SOFTWARE VERIFICATION RESEARCH CENTRE

THE UNIVERSITY OF QUEENSLAND

Queensland 4072
Australia

TECHNICAL REPORT

No. 02-17

On Randomization versus Synchronization

Hagen Völzer

Software Verification Research Centre
The University of Queensland
Australia

April 19, 2002

Phone: +61 7 3365 1003
Fax: +61 7 3365 1533
http://svrc.it.uq.edu.au

Note: Most SVRC technical reports are available via anonymous ftp, from `svrc.it.uq.edu.au` in the directory `/pub/techreports`. Abstracts and compressed postscript files are available from `http://svrc.it.uq.edu.au`

On Randomization versus Synchronization

Hagen Völzer*

Software Verification Research Centre
The University of Queensland
Australia

Abstract

We expose limitations of randomization in asynchronous distributed systems by presenting two new impossibility results for paradigmatic problems. The first result states that mutual exclusion cannot be solved starvation free with probability 1 in the absence of (strongly) fair synchronization. That is, for example, an agent waiting to access a shared variable eventually accesses the variable. The second result states that a crash-tolerant version of the dining philosophers problem cannot be solved starvation free with probability 1 even if fair synchronization is assumed. In both cases, randomization does not (sufficiently) increase the ability of the underlying model to synchronize independent objects.

1 Introduction

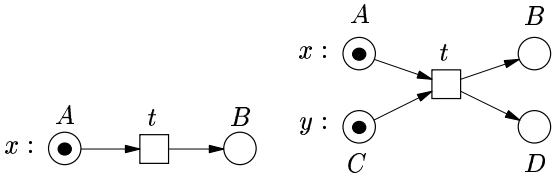
The benefit of randomization in distributed algorithms, i.e., agents flipping coins during the execution of their program, is well known. Randomized algorithms are often simpler and more efficient than their deterministic counterparts and they sometimes solve problems deterministic algorithms provably cannot solve. Much less is known about the limitations of randomization with respect to the solvability of coordination and synchronization problems.

One of the most prominent examples for the benefit of randomization is symmetry breaking. There, randomization allows us to construct symmetric algorithms that start in a symmetric state and lead to an asymmetric state with probability 1. Examples for symmetry breaking problems are leader election and mutual exclusion.

Hart, Sharir, and Pnueli [5] show however that it still depends on the particular problem whether a symmetric randomized solution exists. While there is a symmetric solution to the leader election problem, they show that every fully symmetric mutual exclusion protocol where processors communicate via a shared variable with separate read/write operations deadlocks with positive probability. They conclude: "These phenomena call for further study to understand better the distinction between those concurrent problems that admit probabilistic solutions that are better than deterministic solutions, and those problems that do not benefit from introduction of randomization."

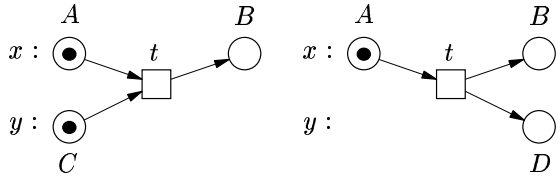
For some problems and a given model it turns out that the liveness and the safety specifications of the problem are incompatible, i.e., the liveness specification implies that there are executions that violate the safety specification. In these cases there is no hope that the problem can be

*Postal address: Hagen Voelzer, SVRC, The University of Queensland, Qld 4072, Australia; e-mail: voelzer@svrc.uq.edu.au; Phone: +61 7 3365 1647; Fax: +61 7 3365 1533; supported by Deutsche Forschungsgemeinschaft: Project "Konsensalgorithmen" and funded in part by Australian Research Council, Large Grant A49801500, A Unified Formalism for Concurrent Real-Time Software Development.



