, taken from [Gla19a]. It models Cataline eating a croissant in Paris and abstracts from all activity in the world except the eating of that croissant. It thus has two states only—the states of the world before and after this event—and one transition. A possible liveness property $\mathcal{G}$ says that the croissant will be eaten. It corresponds with reaching state 2. This liveness property does not hold if the system may remain forever in the initial state 1. The assumption of progress rules out that behaviour. In the context of *closed systems*, having no run-time interactions with the environment, it is the assumption that a system will never get stuck in a state with outgoing transitions. It is only when assuming progress that $\mathcal{G}$ holds.

For *reactive systems*, having run-time interactions with their environment, the progress assumption as formulated above would rule out too many behaviours. Take for instance the one-bit buffer of Figure 1(a). When assuming that the system can not get stuck in a state with outgoing transitions, it would follow that either s_0 or s_1 will occur. Yet, in real life the system may be stuck in the state right after i, due to the environment not providing any value on the system's input channel. In general, a transition may represent an interaction between the distributed system being modelled and its environment. In many cases it can occur only if both the modelled system *and* the environment are ready to engage in it. I therefore distinguish *blocking* and *non-blocking* transitions [GH15b].[3] A transition is non-blocking if

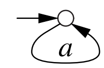
[3] In [FGH$^+$13] the *internal* and *output* transitions constitute the non-blocking ones. In [Rei13] blocking and non-blocking transitions are called *cold* and *hot*, respectively.

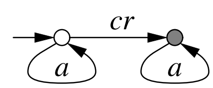
the environment cannot or will not block it, so that its execution is entirely under the control of the system under consideration. A blocking transition on the other hand may fail to occur because the environment is not ready for it. In [GH18], paraphrasing [FGH+13, GH15b], the assumption of progress was formulated as follows:

> *A (transition) system in a state that admits a non-blocking transition will eventually progress, i.e., perform a transition.*

In other words, a run will never get stuck in a state with outgoing non-blocking transitions.

Justness is an assumption strengthening progress, proposed in [FGH+13, GH15b, GH18]. It can be argued that once one adopts progress it makes sense to go a step further and adopt even justness.





The transition system on the right models Alice making an unending sequence of phone calls in London. There is no interaction of any kind between Alice and Cataline. Yet, I may choose to abstract from all activity in the world except the eating of the croissant by Cataline, and the making of calls by Alice. This yields the combined transition system on the bottom right. Even when taking the *cr*-transition to be non-blocking, progress is not a strong enough assumption to ensure that Cataline will ever eat the croissant. For the infinite run that loops in the first state is progressing. Nevertheless, as nothing stops Cataline from making progress, in reality *cr* will occur. [GH18, Gla19a]

This example is not a contrived corner case, but a rather typical illustration of an issue that is central to the study of distributed systems. Other illustrations of this phenomenon occur in [FGH+13, Section 9.1], [GH15a, Section 10], [Gla15, Section 1.4] and [DGH17, Section 4]. The assumption of justness aims to ensure the liveness property occurring in these examples. In [GH18] it is formulated as follows:

> *Once a non-blocking transition is enabled that stems from a set of parallel components, one (or more) of these components will eventually partake in a transition.*

In the above example, *cr* is a non-blocking transition enabled in the initial state. It stems from the single parallel component Cataline of the distributed system under consideration. Justness therefore requires that Cataline must partake in a transition. This can only be *cr*, as all other transitions involve component Alice only. Hence justness says that *cr* must occur. The infinite run starting in the initial state and not containing *cr* is ruled out as unjust.

A formal definition of justness is supplied in Appendix B. I believe reading it is not necessary to understand the forthcoming material.

4 A hierarchy of completeness criteria

In [Gla19a], assumptions like progress, justness and fairness are called *completeness criteria*. They serve to rule out certain runs of distributed systems that appear to be valid in their representations as transition systems, on grounds that such runs are assumed not to occur in practice. The completeness criterion "progress" for instance, applied to the transition system on page 6, where *cr* is assumed non-blocking, rules out the run in which Cataline does not eat her croissant (i.e. where no transition is ever taken).

One completeness criterion is called *stronger* than another if it rules out more runs. The weakest completeness criterion is the empty one ($\emptyset$)—it rules out no runs. Figure 2 orders several completeness

full fairness
|
strong fairness of componenets
|
weak fairness of componenets
|
J-fairness of componenets
|
justness
|
progress
|
$\emptyset$

Figure 2: A hierarchy of completeness criteria

criteria on strength. Here progress and justness are described in Section 3 and formalised in Appendix B. The three forms of fairness of components are formally defined in Appendix C, taken from [GH18]. The hierarchy of Figure 2 is supported by Proposition 1 in Appendix C. Many other notions of fairness are classified in [GH18]; some of them have a strength incomparable to justness. One notion of fairness from the literature, named *fairness of events* in [GH18], is shown to coincide with justness [GH18, Theorem 15.1]. However, it has been defined only for the special case that the set of blocking actions is empty. Fairness of events was first studied in [CS84]—although on a restriction-free subset of CCS where it coincides with fairness of components—and then in [CDV06a], under the name fairness of actions.

A liveness property holds for a distributed system iff it holds for each of its runs. It holds under a progress, justness or fairness assumption, or in general when adopting a particular completeness criterion, if it holds for all runs that are not ruled out by that assumption or criterion. So a liveness property is more often satisfied when adopting a stronger completeness criterion.

The concept of *full fairness* described in Section 1.4 and illustrated in Section 2.2 is also formalised in [GH18]. It is not a fairness notion as defined in Appendix C, since it does not rule out a specific set of runs. Yet its strength can be compared with other notions of fairness based on which liveness properties are obtained by making this assumption. Full fairness turns out to be the strongest of all possible fairness assumptions [GH18].

5 Process algebras without fairness assumptions, and their limitations

Strong bisimulation equivalence [Mil89, Gla11a], also called *strong bisimilarity*, is one of the most prominent semantic equivalences considered in concurrency theory. Virtually all semantic equivalences or refinement preorders employed in process algebra are coarser than or equal to strong bisimilarity; that is, they identify strongly bisimilar systems.

Here I will argue that these approaches *cannot* make sufficiently strong progress assumptions to establish meaningful liveness properties in realistic applications. Namely, I will show two programs, P and Q, that are strongly bisimilar, and hence equated in virtually all process-algebraic approaches to date. Yet, there is a crucial liveness property that holds for P but not for Q, when assuming justness but not fairness. So the process algebra must either claim that both programs have the liveness property, which in case of Q could be an unwarranted conclusion, possibly leading to the design of systems with dangerous or catastrophic behaviour, or it falls short in asserting the liveness property of P.

$$x := 1 \quad \| \quad \textbf{repeat} \;\; y := y+1 \;\; \textbf{forever} \tag{P}$$

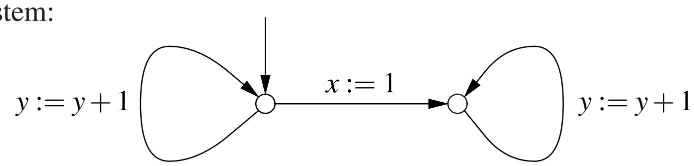
Program P is the parallel composition of two non-interacting processes, one of which sets the variable x to 1, and the other repeatedly increments a variable y. I assume that both variables x and y are initialised to 0. The reader may assume that x is a local variable maintained by one component, and y by the other, or that x and y reside in central memories available to both components; but in the latter case x and y reside in completely disconnected central memories, so that an access to the variable y by the right component in no way interferes with an access to variable x by the left one.

In program Q the *case*-statement is interpreted such that if the conditions of multiple cases hold, a non-deterministic choice is made which one to execute. The conditional write **if** $x=0$ **then** x:=1 **fi** describes an atomic read-modify-write (RMW) operation[4]. Such operators, supported by modern hardware, read a memory location and simultaneously write a new value into it that may be a function of the previous value.

```
repeat
  case
    if True then y:=y+1 fi
    if x = 0 then x:=1 fi
  end
forever                              (Q)
```

[4] `https://en.wikipedia.org/wiki/Read-modify-write`

The programs P and Q are strongly bisimilar; both can be represented by means of the following labelled transition system:



As a warm-up exercise, one may ask whether the variable y in P or Q will ever reach the value 7—a liveness property. A priori, I cannot give a positive answer, for one can imagine that after incrementing y three times, the program for no apparent reason stops making progress and does not get around to any further activity. In most applications, however, it is safe to assume that this scenario will not occur. To accurately describe the intended behaviour of P or Q, or any other program, one makes a progress assumption as described in Section 3, saying that if a program is in a state where further activity is possible (and this activity is not contingent on input from the environment that might fail to occur) some activity will in fact happen. This assumption is sufficient to ensure that in P or Q the variable y will at some point reach the value 7.

Progress assumptions are commonplace in process algebra and many other formalisms. They are explicitly or implicitly made in CCS, ACP, the π-calculus, CSP, etc., whenever such formalisms are employed to establish liveness properties. Temporal logics, such as LTL [Pnu77] and CTL [EC82], have progress assumptions built in, namely by disallowing states without outgoing transitions and evaluating temporal formulas by quantifying over infinite paths only; they can formalise the statement that y will in fact reach the value 7.

A more interesting question is whether x will ever reach the value 1. This liveness property is *not* guaranteed by progress assumptions as made in any of the standard process algebras or temporal logics. The problem is that all these formalisms rest on a model of concurrency where parallel composition is modelled as arbitrary interleaving. The programs P and Q have computations like

$$\begin{array}{lllllll}
x := 1; & y := y+1; & y := y+1; & y := y+1; & y := y+1; & y := y+1; & \dots \\
y := y+1; & x := 1; & y := y+1; & y := y+1; & y := y+1; & y := y+1; & \dots \\
y := y+1; & y := y+1; & y := y+1; & y := y+1; & x := 1; & y := y+1; & \dots
\end{array}$$

where the action $x := 1$ can be scheduled arbitrary far in the sequence of y-incrementations, but also a computation

$$y := y+1; \quad y := y+1; \quad y := y+1; \quad y := y+1; \quad y := y+1; \quad y := y+1; \quad \dots \qquad (C^\infty)$$

in which $x := 1$ never happens, because $y := y+1$ is always scheduled instead. For this reason, temporal logic as well as process algebra—when not making fairness assumptions—say that x is not guaranteed to reach the value 1, regardless whether talking about P or Q.