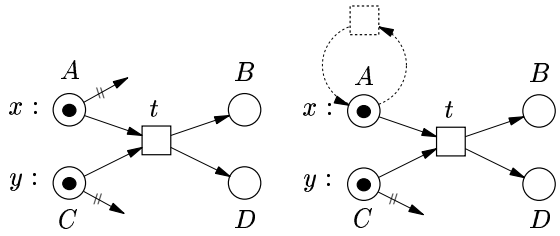
(a) Local change (b) Synchronization

Fig. 1: Two transitions



(a) Join (b) Fork

Fig. 2: Special cases of synchronization



(a) Progress (b) Fairness

Fig. 3: Two synchronization assumptions

solved with probability 1 by introducing randomization to the model. Examples for impossibilities of that kind are finding the size of a ring where agents are indistinguishable [6] and reaching consensus by message passing if as many as half of the agents may crash [2].

If the safety and liveness specifications are compatible then the existence of a solution depends on the *liveness assumption* of the model, that is for example, wait-freedom or other fairness assumptions for shared objects, the assumption that each message will eventually be received, and weak and strong completeness of failure detectors.

In this paper, we consider problems where the existence of a solution depends on liveness assumptions, in particular on *synchronization assumptions*.

We use Petri nets [12, 14] to formalize and visualize the synchronization assumptions. Fig. 1a shows an object x in state A that changes its state to B by the occurrence of transition t (t removes the token from A and puts a token into B). An object could be a processor, a shared variable, or a message. Fig. 1b shows a synchronization of the objects x and y. Both objects change their state simultaneously with the occurrence of t. Object x changes from state A to B and y changes from C to D. Object x could be a processor and y could be a shared variable in this case. If object x is in state A we also say x is *ready* for t.

Fig. 2 shows a join and a fork, two special cases of the synchronization in Fig. 1b, which can represent a processor x receiving and x sending a message y, respectively.

We distinguish three basic architecture independent synchronization assumptions, *progress*, *fairness*, and *hyperfairness*. We describe the assumptions here for two-party synchronization. Generalization to n-party synchronization is straight-forward and can be found in the remainder of this paper.

Progress means that both objects eventually synchronize in t if both wait for each other when ready for t, i.e., neither object engages in another transition departing from A and C, respectively. This is symbolically depicted by the negated arcs in Fig. 3a. Progress for t is essentially *weak fairness*[1] (also *justice*) for transition t. That is t eventually occurs when continuously enabled. If x is a shared object, it means deadlock freedom of object x—as long there is some process that is ready to access x then some process will access x.

Fairness means that both objects eventually synchronize in t if one object waits for the other and the other object is always eventually ready,

[1] Depending on the formalization, weak fairness can be slightly stronger than progress, but, in a faithful modelling of an asynchronous system, both notions coincide.

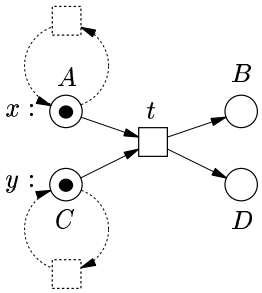


Fig. 4: Hyperfairness

i.e., it also waits or it always returns to the state where it is ready for t (Fig. 3b). Fairness is essentially *strong fairness*[2] (also *compassion*) for t, that is t occurs when always eventually enabled. If x is a shared object then fairness means wait-freedom (also starvation freedom) of x, that is y will eventually access x when waiting to access x. In the context of messages it means that each message that is sent to a process x is eventually received by x even if another process independently sends infinitely many messages to x.

Hyperfairness means that both objects synchronize in t when both objects are always, independently from each other, eventually ready (Fig. 4). This is a strong assumption, usually too strong to postulate as it assumes that both objects are eventually ready at the same time although they leave and enter their ready state independently from each other. Hyperfairness was introduced by Attie, Francez, and Grumberg [1] and our notion gives a new formalization for their concept. Lamport's hyperfairness [10] is a much stronger notion which we do not consider here. Elsewhere [16], we have shown that hyperfairness can be often implemented with the help of partial synchrony[3]. Here, hyperfairness occurs only for completing the picture.

The relative power of the three assumptions is illustrated by the following example, which is a three-party synchronization. Consider a pedestrian that wants to cross an asynchronous two-lane road (Fig. 5). To cross, the pedestrian needs two resources at the same time: a sufficiently wide space on the left lane and one on the right lane (assume that he cannot safely wait between the two lanes). Progress assumes that he will cross the road if both resources are permanently available, i.e., if no cars come. Fairness assumes that he will also eventually cross the road if one resource is permanently available and the other always eventually. This includes cases with infinitely many cars on one lane. Hyperfairness assumes that the pedestrian also eventually crosses if both resources are always, independently from each other, eventually available. This includes cases with infinitely many cars on both lanes.

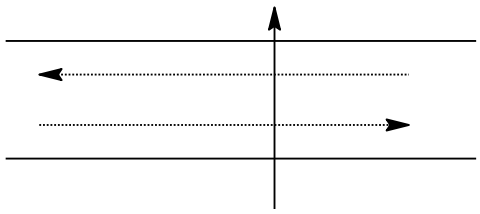


Fig. 5: A pedestrian crossing a two-lane road

Consider now the classical mutual exclusion problem for two agents: Each agent cycles through the three states *quiet*, *hungry*, and *critical* such that both agents are never critical at the same time. Fig. 6 shows a simple mutex system Σ_1 with a central key that is needed for entering the critical state. Transitions a_l and a_r are grey shaded to designate that no liveness is assumed for them, i.e., an agent may remain quiet forever. For all other transitions, progress is assumed. This still does not guarantee *starvation freedom* of the mutex system as one agent may remain hungry forever while the other is using the key infinitely often.

It is known that starvation-free mutex cannot be solved if only progress is assumed [7, 15]. All known mutex-solutions use fairness at one place or another[4]. Note that Σ_1 in Fig. 6 guarantees

[2] In a faithful modelling of an asynchronous system, both notions coincide.

[3] That is the assumption that each event consumes a bounded amount of time, where, however, the bound is not known.

[4] The fairness assumption is sometimes not explicit. In Lamport's bakery algorithm [8], for example, the fairness assumption is that each customer eventually draws

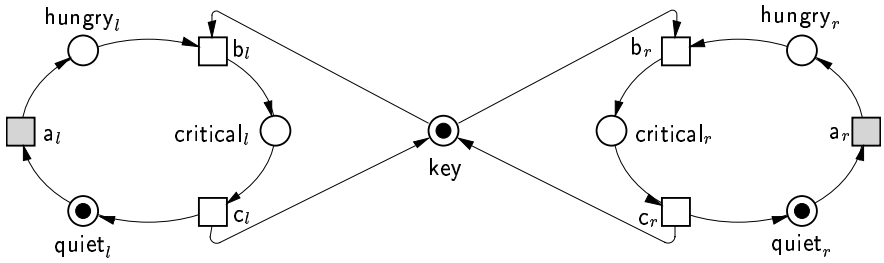


Fig. 6: Σ_1 – a mutex system

starvation-freedom if fairness is assumed for b_l and b_r.

As a new result, we show in this paper that mutex cannot be solved starvation-free with probability 1 by progress and randomization. This implies that fairness cannot be implemented with probability 1 by progress and randomization.

When we compare mutex with the classical message-passing consensus problem with crashing agents (as defined by Fischer et al. [4]) we get the picture in Fig. 7. While mutex cannot be solved with progress and randomization, consensus can (Ben-Or's algorithm [2] does not need fairness) and, on the other hand while consensus cannot be solved with fairness alone, mutex can (fairness is as strong as the liveness assumed by Fischer et al. [4], which we proved elsewhere [16]). Hence fairness and randomization are incomparable with respect to their expressive power and mutex and consensus are incomparable with respect to their solvability.

It is therefore natural to investigate the power of the combination of fairness and randomization. Here we can obviously solve mutex as well as consensus. To show the limitation of this model, we consider a generalization of the mutex problem known as the dining philosophers problem [3]. Consider a finite set A of agents and an irreflexive and symmetric relation $N \subseteq A \times A$. If $(x, y) \in N$ we call x and y *neighbors*. We want to solve the mutex problem for each pair of neighbors simultaneously, i.e., two neighbors are never critical at the same time. Assume now that agents may crash. Since an agent may crash while being

a ticket in spite of other customers drawing infinitely many tickets.