When assuming that parallel composition is implemented by means of a scheduler that arbitrarily interleaves actions from both processes, this conclusion for P appears plausible. However, when $\|$ denotes a true parallel composition, where the program P consists of two completely independent processes, it appears more reasonable to adopt the assumption of justness from Section 3, which guarantees that x will reach the value 1. However, whereas justness disqualifies the computation C^∞ for P, it rightly allows it for Q. As the choice between the two cases of the **case**-statement is made by forces outside our grasp—see Section 2.2—one may not rule out the possibility that the first case is chosen every round.

Liveness goal:	$y=7$		$x=1$	
Program	P	Q	P	Q
full fairness	+	+	+	+
justness	+	+	+	–
progress	+	+	–	–
$\emptyset$	–	–	–	–

Figure 3: *Liveness properties obtained as a function of assumptions made*

A sufficiently strong fairness assumption would (unjustly) eliminate this computation even for Q—see Figure 3; here I argue for a theory of concurrency in which such a fairness assumption is not made.

Hence, virtually all existing process-algebraic approaches equate two programs, of which one has the liveness property that eventually x will reach the value 1, and the other does not, at least not without assuming fairness. So those approaches that do not assume fairness [DH84, BHR84, Hoa85, Wal90, Ros97, GLT09] lack the power to establish this property for P.

Variations on this example Readers that prefer the states in the transition systems for P and Q to be valuations of the variables x and y may easily unfold the given transition system into an infinite-state one with this property. This unfolding preserves the state of affairs that P and Q are strongly bisimilar systems of which one has a liveness property that the other lacks (when assuming justness but not fairness).

For readers that dislike the RMW operations in Q, one can easily skip the preconditions `True` and $x = 0$. To reobtain bisimilarity, P needs then be changed into

$$\textbf{repeat}\ \ x := 1\ \ \textbf{forever}\quad \|\quad \textbf{repeat}\ \ y := y+1\ \ \textbf{forever} \tag{P'}$$

The resulting labelled transition system of both P' and Q' features one state and two loop-transitions. Again one obtains two bisimilar systems of which only one has the liveness property that x will be 1.

I have presented this example in pseudocode to stress that the problem is not specific to process algebras. However, it can also be expressed in process algebras like CCS. Let Q for instance be the process $Q := (X|\bar{b})\backslash b$ discussed on page 4, and let $P = (Y|\tau)$ with $Y \stackrel{def}{=} a.Y$ be the parallel composition of Alice and Cataline discussed on page 7, but with cr rendered as τ. Both systems can be represented by the labelled transition system drawn on page 4. Hence they are strongly bisimilar. Yet, when assuming justness but not fairness, P has the liveness property that the second state will be reached, whereas Q does not.

6 A research agenda

This brings me to my research agenda in this matter: the development of a theory of concurrency that is equipped to ensure liveness properties of distributed systems, incorporating justness assumptions as explained above, but without making fairness assumptions. This theory should encompass process algebra, temporal logic, Petri nets and other semantic models, as well as treatments of real-time, and of the interaction between probabilistic and nondeterministic choice.

Since this involves distinguishing programs that are strongly bisimilar, it requires a complete overhaul of the basic machinery that has been built in the last few decennia. It requires new equivalence relations between processes, new axiomatisations, new induction principles to reason about infinite processes, new congruence formats for operational semantics ensuring compositionality of operators, and matching extensions with time and probabilities.

As in the absence of fairness assumptions some crucial systems like fair schedulers or mutual exclusion protocols cannot be accurately specified in Petri nets or standard process algebras [Vog02, KW97, GH15a], it also involves the study of adequate model or language extensions, and their expressive power.

My agenda furthermore aims at developing a methodology that allows successful application of the envisioned theory of concurrency to the specification, analysis and verification of realistic distributed systems, focusing on cases where the new balance in establishing liveness properties bears fruit. An example of this is the analysis of routing protocols in wireless networks, where the packet delivery property from Section 2 can hold in a meaningful way only when assuming justness but not full fairness.

This research agenda involves the following tasks:

1. Setting up a framework for modelling specifications and implementations of distributed systems that encompasses justness without making global fairness assumptions.
2. To investigate and classify semantic equivalences (necessarily finer than or incomparable with strong bisimilarity) that respect liveness when assuming justness but not fairness.
3. To study liveness and justness properties in non-interleaved semantic models like Petri nets, event structures and higher dimensional automata.
4. To find complete axiomatisations and adequate induction principles for process algebras with justness.
5. To find syntactic requirements for the operational specification of operators that guarantee that relevant justness-preserving equivalences are congruences.
6. To study the necessary extensions to process algebras or Petri nets to model simple systems like fair schedulers, and investigate the relative expressiveness of process algebras with and without them.
7. To re-evaluate the possibility and impossibility results for encoding synchrony in asynchrony when insisting that justness properties are preserved.
8. To extend relevant justness-preserving formalisms with treatments of real-time.
9. To adapt the existing testing theory for nondeterministic probabilistic processes to a setting where justness is preserved.
10. To apply the obtained formalisms to (dis)prove liveness properties for real distributed systems.

Below I describe these tasks in more detail.

Task 1: A framework for modelling distributed systems

Process algebra remains my favourite framework for modelling specifications and implementations of distributed systems. When the aim is to establish liveness properties, the semantics of process-algebraic specification languages should come with a precise definition of which runs count as just. The paper [GH18] provides a general definition of justness, which is applied to CCS and several of its extensions in [Gla19a]. It appears that the same style of definition can easily be applied to process algebras involving a CSP-style communication mechanism [Hoa85, Ros97] or name-binding [Mil99, SW01], as well as extensions of traditional process algebras with data [GM14]. Of course this needs to be checked carefully. For some less usual process algebras it is not yet clear how to formalise the concept of justness. This applies in particular to process algebras with a priority mechanism [CLN01].

Open Problem 1 *Formally define justness for process algebras with priorities.*

As pointed out in Section 1.3, for some applications it is warranted to make a fairness assumption. In [GH18] we make a distinction between *local* and *global* fairness assumptions. The latter apply globally to all scheduling problems of a given kind that appear in a distributed system; these are the kind of fairness assumptions classified in [GH18] and in Section 4 of the present paper. A *local* fairness assumption, on the other hand, can be justified for a particular scheduling problem in a system under consideration, and does not apply automatically to other scheduling problems of the same kind. In general I think it is warranted to employ local fairness assumptions in specific circumstances, on top of a global justness assumption. A good way to formalise this is to use specifications of distributed systems of the form $(P,\mathscr{F})$, where P is an expression in a suitable process algebra, whose semantics might consist of a labelled transition system, equipped with a classification of some of its runs as just, and $\mathscr{F}$ is a *fairness specification*, ruling out some of the runs as unfair. The fairness specification can for instance be given as a collection of formulas in a temporal logic, each formalising a local fairness assumption. The runs of the system that count for validating liveness properties then are those that are both just and not ruled out by $\mathscr{F}$. This mode of specification is discussed in greater detail in [GH15b], where also a consistency criterion on tuples $(P, \mathscr{F})$ is formulated. It has been used earlier in the specification language TLA$^+$ [Lam02]. It also is the type of specification applied in [FGH$^+$13, Section 9] for the formal specification of the Ad hoc On-demand Distance Vector (AODV) protocol [PBD03], a wireless mesh network routing protocol, using the process algebra AWN [FGH$^+$12] for the first part and LTL [Pnu77] for the second. Interestingly, the local fairness assumptions we needed in that work can be positioned strictly between strong and weak fairness. I conjecture that this will be the case for many applications. In [GH18, Section 7] this form of fairness was formalised in a general setting, and called *strong weak fairness*.

Task 2: A classification of semantic equivalences and preorders

A crucial prerequisite for verifying that an implementation meets a specification is a definition of what this means. Such a definition can be given in the form of an equivalence relation or preorder on a space of models that can be used to describe both specifications and implementations. For sequential systems, an overview of suitable preorders and equivalence relations defined on labelled transition systems is given in [Gla01, Gla93]. Preorders and equivalences specifically tailored to preserve safety and liveness properties are explored in [Gla10]. Equivalences for non-sequential systems are discussed, e.g., in [GG01]. They include *interleaving equivalences*, in which parallelism is equated with arbitrary interleaving, as well as equivalence notions that take, to some degree, concurrency explicitly into account. In [Gla15] I show that none of these equivalences respect *inevitability* [MOP89]: when assuming justness but not fairness, they all equate systems of which only one has the property that all its runs reach a specific success state. Hence, none of these equivalences are suitable for a process-algebraic framework destined to establish liveness properties under the justness assumption advocated above.

Open Problem 2 *Find and classify suitable semantic equivalences that respect liveness when assuming justness but not fairness.*

As shown in [Gla10], safety and liveness properties are intimately linked with the notions of may- and must-testing of De Nicola & Hennessy [DH84]. However, [Gla10] also treats *conditional liveness* properties that surpass the power of must-testing. In [Gla19b] I propose a notion of *reward testing*, and show that it matches with conditional liveness properties. Similar testing frameworks can be applied to derive preorders for concurrent processes that respect (conditional) liveness properties in the presence of the justness assumption. This may yield a result similar to the fair failure preorder of [Vog02].

Additionally, variants of most existing preorders and equivalences may be found that respect liveness under justness assumptions. Strong bisimulation, for instance, induces a relation between the runs of two bisimilar systems.[5] A bisimulation may be called *justness preserving* if it relates just runs with just runs only. Now justness-preserving strong bisimilarity is a finer variant of strong bisimilarity that respects liveness when assuming justness but not fairness.

Possibly, just forcing an existing preorder to respect liveness by adding appropriate clauses to its definition as sketched above gives a result that is less suitable for verification tasks. The resulting relation may be hard to decide, and may fail to satisfy the *recursive specification principle* (RSP) of [BK86a, BBK87, GV93], saying that guarded recursive specifications have unique solutions. This principle plays a central rôle in verification by means of equivalence checking [Bae90, GV93, GM14]. To sketch the problem, consider the processes P and Q of Section 5. In strong bisimulation semantics these processes are identified, and this can be shown by means of RSP. For consider the system of the following two

[5] One could simply say that two paths π_1 and π_2 are related iff for all n the n^{th} state of π_1 is related to the n^{th} state of π_2, and the actions between states are the same too. An alternative is to relate fewer paths, namely only those that are matched by a strategy for the bisimulation game as proposed in [HR00]—they did this to capture fairness rather than justness.

equations:

$$U = (y := y+1).U + (x := 1).V \qquad V = (y := y+1).V$$

where $(y := y+1)$ and $(x := 1)$ are simply treated as atomic actions. This system of equations is *guarded*, as defined in [Mil89, BK86a, BBK87]. Moreover, P is a solution for this system of equations up to strong bisimilarity, in the sense that if one substitutes P for the first process variable U and something suitable for the variable V, one obtains two statements that hold when interpreting $=$ as strong bisimilarity. Similarly, also Q is a solution. Based on this, RSP tells that P and Q are strongly bisimilar with each other. This is one of the standard ways to show the semantic equivalence of specifications and implementations. Now when moving from ordinary to justness-preserving strong bisimilarity, the processes P and Q are no longer equivalent. For a bisimulation would relate the run of P that forever takes the left $y := y+1$-loop, with the corresponding run of Q, and thus relates an unjust run to a just one. Nevertheless both P and Q turn out to be solutions to the above system of equations even up to justness-preserving strong bisimilarity. So applying RSP would yield the wrong conclusion that P and Q are equivalent. It follows that RSP, while sound for strong bisimilarity, is not sound for justness-preserving strong bisimilarity. For the same reasons it appears to be unsound for the fair failure preorder alluded to above. This robs us of a valuable verification tool.

This problem might be addressed by formulating more discriminating preorders and equivalences that do not feature explicit conditions on infinite runs, yet respect liveness properties. An idea might be a version of strong bisimilarity that also takes the component labels of Appendix A into account, rather than merely the action labels. While such an equivalence surely preserves justness, it may be considered too fine, in that it distinguishes processes that for all practical purposes should be identified. Examples are $P|Q \neq Q|P$ and $P|\mathbf{0} \neq P$. A suitable semantic equivalence would be less discriminating than that, but, in some aspects more discriminating than justness-preserving bisimilarity.

Another equivalence that respects liveness when assuming justness but not fairness is the *structure preserving bisimilarity* of [Gla15]. That equivalence is most likely also too discriminating for many verification tasks, so more research is called for. Location based equivalences [BCHK94] might be of use here.

Task 3: Petri nets and other semantic models

The standard semantics of process algebras is in terms of labelled transition systems. However, for accurately capturing causalities between event occurrences, models like Petri nets [Rei13], event structures [Win87] or higher dimensional automata [Pra91, Gla91] are sometimes preferable. As shown above, unaugmented labelled transition systems are not sufficient to capture liveness properties when assuming justness but not fairness. On the other hand, Petri nets naturally offer a structural characterisation of justness: if a transition is enabled, and none of the tokens enabling it are ever consumed by a competing transition, then it will eventually fire.

Open Problem 3 *Is the structural characterisation of justness from Petri nets consistent with the characterisations of justness for process algebras from [GH15b, GH18, Gla19a]?*

To be precise, a process-algebraic expression P is translated into a Petri net $[\![P]\!]_{\mathrm{PN}}$ through the standard Petri nets semantics of [GM84, Win84, GV87, Old91]. The just runs of $[\![P]\!]_{\mathrm{PN}}$ are structurally determined. So $[\![P]\!]_{\mathrm{PN}}$ translates further to labelled transition system $[\![[\![P]\!]_{\mathrm{PN}}]\!]_{\mathrm{LTS}}$ in which some of the runs are marked as just. On the other hand, using for instance the component-enriched structural operational semantics of Appendix A, P translates directly into such a labelled transition system $[\![P]\!]_{\mathrm{LTS}}$. The question now is whether $[\![[\![P]\!]_{\mathrm{PN}}]\!]_{\mathrm{LTS}}$ is semantically equivalent to $[\![P]\!]_{\mathrm{LTS}}$. Of course this question can be addressed only when a suitable equivalence has been chosen.

Another interesting question is whether event structures or higher dimensional automata also offer structural characterisations of justness.

Task 4: Complete axiomatisations and induction principles

Many process-algebraic verifications [Bae90] employ principles like the *recursive specification principle* (described in Task 2) and the *approximation induction principle* [BK86a, BBK87], allowing to derive properties of infinite systems through analysis of their finite approximations. As argued above, it is likely that these principles do not hold in straightforward variants of existing semantic equivalences that respect liveness when assuming justness but not fairness. The two ways to cope with that are (1) searching for finer equivalences that do not have this shortcoming, or (2) searching for alternative induction principles that hold and are useful in verification. At this time I cannot say which of these directions is the most promising; more research is in order here.

Algebraic laws have also shown their use in verification, and the isolation of a complete collection of such laws is often the starting point of both a good verification toolset and a better understanding of the semantic concepts involved. For these reasons, finding complete axiomatisations of suitable equivalences to deal with justness and liveness is an important task.

Open Problem 4 *Find complete axiomatisations and useful induction principles for suitable equivalences that respect liveness when assuming justness but not fairness.*

Task 5: Congruence formats for structural operational semantics

In process-algebraic verification it is essential that composition operators on processes, such as the parallel composition, are *compositional* w.r.t. the semantic equivalence employed. This means that the composition of two processes, each given as an equivalence class of, say, labelled transition systems, is independent on the choice of representatives within these equivalence classes. Compositionality of an operator w.r.t. an equivalence is the same as the equivalence being a congruence w.r.t. the operator.

Starting with [Sim85, BIM95, GV92], the most elegant and efficient way to establish compositionality results in process algebra is by means of *congruence formats*, sets of syntactic restrictions on the operational specification of the behaviour of composition operators (i.e. on rules like the ones of Table 1) that ensure compositionality. This line of research is continued in [Gro93, BG96, Uli92, Ver95, Blo95, Uli00, UP02, UY00, Fok00b, BFG04, Gla11b, GMR06, FGW12, Gla17].

Open Problem 5 *Find congruence formats tailored to the equivalences produced by Task 2.*

Task 6: Expressiveness

In the absence of fairness assumptions even simple systems like fair schedulers or mutual exclusion protocols cannot be accurately specified in standard process algebras like CCS or in Petri nets. This is shown in [Vog02, KW97, GH15a]. However, these systems *can* be accurately specified in process algebras that feature *non-blocking reading* [CDV09b, BCC$^+$11], i.e. where one can model write actions to a shared memory that can not be blocked/delayed by read actions to that memory. Such process algebras include an extension of PAFAS [CDV09b], as well as extensions of CCS with broadcast communication [GH15b], priorities [GH15a] or signals [DGH17].

When assuming fairness, fair schedulers or mutual exclusion protocols can be correctly specified in CCS [CDV09a]. But as argued in Section 2, a fairness assumption is not warranted here. Fair schedulers, in particular, are used to implement fairness assumptions made in specifications of systems; assuming fairness to show that a fair scheduler operates as intended totally defeats this purpose.

When assuming only progress and not justness, there is no hope of ever specifying a correct fair scheduler or mutual exclusion protocol, not even in the mentioned extensions of CCS. It is for this reason that this problem falls within the scope of the present paper.

Since there are at least three extensions of CCS that are expressive enough to model fair schedulers and mutual exclusion protocols, while CCS itself lacks the required expressiveness, the relative expressiveness of these extensions is an interesting question. It is an issue that needs to be studied along with other arguments for one specification formalism over another.

Open problem 6a Compare the relative expressiveness of the process algebras that can capture mutual exclusion.

The resulting study on the relative expressiveness of process algebras ought to be placed within the formal frameworks developed for comparisons of expressive power provided in [Bou85, Gor10, Gla12]. That is, to show that a specification formalism B is at least as expressive as a specification formalism A, a *valid encoding* from A into B needs to be presented. To show that B has not all the expressiveness of A one shows that no such encoding exists. Here a valid encoding from A into B is defined as a translation that satisfies some properties. In the work of Gorla [Gor10] this amounts to five correctness criteria on the translation. In [Bou85, Gla12] on the other hand, it is required that for any specification P of a system in formalism A, the translation of P into formalism B is semantically equivalent to P. This form of validity is parametrised by the choice of a semantic equivalence that spans the semantic domains in which the languages A and B are interpreted.

When comparing languages like CCS and its extensions mentioned above, preservation of properties like justness is essential. So for such purposes, another criterion needs to be added to the definition of a valid translation in the sense of [Gor10], namely a criterion that guarantees that a system P and its translation have the same liveness properties when assuming justness and not fairness. When working with valid encodings as in [Bou85, Gla12], the semantic equivalence that is chosen as parameter in the comparison should be one as found under Task 2.

In fact, when sharpening the concept of a valid encoding in this way, many relative expressiveness results established in the literature need to be reconfirmed or sharpened as well.

Open problem 6b Recast relative expressiveness results from the literature in a justness-respecting framework.

In [Sim85, Gla94] for instance, results are obtained saying that all languages with a structural operational semantics of a certain form can be translated into versions of the process algebras MEIJE [AB84] and ACP [BK86a], respectively. These translations are correct up to strong bisimilarity. An interesting question is what happens to such results when strengthening the strong bisimilarity in a justness-preserving way. Additionally, one may wonder if some process algebra upgraded with signals, broadcast communication or priorities may play a similar unifying rôle in a justness-preserving setting.

Open problem 6c Find a simple process algebra in the style of MEIJE such that all common process algebras can be translated in to it, where the translation preserves strong bisimilarity as well as liveness properties when assuming justness but not fairness.

The expressiveness of models of concurrency like Petri nets, event structures and higher dimensional automata, is also an interesting problem. The extensions of CCS with broadcast communication, priorities or signals resist translation into the default incarnations of these models. CCS with signals appears to be expressible into Petri nets extended with read arcs however [Vog02].