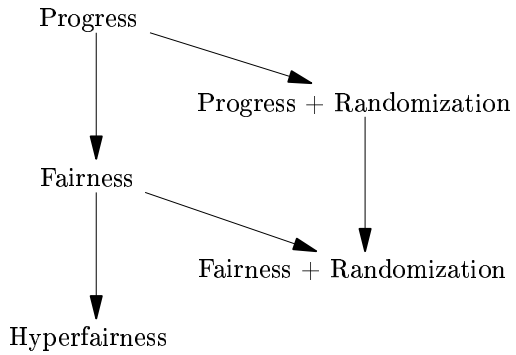


Fig. 7: The hierarchy of models

critical and we cannot detect crashes, we have to weaken the starvation freedom requirement. We demand that each hungry agent eventually becomes critical unless itself or one of its neighbors crashes. In the case of a crashing agent, this still requires a hungry agent with at least distance 2 to the crashing agent in the neighborhood graph to eventually become critical.

Although this seems to be a weak requirement, we show that this problem cannot be solved by fairness and randomization. This problem requires stronger synchronization than is provided by fairness. It requires synchronization of several independent objects that cannot wait for synchronization because of the crash-tolerance. Crash-tolerant generalized mutex can be solved by hyperfairness and, as we show elsewhere [16] by fairness, randomization, and partial synchrony.

We use partial-order semantics in this paper, i.e., an execution represents only the causal order of events (Lamport's happens-before relation [9]). This simplifies the definition of synchronization assumptions and some proofs. In particular, we

use a recently proposed partial-order semantics for randomized algorithms [17]. That semantics models an adversary, called the *distributed adversary*, that is weaker than the classical adversary, which strengthens our impossibility results. The distributed adversary does not know, in contrast to the classical adversary, any order on causally independent events and it cannot base a nondeterministic choice on the outcome of coin flips that are causally independent from that nondeterministic choice.

The remainder of the paper is organized as follows. In Sect. 2, we introduce our computational model. In Sect. 3, we define progress and present our first result. In Sect. 4, we define fairness and present our second result. Finally, we briefly discuss hyperfairness in Sect. 5.

2 Computational Model

In this section, we define Petri nets, our computational model, their partial-order semantics, and their randomized version.

Systems. A *Petri net* (or *net* for short) $N = (P, T, F)$ consists of two disjoint non-empty, countable sets P and T and a binary relation $F \subseteq (P \times T) \cup (T \times P)$. Elements of P, T, and F are called *places*, *transitions*, and *arcs* of the net respectively. We graphically represent a place by a circle, a transition by a square, and an arc by an arrow between the corresponding elements. An element of $P \cup T$ is also called an *element* of N. For each element x, we define the *preset* of x by ${}^\bullet x = \{y \mid (y, x) \in F\}$ and the *postset* of x by $x^\bullet = \{y \mid (x, y) \in F\}$. For a set X of elements, let ${}^\bullet X = \bigcup_{x \in X} {}^\bullet x$ and $X^\bullet = \bigcup_{x \in X} x^\bullet$. We restrict our attention to nets in which for each transition t, the preset ${}^\bullet t$ and the postset $t^\bullet$ are non-empty and finite. A set C of transitions of a net such that $|C| > 1$ is called a *conflict* if $\bigcap_{t \in C} {}^\bullet t \neq \varnothing$. A conflict C is *maximal* if there is no conflict that contains C.

A *state*, also called a *marking*, M of a net is a finite bag (or multiset) over P. A marking is graphically represented by black tokens in the places of the net. By $+, -$, and $\leq$ we denote bag addition, subtraction , and inclusion, respectively. A subset of P is treated as a bag over P by identifying it with its characteristic function. A transition t is *enabled* in a given marking M if ${}^\bullet t \leq M$, i.e., if all tokens specified by the preset of t are present in M. If t is enabled in a marking M_1 then t may *occur*, resulting in the *follower marking* $M_2 = (M_1 - {}^\bullet t) + t^\bullet$. This is denoted $M_1 \xrightarrow{t} M_2$.

A pair $\Sigma = (N, M^0)$ of a net N and a marking M^0 of N is called a *system*. The marking M^0 is called the *initial marking* of Σ. A finite or infinite alternating sequence $M_0, t_1, M_1, \ldots$ of markings and transitions of Σ that starts in the initial marking of Σ and respects the occurrence relation $\xrightarrow{t}$ is called a *sequential execution* of Σ. Similarly, a *computation tree* of Σ is a finite or infinite labelled tree where vertices are labelled with markings and edges are labelled with transitions such that the root is labelled with the initial marking and the labelling respects the occurrence relation. There is a natural prefix order on all computation trees of Σ with a maximal computation tree that is unique up to isomorphism. Note that a sequential execution is a non-branching computation tree.