Open problem 6d Find extensions of the standard models of Petri nets, event structures and higher dimensional automata that capture broadcast communication, priorities and/or signals. If possible, show that the resulting models are "fully expressive" in some sense.

Task 7: Asynchronous interaction in distributed systems

In [GGS13], a precise characterisation is given of those distributed systems, modelled as structural conflict nets, a large class of Petri nets, that can be implemented without using synchronous communication. This is part of research aiming to determine to what extent synchronous communication can be simulated by asynchronous communication. In this work an original net and its asynchronous implementation are compared by means of a semantic equivalence that takes divergence, branching time and causality to some extent into account. It turned out that the result was to a large extent independent on the precise choice such an equivalence, i.e. on the degree to which it takes divergence, branching time and causality into account. This result needs to be revisited when using semantic equivalences with the appropriate respect for justness. It would be interesting to see if this changes the class of distributed systems that have asynchronous implementations.

Open Problem 7 *Characterise the class of structural conflict nets that have asynchronous implementations under an equivalence that not only takes divergence, branching time and causality to some extent into account, but also respects liveness when assuming justness but not fairness.*

Task 8: Real-time

When using a process algebra that takes time explicitly into account, justness may be obtained as a derived concept: a run is just iff time grows unboundedly.[6] As a consequence, extra structure to model justness may not be needed. Naturally, one may wonder if a given untimed semantics of a process algebra featuring justness can be obtained through an extension of the model with time; and vice versa how a notion of a just run obtained from a given timed process algebra can be characterised when abstracting from time. A version of the first question has already been addressed in [CDV06a] in the context of the process algebra PAFAS.

Open Problem 8 *Establish the relationship between notions of justness defined explicitly in untimed process algebra, and those derived from everlasting runs in timed extensions.*

Task 9: Extensions with probabilistic choice

Many semantic equivalences for distributed systems have been extended to a setting featuring both nondeterminism and probabilistic choice. Prominent examples are the probabilistic bisimulation equivalences of [Seg96], and the may- and must-testing equivalences defined in [WL92] and characterised in [DGHM08]. The latter work determines the coarsest semantic equivalences for probabilistic processes that respect safety and liveness properties, respectively, when assuming progress but not justness. A natural task is to redo this work when assuming justness.

Open Problem 9 *Characterise the coarsest semantic equivalence for nondeterministic probabilistic processes that respect liveness properties when assuming justness but not fairness.*

More in general, extended the relevant semantic equivalences from Task 2 to a probabilistic setting.

Task 10: Applications

Having argued that the work outlined above is a necessary step towards formalising liveness properties for realistic distributed systems, naturally one would like to see applications of such liveness properties, including proofs that they do or do not hold, for cases where fairness assumptions are truly unwarranted,

[6] A link between fair runs and runs where time grows unboundedly was made in [Lyn96].

and merely assuming progress is insufficient. One such application concerns the packet delivery property for routing protocols in wireless networks. I have indicated in Section 2 that assuming fairness allows one to establish versions of this property that do not hold in reality. On the other hand, without assuming justness, no useful packet delivery property will ever hold. Assume a scenario where four network nodes are active, with nodes 1 and 2 within transition range of each other, and outside transmission range of nodes 3 and 4. The packet delivery property says that an attempt of node 1 to deliver a message to node 2 ought to succeed. Yet, there exists an infinite run in which the message of 1 to 2 is never sent, let alone received, because all that occurs is an infinite sequence of chatter between nodes 3 and 4. In fact, attempts to properly formalise packet delivery [FGH$^+$13] were one of the reasons to formalise the notion of justness in [GH15b, GH18].

Open Problem 10 *Establishing packet delivery for suitable routing protocols for wireless networks.*

7 Conclusion

I have presented a research agenda aiming at laying the foundations of a theory of concurrency that is equipped to ensure liveness properties of distributed systems without making fairness assumptions. The agenda also includes the application of this theory to the specification, analysis and verification of realistic distributed systems. I have divided this agenda into 10 tasks, each of which involves solving an open problem. It is my hope that this document stimulates its readership to address some of these tasks and problems.

A CCS

CCS [Mil89] is parametrised with sets $\mathscr{K}$ of *agent identifiers* and $\mathscr{A}$ of *names*; each $X \in \mathscr{K}$ comes with a defining equation $X \stackrel{def}{=} P$ with P being a CCS expression as defined below. $Act := \mathscr{A} \dot{\cup} \bar{\mathscr{A}} \dot{\cup} \{\tau\}$ is the set of *actions*, where τ is a special *internal action* and $\bar{\mathscr{A}} := \{\bar{a} \mid a \in \mathscr{A}\}$ is the set of *co-names*. Complementation is extended to $\bar{\mathscr{A}}$ by setting $\bar{\bar{a}} = a$. Below, a ranges over $\mathscr{A} \cup \bar{\mathscr{A}}$, α over Act, and X,Y over $\mathscr{K}$. A *relabelling* is a function $f : \mathscr{A} \to \mathscr{A}$; it extends to Act by $f(\bar{a}) = \overline{f(a)}$ and $f(\tau) := \tau$. The set $\mathrm{T}_{\mathrm{CCS}}$ of CCS expressions or *processes* is the smallest set including:

$\sum_{i\in I} \alpha_i.P_i$	for I an index set, $\alpha_i \in Act$ and $P_i \in \mathrm{T}_{\mathrm{CCS}}$	*guarded choice*
$P \vert Q$	for $P,Q \in \mathrm{T}_{\mathrm{CCS}}$	*parallel composition*
$P\backslash L$	for $L \subseteq \mathscr{A}$ and $P \in \mathrm{T}_{\mathrm{CCS}}$	*restriction*
$P[f]$	for f a relabelling and $P \in \mathrm{T}_{\mathrm{CCS}}$	*relabelling*
X	for $X \in \mathscr{K}$	*agent identifier*

The process $\sum_{i\in\{1,2\}} \alpha_i.P_i$ is often written as $\alpha_1.P_1 + \alpha_2.P_2$, $\sum_{i\in\{1\}} \alpha_i.P_i$ as $\alpha_1.P_1$, and $\sum_{i\in\emptyset} \alpha_i.P_i$ as $\mathbf{0}$. Moreover, one abbreviates $\alpha.\mathbf{0}$ by α, and $P\backslash\{a\}$ by $P\backslash a$. The semantics of CCS is given by the labelled transition relation $\to \subseteq \mathrm{T}_{\mathrm{CCS}} \times Act \times \mathscr{P}(\mathscr{C}) \times \mathrm{T}_{\mathrm{CCS}}$, where transitions $P \xrightarrow{\alpha, C} Q$ are derived from the rules of Table 1. The second labels $C \in \mathscr{P}(\mathscr{C})$, displayed in red, are not part of these transitions; they will be introduced, with the definition of $\mathscr{C}$, below. Such a transition indicates that process $P \in \mathrm{T}_{\mathrm{CCS}}$ can perform the action $\alpha \in Act$ and thereby transform into process $Q \in \mathrm{T}_{\mathrm{CCS}}$. The process $\sum_{i\in I} \alpha_i.P_i$ performs one of the actions α_j for $j \in I$ and subsequently acts as P_j. The parallel composition $P|Q$ executes an action from P, an action from Q, or in the case where P and Q can perform complementary actions c and $\bar{c}$, the process can perform a synchronisation, resulting in an internal action τ. The restriction operator $P\backslash L$ inhibits execution of the actions from L and their complements. The relabelling $P[f]$ acts like process P with all labels α replaced by $f(\alpha)$. Finally, the rule for agent identifiers says that an agent X has the same transitions as the body P of its defining equation. The standard version of CCS [Mil89] features a *choice* operator $\sum_{i\in I} P_i$; here I use the fragment of CCS that merely features guarded choice.

Table 1: Structural operational semantics of CCS

$$\sum_{i\in I}\alpha_i.P_i \xrightarrow{\alpha_j\,,\,\{\varepsilon\}} P_j \quad (j\in I)$$

$$\frac{P \xrightarrow{\alpha\,,\,C} P'}{P|Q \xrightarrow{\alpha\,,\,\mathrm{L}\cdot C} P'|Q} \qquad \frac{P \xrightarrow{a\,,\,C} P',\ Q \xrightarrow{\bar{a}\,,\,D} Q'}{P|Q \xrightarrow{\tau\,,\,\mathrm{L}\cdot C\,\cup\,\mathrm{R}\cdot D} P'|Q'} \qquad \frac{Q \xrightarrow{\alpha\,,\,D} Q'}{P|Q \xrightarrow{\alpha\,,\,\mathrm{R}\cdot D} P|Q'}$$

$$\frac{P \xrightarrow{\alpha\,,\,C} P'}{P\backslash L \xrightarrow{\alpha\,,\,C} P'\backslash L}\ (\alpha,\bar{\alpha}\notin L) \qquad \frac{P \xrightarrow{\alpha\,,\,C} P'}{P[f] \xrightarrow{f(\alpha)\,,\,C} P'[f]} \qquad \frac{P \xrightarrow{\alpha\,,\,C} P'}{X \xrightarrow{\alpha\,,\,C} P'}\ (X \stackrel{def}{=} P)$$

Components The second label of a transition indicates the set of (parallel) *components* involved in executing this transition. The set $\mathscr{C}$ of components is defined as $\{\mathrm{L},\mathrm{R}\}^*$, that is, set of strings over the indicators Left and Right, with $\varepsilon\in\mathscr{C}$ denoting the empty sequence and $\mathrm{D}\cdot C := \{\mathrm{D}\sigma \mid \sigma\in C\}$ for $\mathrm{D}\in\{\mathrm{L},\mathrm{R}\}$. The process Q from page 4 for instance has the transitions $Q \xrightarrow{a\,,\,\{\mathrm{L}\}} Q$ and $Q \xrightarrow{\tau\,,\,\{\mathrm{L},\mathrm{R}\}} (X|\mathbf{0})\backslash c$. The first transition denotes Alice performing *a*ny other activity; it involves the left component of the parallel composition only. The second transition denotes a call between Alice and Bob; it involves both components. The idea to extend the structural operational semantics of CCS with a component labelling as indicated in Table 1, to achieve an elegant formalisation of justness, stems from Victor Dyseryn [personal communication].

B Justness

This appendix interprets each CCS process as a state in a component-labelled transition system. This is an ordinary labelled transition system, upgraded with a labelling of the transitions t by the parallel components involved in performing t. It also enriches such systems with the set B of blocking actions. Subsequently, it presents the definitions of a path and the completeness criteria progress and justness, thereby formalising the intuitions from Section 3.

Definition 1 A *component-labelled transition system* (CLTS) is a tuple $(S,Tr,source,target,\ell,B,comp)$ with S and Tr sets (of *states* and *transitions*), $source,target : Tr\rightarrow S$, $\ell : Tr\rightarrow Act$ for a set of actions Act, $B\subseteq Act$ a set of *blocking* actions, and $comp : Tr\rightarrow\mathscr{P}(\mathscr{C})\setminus\emptyset$ for some set of components $\mathscr{C}$, such that:

$$\begin{array}{l}\text{for all } t,v\in Tr \text{ with } source(t)=source(v) \text{ and } comp(t)\cap comp(v)=\emptyset,\\ \text{there is a } u\in Tr \text{ with } source(u)=target(v),\ \ell(u)=\ell(t) \text{ and } comp(u)=comp(t).\end{array} \tag{1}$$

The underlying idea [GH18] is that if a transition v occurs that does not affect components in $comp(t)$, then the internal state of those components is unchanged, so any synchronisation between these components that was possible before v occurred, is still possible afterwards.

Let $Tr_{\neg B} := \{t\in Tr \mid \ell(t)\notin B\}$ be the set of *non-blocking* transitions.

The following CLTS serves as a semantic model for CCS: Take S to be the set $\mathrm{T}_{\mathrm{CCS}}$ of CCS processes, and Tr the set of transitions $P\xrightarrow{\alpha\,,\,C}Q$ derivable from the rules of Table 1. For each such $t\in Tr$ one takes $source(t) := P$, $target(t) := Q$, $\ell(t)=\alpha$ and $comp(t) := C$. It is straightforward to check that property (1) is satisfied (or see [Gla19a]). The set $B\subseteq Act$ can be chosen at will, depending on the intended application, as long as one promises to use only restrictions $P\backslash L$ with $L\subseteq B$ and no renamings f that rename a non-blocking action into a blocking one. So the default choice for B is $Act\setminus\{\tau\}$.

A CLTS could also have been defined as a triple (S,Tr,B), with S a set, $Tr\subseteq S\times Act\times\mathscr{P}(\mathscr{C})\times S$ for sets Act of actions and $\mathscr{C}$ of components, and $B\subseteq Act$.

Definition 2 A *path* in a CLTS $(S, Tr, source, target, \ell, B, comp)$ is an alternating sequence $s_0\, t_1\, s_1\, t_2\, s_2 \cdots$ of states and transitions, starting with a state and either being infinite or ending with a state, such that $source(t_i) = s_{i-1}$ and $target(t_i) = s_i$ for all relevant i.

A *completeness criterion* is a unary predicate on the paths in a (component-labelled) transition system.

Definition 3 A path in a CLTS is *progressing* if either it is infinite or its last state is the source of no non-blocking transition $t \in Tr_{\neg B}$.

Progress is a completeness criterion.

Definition 4 Two transitions $t, u \in Tr$ are *concurrent*, notation $t \smile u$, if $comp(t) \cap comp(u) = \emptyset$, i.e., they have no components in common.

If t and u are not concurrent, $t \not\smile u$, then u is said *to interfere with* t.

Definition 5 A path π in an CLTS is *just* if for each transition $t \in Tr_{\neg B}$ with $s := source(t) \in \pi$, a transition u occurs in π past the occurrence of s, such that $t \not\smile u$.

Informally, justness requires that once a non-blocking transition t is enabled, sooner or later a transition u will occur that interferes with it, possibly t itself.

Note that justness is a completeness criterion stronger than progress.

Historical notes and disclaimers Definition 5 of justness stems from [GH18]. There it was formulated for transition systems without the labelling function ℓ, and with an initial state $I \in S$. This makes no difference. Appendix A of [GH18] defines the function $comp : Tr \to \mathscr{P}(\mathscr{C}) \setminus \emptyset$ for a somewhat different fragment of CCS than is used here. Although that definition has a rather different style than the one of this paper, on the intersection of both fragments of CCS the resulting functions *comp* are easily seen to be the same.

The original definition of justness applied to CCS stems from [GH15b]. That (coinductive) definition is in a very different style and does not use the concepts *comp* and $\smile$. In [Gla19a] it is shown that the concept of justness from [GH15b] is the same as that from [GH18] and the present paper. Moreover, [Gla19a] contemplates 5 different concurrency relations $\smile$ between transitions for full CCS, and shows that through Definition 5 all of them give rise to the same concept of justness. The concurrency relation of Definition 4 is $\smile_c'$ in [Gla19a].

The components $comp(t)$ of a CCS transition t are necessary participants in the execution of t, in the sense that all of them should be ready in order for the transition to be enabled. They also are the components that are affected by the execution of t, in the sense that t may induce a state-change within all these components. This situation is common for many process algebras. In [Gla19a] however, extensions of CCS are reviewed in which only some components are necessary and only some are affected. Here, subsets $npc(t), afc(t) \subseteq comp(t)$ of *necessary participants* and *affected components* are defined, and Definition 4 declares $t \smile\!\!\bullet\, u$ iff $npc(t) \cap afc(u) = \emptyset$. In this setting it can be that u interferes with t (notation $t \not\smile\!\!\bullet\, u$) even though t does not interfere with u, thus giving rise to an asymmetric concurrency relation. The treatment above just deals with the special case that $npc(t) = afc(u) = comp(t)$ for all $t \in Tr$.

In [Gla19a] one encounters transition systems featuring *signal transitions* t. Such a transition satisfies $source(t) = target(t)$ and does not model an action occurrence or state change in the represented distributed system. To properly apply the Definitions 2–7 to such transition systems, one should first normalise the transition system by deleting all signal transitions [Gla19a].

In [Gla19a] the component B is absent from the definition of a transition system, meaning that actions are not a priory distinguished into blocking and non-blocking ones. Instead, the concepts of progress, justness and fairness are indexed with a B: a path is called B-just if it satisfies the definition of justness when taking B to be the set of blocking actions. This makes justness into a family of predicates on paths, rather than a single predicate.

C Fairness

To formalise weak and strong fairness I use labelled transition systems $(S, Tr, source, target, \ell, B, \mathscr{T})$ that are augmented with a set $\mathscr{T} \subseteq \mathscr{P}(Tr)$ of *tasks* $T \subseteq Tr$, each being a set of transitions. Here $S, Tr, source, target, \ell, B$ are exactly as in Definition 1 of a CLTS.

Definition 6 ([GH18]) For such a $\mathbb{T} = (S, Tr, source, target, \ell, B, \mathscr{T})$, a task $T \in \mathscr{T}$ is *enabled* in a state $s \in S$ if there exists a non-blocking transition $t \in T$ with $\ell(t) \notin B$ and $source(t) = s$. The task is said to be *perpetually enabled* on a path π in $\mathbb{T}$, if it is enabled in every state of π. It is *relentlessly enabled* on π, if each suffix of π contains a state in which it is enabled.[7] It *occurs* in π if π contains a transition $t \in T$.

A path π in $\mathbb{T}$ is *weakly fair* if, for every suffix π' of π, each task that is perpetually enabled on π', occurs in π'. A path π in $\mathbb{T}$ is *strongly fair* if, for every suffix π' of π, each task that is relentlessly enabled on π', occurs in π'.

In [GH18] many notions of fairness occurring in the literature were casts as instances of this definition. For each of them the set of tasks $\mathscr{T}$ was derived, in different ways, from some other structure present in the model of distributed systems from the literature. In fact, [GH18] considers 7 ways to construct the collection $\mathscr{T}$, and speaks of fairness of *actions*, *transitions*, *instructions*, *synchronisations*, *components*, *groups of components* and *events*. This yields 14 notions of fairness. To compare them, each is defined formally on a fragment of CCS, and the 14 fairness notions, together with progress, justness, and some other completeness criteria, are ordered by strength by placing them in a lattice.

For transition systems $(S, Tr, source, target, \ell, B, \smile, \mathscr{T})$ augmented with a concurrency relation as well as a set of tasks, the following notion of J-fairness is proposed in [GH18]:

Definition 7 ([GH18]) For $\mathbb{T} = (S, Tr, source, target, \ell, B, \smile, \mathscr{T})$, a task $T \in \mathscr{T}$ is enabled *during the execution* of a transition $u \in Tr$ if there exists a $t \in T$ with $\ell(t) \notin B$, $source(t) = source(u)$ and $t \smile u$. It is *continuously enabled* on a path π iff it is enabled in every state and during every transition of π. A path π is J-*fair* if, for every suffix π' of π, each task that is continuously enabled on π', occurs in π'.

The name J-*fairness* is inspired by the notion of *justice* from Lehmann, Pnueli & Stavi [LPS81]. They called a computation *just* "if it is finite or if every transition[8] which is continuously enabled beyond a certain point is taken infinitely many times." What this means exactly depends on how one formalises the notion of a component being "continuously enabled". The literature following [LPS81] has systematically interpreted this as meaning "in every state" ("beyond a certain point"), thereby translating it as "perpetually enabled" from Definition 6. This is consistent with the words of [LPS81], as at some point they write "[This] is an unjust computation, since [the transition or component] f_1 is enabled on all states in it but is never taken." This makes justice a form of weak fairness as in Definition 6. However, a stricter interpretation of "continuously enabled" is given in Definition 7, and this gives rise to a yet weaker notion of fairness that so far has not received much explicit attention in the literature.

I now instantiate the definitions above for a particular choice of $\mathscr{T}$, namely fairness of components [GH18]. Its definition applies to component-labelled transition systems. Under fairness of components each component determines a task. A transition belongs to that task if that component contributes to it. So $\mathscr{T} := \{T_\sigma \mid \sigma \in \mathscr{C}\}$ with $T_\sigma := \{t \in Tr \mid \sigma \in comp(t)\}$. This is the type of fairness studied in [CS87, CDV06b]; it also appears in [KR83, AFK88] under the name process fairness. Fairness of components can also be regarded as the type of fairness studied in [LPS81], although that paper does not address synchronisation, and thus deals with the special case that $comp(t)$ is a singleton for all $t \in Tr$.

[7] This is the case if the task is enabled in infinitely many states of π, in a state that occurs infinitely often in π, or in the last state of a finite π.

[8] The notion of "transition" from [LPS81] is the same as what I call "component".

Proposition 1 On any CLTS, a strongly fair path is always weakly fair, a weakly fair path is always J-fair, a J-fair path—under fairness of components—is always just, and a just path is always progressing.

Proof: W.l.o.g. let π be a J-fair path under fairness of components, such that a transition t is enabled in its first state.[9] Assume, towards a contradiction, that π contains no transition w with $comp(t) \cap comp(w) \neq \emptyset$. Let $\sigma \in comp(t)$. Then, using (1), for each transition v occurring in π, a transition u_v with $comp(u_v) = comp(t) \ni \sigma$ is enabled right after v. Note that $u_v \in T_\sigma$. So the task T_σ is enabled in each state of π. Since $u_v \smile w$ for each transition w in π, the task T_σ is also enabled during each transition of π. So, by J-fairness, the task T_σ must occur in π, contradicting the assumption. It follows that π is just.

The other three statements of Proposition 1 follow immediately from the definitions. □

In fact, all implications of Proposition 1 are strict. Counterexamples against their reverses appear in [GH18, Examples 3, 20, 12 and 21].[10]

Acknowledgement This paper benefited greatly from the insightful comments of three referees.

References

[AB84] D. Austry & G. Boudol (1984): *Algèbre de processus et synchronisations*. Theoretical Computer Science 30(1), pp. 91–131, doi:10.1016/0304-3975(84)90067-7.

[AFK88] K.R. Apt, N. Francez & S. Katz (1988): *Appraising Fairness in Languages for Distributed Programming*. Distributed Computing 2(4), pp. 226–241, doi:10.1007/BF01872848.

[Bae90] J.C.M. Baeten, editor (1990): *Applications of Process Algebra*. Cambridge Tracts in Theoretical Computer Science 17, Cambridge University Press.

[BBK87] J.C.M. Baeten, J.A. Bergstra & J.W. Klop (1987): *On the Consistency of Koomen's Fair Abstraction Rule*. Theoretical Computer Science 51(1/2), pp. 129–176, doi:10.1016/0304-3975(87)90052-1.

[BBR10] J.C.M. Baeten, T. Basten & M.A. Reniers (2010): *Process Algebra: Equational Theories of Communicating Processes*. Cambridge University Press.

[BCC+11] F. Buti, M. Callisto De Donato, F. Corradini, M.R. Di Berardini & W. Vogler (2011): *Automated Analysis of MUTEX Algorithms with FASE*. In G. D'Agostino & S. La Torre, editors: Proc. GandALF '11, EPTCS 54, Open Publishing Association, pp. 45–59, doi:10.4204/EPTCS.54.4.

[BCHK94] G. Boudol, I. Castellani, M. Hennessy & A. Kiehn (1994): *A Theory of Processes with Localities*. Formal Aspects of Computing 6(2), pp. 165–200, doi:10.1007/BF01221098.

[BDL04] G. Behrmann, A. David & K.G. Larsen (2004): *A Tutorial on Uppaal*. In M. Bernardo & F. Corradini, editors: Revised Lectures on Formal Methods for the Design of Real-Time Systems, LNCS 3185, Springer, pp. 200–236, doi:10.1007/978-3-540-30080-9_7.

[BFG04] B. Bloom, W.J. Fokkink & R.J. van Glabbeek (2004): *Precongruence Formats for Decorated Trace Semantics*. Transactions on Computational Logic 5(1), pp. 26–78, doi:10.1145/963927.963929.

[BG96] R.N. Bol & J.F. Groote (1996): *The meaning of negative premises in transition system specifications*. Journal of the ACM 43(5), pp. 863–914, doi:10.1145/234752.234756.

[9] The general case that $s = source(t)$ occurs in π can be reduced to this special case by taking the suffix of π starting at s.

[10] In [GH15b] is it shown that the concept of justice from [LPS81], when interpreting "continuously enabled" as in Definition 7, and when translating the notion of "transition" from [LPS81] by "abstract transition" for CCS, as defined in [GH15b], coincides with justness. Here an "abstract transition" is an equivalence class of transitions where two transitions are equivalent if they merely differ in the internal state of a component not involved in these transitions. The two a-labelled transitions in the CCS process $a|b$ for instance are deemed equivalent, because they differ only in the state of the b-component.

This result appears at odds with the strictness of the inclusions of Proposition 1. But it is not, for Proposition 1 applies to a translation of "transition" from [LPS81] by "component" in CCS, which upon closer inspection of [LPS81] is more accurate. A component can be seen as an even larger equivalence class of transitions than an abstract transition.

[BGRV15] J. Borgström, R. Gutkovas, I. Rodhe & B. Victor (2015): *The Psi-Calculi Workbench: A Generic Tool for Applied Process Calculi*. ACM Transactions on Embedded Computing Systems 14(1), pp. 9:1–9:25, doi:10.1145/2682570.

[BHR84] 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, doi:10.1145/828.833.

[BIM95] B. Bloom, S. Istrail & A.R. Meyer (1995): *Bisimulation Can't be Traced*. Journal of the ACM 42(1), pp. 232–268, doi:10.1145/200836.200876.

[BJPV11] J. Bengtson, M. Johansson, J. Parrow & B. Victor (2011): *Psi-calculi: a framework for mobile processes with nominal data and logic*. Logical Methods in Computer Science 7(1), doi:10.2168/LMCS-7(1:11)2011.

[BK86a] J.A. Bergstra & J.W. Klop (1986): *Algebra of communicating processes*. In J.W. de Bakker, M. Hazewinkel & J.K. Lenstra, editors: Mathematics and Computer Science, CWI Monograph 1, North-Holland, Amsterdam, pp. 89–138.

[BK86b] J.A. Bergstra & J.W. Klop (1986): *Verification of an alternating bit protocol by means of process algebra*. In W. Bibel & K.P. Jantke, editors: Mathematical Methods of Specification and Synthesis of Software Systems '85, LNCS 215, Springer, pp. 9–23, doi:10.1007/3-540-16444-8_1.

[BK08] C. Baier & J. Katoen (2008): *Principles of model checking*. MIT Press.

[Blo95] B. Bloom (1995): *Structural operational semantics for weak bisimulations*. Theoretical Computer Science 146, pp. 25–68, doi:10.1016/0304-3975(94)00152-9.

[Bou85] G. Boudol (1985): *Notes on algebraic calculi of processes*. In K. Apt, editor: Logics and Models of Concurrent Systems, NATO ASI Series F13, Springer, pp. 261–303, doi:10.1007/978-3-642-82453-1_9.

[BRV95] E. Brinksma, A. Rensink & W. Vogler (1995): *Fair Testing*. In I. Lee & S. Smolka, editors: Proceedings 6th International Conference on Concurrency Theory (CONCUR'95), LNCS 962, Springer, pp. 313–327, doi:10.1007/3-540-60218-6_23.

[BRV96] E. Brinksma, A. Rensink & W. Vogler (1996): *Applications of Fair Testing*. In R. Gotzhein & J. Bredereke, editors: Proceedings IFIP TC6 WG6.1 International Conference on Formal Description Techniques IX: Theory, application and tools / Protocol Specification, Testing and Verification XVI, IFIP Conference Proceedings 69, Chapman & Hall, pp. 145–160.

[BSW69] K.A. Bartlett, R.A. Scantlebury & P.T. Wilkinson (1969): *A note on reliable full-duplex transmission over half-duplex links*. CACM 12, pp. 260–261, doi:10.1145/362946.362970.

[BW90] J.C.M. Baeten & W.P. Weijland (1990): *Process Algebra*. Cambridge Tracts in Theoretical Computer Science 18, Cambridge University Press, doi:10.1017/CBO9780511624193.

[CDV06a] F. Corradini, M.R. Di Berardini & W. Vogler (2006): *Fairness of Actions in System Computations*. Acta Informatica 43(2), pp. 73–130, doi:10.1007/s00236-006-0011-2.

[CDV06b] F. Corradini, M.R. Di Berardini & W. Vogler (2006): *Fairness of Components in System Computations*. Theoretical Computer Science 356(3), pp. 291–324, doi:10.1016/j.tcs.2006.02.011.

[CDV09a] F. Corradini, M.R. Di Berardini & W. Vogler (2009): *Liveness of a Mutex Algorithm in a Fair Process Algebra*. Acta Informatica 46(3), pp. 209–235, doi:10.1007/s00236-009-0092-9.

[CDV09b] F. Corradini, M.R. Di Berardini & W. Vogler (2009): *Time and Fairness in a Process Algebra with Non-blocking Reading*. In M. Nielsen, A. Kucera, P.B. Miltersen, C. Palamidessi, P. Tuma & F.D. Valencia, editors: Theory and Practice of Computer Science (SOFSEM'09), LNCS 5404, Springer, pp. 193–204, doi:10.1007/978-3-540-95891-8_20.

[CLN01] R. Cleaveland, G. Lüttgen & V. Natarajan (2001): *Priority in Process Algebra*. In J.A. Bergstra, A. Ponse & S.A. Smolka, editors: Handbook of Process Algebra, chapter 12, Elsevier, pp. 711–765, doi:10.1016/B978-044482830-9/50030-8.

[CS84] G. Costa & C. Stirling (1984): *A Fair Calculus of Communicating Systems*. Acta Informatica 21, pp. 417–441, doi:10.1007/BF00271640.

[CS87] G. Costa & C. Stirling (1987): *Weak and Strong Fairness in CCS*. Information and Computation 73(3), pp. 207–244, doi:10.1016/0890-5401(87)90013-7.

[CS01] R. Cleaveland & O. Sokolsky (2001): *Equivalence and Preorder Checking for Finite-State Systems*. In J.A. Bergstra, A. Ponse & S.A. Smolka, editors: Handbook of Process Algebra, chapter 6, Elsevier, pp. 391–424, doi:10.1016/B978-044482830-9/50024-2.

[DGH17] V. Dyseryn, R.J. van Glabbeek & P. Höfner (2017): *Analysing Mutual Exclusion using Process Algebra with Signals*. In K. Peters & S. Tini, editors: Proceedings Combined 24th International Workshop on Expressiveness in Concurrency and 14th Workshop on Structural Operational Semantics, EPTCS 255, Open Publishing Association, pp. 18–34, doi:10.4204/EPTCS.255.2.

[DGHM08] Y. Deng, R.J. van Glabbeek, M. Hennessy & C.C. Morgan (2008): *Characterising Testing Preorders for Finite Probabilistic Processes*. Logical Methods in Computer Science 4(4):4, doi:10.2168/LMCS-4(4:4)2008.

[DH84] R. De Nicola & M. Hennessy (1984): *Testing equivalences for processes*. Theoretical Computer Science 34, pp. 83–133, doi:10.1016/0304-3975(84)90113-0.

[EC82] E.A. Emerson & E.M. Clarke (1982): *Using Branching Time Temporal Logic to Synthesize Synchronization Skeletons*. Science of Computer Programming 2(3), pp. 241–266, doi:10.1016/0167-6423(83)90017-5.

[FGH+12] A. Fehnker, R.J. van Glabbeek, P. Höfner, A.K. McIver, M. Portmann & W. Tan (2012): *A Process Algebra for Wireless Mesh Networks*. In H. Seidl, editor: Programming Languages and Systems: Proceedings 21st European Symposium on Programming (ESOP'12), LNCS 7211, Springer, pp. 295–315, doi:10.1007/978-3-642-28869-2_15.

[FGH+13] A. Fehnker, R.J. van Glabbeek, P. Höfner, A.K. McIver, M. Portmann & W.L. Tan (2013): *A Process Algebra for Wireless Mesh Networks used for Modelling, Verifying and Analysing AODV*. Technical Report 5513, NICTA. Available at http://arxiv.org/abs/1312.7645.

[FGW12] W.J. Fokkink, R.J. van Glabbeek & P. de Wind (2012): *Divide and congruence: From decomposition of modal formulas to preservation of branching and η-bisimilarity*. Information and Computation 214, pp. 59–85, doi:10.1016/j.ic.2011.10.011.

[Fok00a] W.J. Fokkink (2000): *Introduction to Process Algebra*. Texts in Theoretical Computer Science, An EATCS Series, Springer, doi:10.1007/978-3-662-04293-9.

[Fok00b] W.J. Fokkink (2000): *Rooted branching bisimulation as a congruence*. Journal of Computer and System Sciences 60(1), pp. 11–27, doi:10.1006/jcss.1999.1663.

[GABR14] T. Gibson-Robinson, P.J. Armstrong, A. Boulgakov & A.W. Roscoe (2014): *FDR3 - A Modern Refinement Checker for CSP*. In E. Ábrahám & K. Havelund, editors: Proceedings 20th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'14), LNCS 8413, Springer, pp. 187–201, doi:10.1007/978-3-642-54862-8_13.

[GG01] R.J. van Glabbeek & U. Goltz (2001): *Refinement of Actions and Equivalence Notions for Concurrent Systems*. Acta Informatica 37, pp. 229–327, doi:10.1007/s002360000041.

[GGS13] R.J. van Glabbeek, U. Goltz & J.-W. Schicke-Uffmann (2013): *On Characterising Distributability*. Logical Methods in Computer Science 9(3):17, doi:10.2168/LMCS-9(3:17)2013.

[GH15a] R.J. van Glabbeek & P. Höfner (2015): *CCS: It's not fair! Fair schedulers cannot be implemented in CCS-like languages even under progress and certain fairness assumptions*. Acta Informatica 52(2-3), pp. 175–205, doi:10.1007/s00236-015-0221-6.

[GH15b] R.J. van Glabbeek & P. Höfner (2015): *Progress, Fairness and Justness in Process Algebra*. Technical Report 8501, NICTA, Sydney, Australia. Available at http://arxiv.org/abs/1501.03268.

[GH18] R.J. van Glabbeek & P. Höfner (2018): *Progress, Justness and Fairness*. Survey paper, Data61, CSIRO, Sydney, Australia. Available at https://arxiv.org/abs/1810.07414. To appear in ACM Computing Surveys.

[Gla91] R.J. van Glabbeek (1991): *Bisimulations for higher dimensional automata*. Email message, July 7, 1991. Available at http://theory.stanford.edu/~rvg/hda.

[Gla93] R.J. van Glabbeek (1993): *The Linear Time – Branching Time Spectrum II; The semantics of sequential systems with silent moves (extended abstract)*. In E. Best, editor: Proceedings *CONCUR'93*, 4th International Conference on *Concurrency Theory*, LNCS 715, Springer, pp. 66–81, doi:10.1007/3-540-57208-2_6.

[Gla94] R.J. van Glabbeek (1994): *On the expressiveness of ACP (extended abstract)*. In A. Ponse, C. Verhoef & S.F.M. van Vlijmen, editors: Proceedings First Workshop on the *Algebra of Communicating Processes* (ACP'94), Workshops in Computing, Springer, pp. 188–217, doi:10.1007/978-1-4471-2120-6_8.

[Gla01] R.J. van Glabbeek (2001): *The Linear Time – Branching Time Spectrum I; The Semantics of Concrete, Sequential Processes*. In J.A. Bergstra, A. Ponse & S.A. Smolka, editors: *Handbook of Process Algebra*, chapter 1, Elsevier, pp. 3–99, doi:10.1016/B978-044482830-9/50019-9.

[Gla10] R.J. van Glabbeek (2010): *The Coarsest Precongruences Respecting Safety and Liveness Properties*. In C.S. Calude & V. Sassone, editors: Proceedings 6th IFIP TC 1/WG 2.2 International Conference on *Theoretical Computer Science* (TCS 2010), held as part of the *World Computer Congress* 2010, *IFIP* 323, Springer, pp. 32–52, doi:10.1007/978-3-642-15240-5_3.

[Gla11a] R.J. van Glabbeek (2011): *Bisimulation*. In D. Padua, editor: *Encyclopedia of Parallel Computing*, Springer, pp. 136–139, doi:10.1007/978-0-387-09766-4_149.

[Gla11b] R.J. van Glabbeek (2011): *On Cool Congruence Formats for Weak Bisimulations*. *Theoretical Computer Science* 412(28), pp. 3283–3302, doi:10.1016/j.tcs.2011.02.036.

[Gla12] R.J. van Glabbeek (2012): *Musings on Encodings and Expressiveness*. In B. Luttik & M.A. Reniers, editors: Proceedings EXPRESS/SOS'19, EPTCS 89, Open Publishing Association, pp. 81–98, doi:10.4204/EPTCS.89.7.

[Gla15] R.J. van Glabbeek (2015): *Structure Preserving Bisimilarity, Supporting an Operational Petri Net Semantics of CCSP*. In R. Meyer, A. Platzer & H. Wehrheim, editors: Proceedings *Correct System Design - Symposium in Honor of Ernst-Rüdiger Olderog on the Occasion of His 60th Birthday*, LNCS 9360, Springer, pp. 99–130, doi:10.1007/978-3-319-23506-6_9.

[Gla17] R.J. van Glabbeek (2017): *Lean and Full Congruence Formats for Recursion*. In: Proceedings 32nd Annual ACM/IEEE Symposium on *Logic in Computer Science*, LICS 2017, IEEE Computer Society Press, doi:10.1109/LICS.2017.8005142.

[Gla19a] R.J. van Glabbeek (2019): *Justness: A Completeness Criterion for Capturing Liveness Properties*. Technical Report, Data61, CSIRO, Sydney, Australia. Available at http://theory.stanford.edu/~rvg/abstracts.html#133. Extended abstract in M. Bojaczyk & A. Simpson, eds.: Proc. FoSSaCS'19, LNCS 11425, Springer, pp.505-522, doi:10.1007/978-3-030-17127-8_29.

[Gla19b] R.J. van Glabbeek (2019): *Reward Testing Equivalences for Processes*. In: M. Boreale, F. Corradini, M. Loreti & R. Pugliese, editors: *Models, Languages, and Tools for Concurrent and Distributed Programming, Essays Dedicated to Rocco De Nicola on the Occasion of His 65th Birthday*, LNCS 11665, Springer, pp. 45–70, doi:10.1007/978-3-030-21485-2_5.

[GLMS11] H. Garavel, F. Lang, R. Mateescu & W. Serwe (2011): *CADP 2010: A Toolbox for the Construction and Analysis of Distributed Processes*. In P.A. Abdulla & K.R.M. Leino, editors: Proceedings 17th International Conference on *Tools and Algorithms for the Construction and Analysis of Systems* (TACAS'11), LNCS 6605, Springer, pp. 372–387, doi:10.1007/978-3-642-19835-9_33.

[GLT09] R.J. van Glabbeek, B. Luttik & N. Trčka (2009): *Branching Bisimilarity with Explicit Divergence*. *Fundamenta Informaticae* 93(4), pp. 371–392, doi:10.3233/FI-2009-109.

[GM84] U. Goltz & A. Mycroft (1984): *On the relationship of CCS and Petri nets*. In J. Paredaens, editor: *Proceedings 11th ICALP*, LNCS 172, Springer, pp. 196–208, doi:10.1007/3-540-13345-3_18.

[GM14] J.F. Groote & M.R. Mousavi (2014): *Modeling and Analysis of Communicating Systems*. MIT Press.

[GMR06] J.F. Groote, M.R. Mousavi & M.A. Reniers (2006): *A Hierarchy of SOS Rule Formats*. *Electronic Notes in Theoretical Computer Science* 156(1), pp. 3–25, doi:10.1016/j.entcs.2005.11.077.

[Gor10] D. Gorla (2010): *Towards a unified approach to encodability and separation results for process calculi*. *Information and Computation* 208(9), pp. 1031–1053, doi:10.1016/j.ic.2010.05.002.

[Gro93] J.F. Groote (1993): *Transition System Specifications with Negative Premises*. Theoretical Computer Science 118(2), pp. 263–299, doi:10.1016/0304-3975(93)90111-6.

[GV87] R.J. van Glabbeek & F.W. Vaandrager (1987): *Petri net models for algebraic theories of concurrency (extended abstract)*. In J.W. de Bakker, A.J. Nijman & P.C. Treleaven, editors: Proceedings PARLE, Parallel Architectures and Languages Europe, Vol. II: Parallel Languages, LNCS 259, Springer, pp. 224–242, doi:10.1007/3-540-17945-3_13.

[GV92] J.F. Groote & F.W. Vaandrager (1992): *Structured Operational Semantics and Bisimulation as a Congruence*. Information and Comp. 100(2), pp. 202–260, doi:10.1016/0890-5401(92)90013-6.

[GV93] R.J. van Glabbeek & F.W. Vaandrager (1993): *Modular Specification of Process Algebras*. Theoretical Computer Science 113(2), pp. 293–348, doi:10.1016/0304-3975(93)90006-F.

[GW96] R.J. van Glabbeek & W.P. Weijland (1996): *Branching Time and Abstraction in Bisimulation Semantics*. Journal of the ACM 43(3), pp. 555–600, doi:10.1145/233551.233556. Available in part at http://Theory.Stanford.EDU/~rvg/abstraction/.

[Hoa85] C.A.R. Hoare (1985): *Communicating Sequential Processes*. Prentice-Hall, Englewood Cliffs.

[Hol04] G.J. Holzmann (2004): *The SPIN Model Checker - primer and reference manual*. Addison-Wesley.

[HR00] T.A. Henzinger & S.K. Rajamani (2000): *Fair Bisimulation*. In S. Graf & M.I. Schwartzbach, editors: Proceedings 6th International Conference on Tools and Algorithms for Construction and Analysis of Systems (TACAS'00), LNCS 1785, Springer, pp. 299–314, doi:10.1007/3-540-46419-0_21.

[KLM+15] G. Kant, A. Laarman, J. Meijer, J. van de Pol, S. Blom & T. van Dijk (2015): *LTSmin: High-Performance Language-Independent Model Checking*. In C. Baier & C. Tinelli, editors: Proceedings 21st International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'15), LNCS 9035, Springer, pp. 692–707, doi:10.1007/978-3-662-46681-0_61.

[KNP10] M. Kwiatkowska, G. Norman & D. Parker (2010): *Advances and Challenges of Probabilistic Model Checking*. In: Proc. 48th Annual Allerton Conference on Communication, Control and Computing, IEEE Press, pp. 1691–1698, doi:10.1109/ALLERTON.2010.5707120.

[KR83] R. Kuiper & W.-P. de Roever (1983): *Fairness Assumptions for CSP in a Temporal Logic Framework*. In D. Bjørner, editor: Formal Description of Programming Concepts II, North-Holland, pp. 159–170.

[KW97] E. Kindler & R. Walter (1997): *Mutex Needs Fairness*. Information Processing Letters 62(1), pp. 31–39, doi:10.1016/S0020-0190(97)00033-1.

[Lam77] L. Lamport (1977): *Proving the correctness of multiprocess programs*. IEEE Transactions on Software Engineering 3(2), pp. 125–143, doi:10.1109/TSE.1977.229904.

[Lam02] L. Lamport (2002): *Specifying Systems, The TLA+ Language and Tools for Hardware and Software Engineers*. Addison-Wesley.

[LPS81] D.J. Lehmann, A. Pnueli & J. Stavi (1981): *Impartiality, Justice and Fairness: The Ethics of Concurrent Termination*. In S. Even & O. Kariv, editors: Automata, Languages and Programming (ICALP), LNCS 115, Springer, pp. 264–277, doi:10.1007/3-540-10843-2_22.

[Lyn68] W.C. Lynch (1968): *Reliable full-duplex file transmission over half-duplex telephone line*. CACM 11(6), pp. 407–410, doi:10.1145/363347.363366.

[Lyn96] N.A. Lynch (1996): *Distributed Algorithms*. Morgan Kaufmann.

[Mil89] R. Milner (1989): *Communication and concurrency*. PHI Series in computer science, Prentice Hall.

[Mil99] R. Milner (1999): *Communicating and mobile systems - the Pi-calculus*. Cambridge University Press.

[MM01] A.K. McIver & C. Morgan (2001): *Demonic, angelic and unbounded probabilistic choices in sequential programs*. Acta Informatica 37(4/5), pp. 329–354, doi:10.1007/s002360000046.

[MOP89] A.W. Mazurkiewicz, E. Ochmanski & W. Penczek (1989): *Concurrent Systems and Inevitability*. Theoretical Computer Science 64(3), pp. 281–304, doi:10.1016/0304-3975(89)90052-2.

[Mor94] C. Morgan (1994): *Programming from specifications*, 2nd edition. Prentice Hall International series in computer science, Prentice Hall.

[NC95] V. Natarajan & R. Cleaveland (1995): *Divergence and Fair Testing*. In Z. Fülöp & F. Gécseg, editors: Proceedings 22nd International Colloquium on Automata, Languages and Programming (ICALP'95), LNCS 944, Springer, pp. 648–659, doi:10.1007/3-540-60084-1_112.

[Old91] E.-R. Olderog (1991): *Nets, Terms and Formulas: Three Views of Concurrent Processes and their Relationship*. Cambridge Tracts in Theoretical Computer Science 23, Cambridge University Press, doi:10.1017/CBO9780511526589.

[PBD03] C.E. Perkins, E.M. Belding-Royer & S. Das (2003): *Ad hoc On-Demand Distance Vector (AODV) Routing*. RFC 3561, Network Working Group. At http://www.ietf.org/rfc/rfc3561.txt.

[Pnu77] A. Pnueli (1977): *The Temporal Logic of Programs*. In: Foundations of Computer Science (FOCS'77), IEEE, pp. 46–57, doi:10.1109/SFCS.1977.32.

[Pra91] V.R. Pratt (1991): *Modeling Concurrency with Geometry*. In: Proc. 18th Annual ACM Symposium on Principles of Programming Languages, pp. 311–322, doi:10.1145/99583.99625.

[Rei13] W. Reisig (2013): *Understanding Petri Nets — Modeling Techniques, Analysis Methods, Case Studies*. Springer, doi:10.1007/978-3-642-33278-4.

[Ros97] A.W. Roscoe (1997): *The Theory and Practice of Concurrency*. Prentice-Hall. Available at http://www.comlab.ox.ac.uk/bill.roscoe/publications/68b.pdf.

[Seg96] R. Segala (1996): *Testing Probabilistic Automata*. In: Proceedings of the 7th International Conference on Concurrency Theory (CONCUR'96), LNCS 1119, Springer, pp. 299–314, doi:10.1007/3-540-61604-7_62.

[Sim85] R. de Simone (1985): *Higher-level synchronising devices in* MEIJE-*SCCS*. Theoretical Computer Science 37, pp. 245–267, doi:10.1016/0304-3975(85)90093-3.

[SW01] D. Sangiorgi & D. Walker (2001): *The π-calculus: A Theory of Mobile Processes*. Cambridge University Press.

[Uli92] I. Ulidowski (1992): *Equivalences on Observable Processes*. In: Proc. Seventh Annual Symposium on Logic in Computer Science (LICS'92), IEEE Computer Society, pp. 148–159, doi:10.1109/LICS.1992.185529.

[Uli00] I. Ulidowski (2000): *Finite axiom systems for testing preorder and De Simone process languages*. Theoretical Computer Science 239(1), pp. 97–139, doi:10.1016/S0304-3975(99)00214-5.

[UP02] I. Ulidowski & I.C.C. Phillips (2002): *Ordered SOS Process Languages for Branching and Eager Bisimulations*. Information and Computation 178(1), pp. 180–213, doi:10.1006/inco.2002.3161.

[UY00] I. Ulidowski & S. Yuen (2000): *Process Languages for Rooted Eager Bisimulation*. In C. Palamidessi, editor: Proceedings 11th International Conference on Concurrency Theory (CONCUR'00), LNCS 1877, Springer, pp. 275–289, doi:10.1007/3-540-44618-4_21.

[Ver95] C. Verhoef (1995): *A Congruence Theorem for Structured Operational Semantics with Predicates and Negative Premises*. Nord. J. Comput. 2(2), pp. 274–302.

[Vog92] W. Vogler (1992): *Modular Construction and Partial Order Semantics of Petri Nets*. LNCS 625, Springer, doi:10.1007/3-540-55767-9.

[Vog02] W. Vogler (2002): *Efficiency of asynchronous systems, read arcs, and the MUTEX-problem*. Theoretical Computer Science 275(1-2), pp. 589–631, doi:10.1016/S0304-3975(01)00300-0.

[Wal90] D.J. Walker (1990): *Bisimulation and divergence*. Information and Computation 85(2), pp. 202–241, doi:10.1016/0890-5401(90)90048-M.

[Win84] G. Winskel (1984): *A new definition of morphism on Petri nets*. In M. Fontet & K. Mehlhorn, editors: Proceedings STACS 84, LNCS 166, Springer, pp. 140–150, doi:10.1007/3-540-12920-0_13.

[Win87] G. Winskel (1987): *Event structures*. In W. Brauer, W. Reisig & G. Rozenberg, editors: Petri Nets: Applications and Relationships to Other Models of Concurrency, Advances in Petri Nets 1986, Part II, LNCS 255, Springer, pp. 325–392, doi:10.1007/3-540-17906-2_31.

[WL92] Wang Yi & K.G. Larsen (1992): *Testing Probabilistic and Nondeterministic Processes*. In: Proceedings of the IFIP TC6/WG6.1 Twelfth International Symposium on Protocol Specification, Testing and Verification, IFIP Transactions C-8, North-Holland, pp. 47–61.