Partial-order semantics. Fig. 8 shows a partial-order execution of Σ_1 from Fig. 6. A square represents an *event*, that is an occurrence of a transition and a circle represents a *condition*, that is the occurrence of a token in a place. In contrast to a sequential execution, a partial-order execution does not represent a total order on the events but a partial order, called *causal order*. While b_r is causally dependent on b_l, the events labelled with b_l and a_r are independent (not ordered). A partial-order execution ρ represents a set of sequential executions, called *sequentializations* of ρ, which can be derived by arbitrarily increasing the causal order of ρ to a total order. The partial-order counterpart of a computation tree is an *unfolding* [11]. Fig. 9 shows an unfolding of Σ_1. A partial-order execution is a conflict-free unfolding.

Let N be a net. For each element x of N, we define the set of *predecessors* of x by $\downarrow x = \{y \mid$

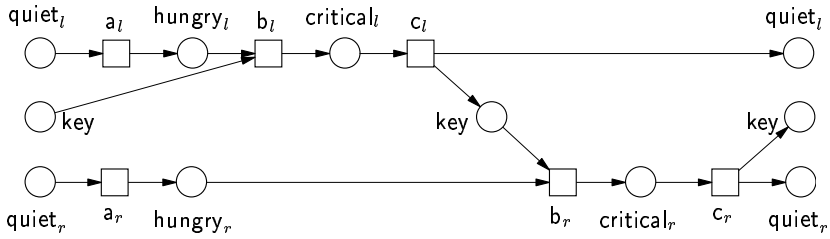


Fig. 8: A partial-order execution of Σ_1

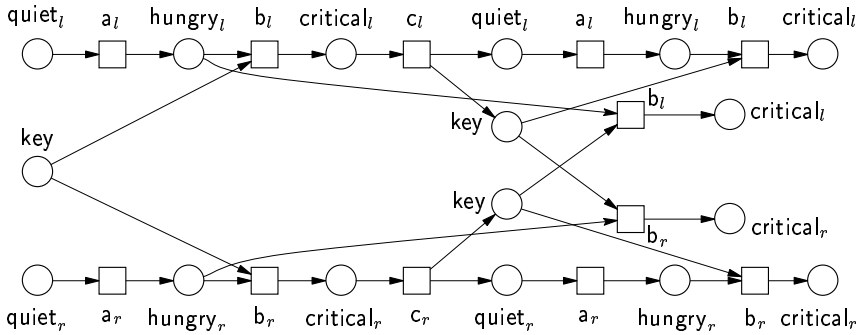


Fig. 9: An unfolding of Σ_1

$yF^+x\}$ where F^+ denotes the transitive closure of F. N is *acyclic* if for each element x of N, we have $x \notin \downarrow x$ and N is *predecessor-finite* if for each element x of N, the set $\downarrow x$ is finite. Let $K = (B, E, \lessdot)$ be an acyclic, predecessor-finite net. A place $b \in B$ is called a *condition* and a transition $e \in E$ is called an *event*. Since K is acyclic, the transitive closure of $\lessdot$, denoted $<$, is a partial order, which we call the *causal order*. If we have $x_1 < x_2$ or $x_2 < x_1$ then we say x_1 and x_2 are *causally dependent*. If we have $x_1 < x_2$ or $x_1 = x_2$ then we write $x_1 \leq x_2$. Two elements are *in (extended) conflict*, denoted $x_1 \# x_2$, if there is a conflict $\{e_1, e_2\}$ such that $e_i \leq x_i$ for $i = 1, 2$. Two different elements are *causally independent* (or *concurrent*), denoted $x_1 \operatorname{co} x_2$, if they are neither causally dependent nor in conflict. A *co-set* is a finite set of pairwise concurrent conditions. A *cut* is a maximal co-set. A co-set D is *reachable* from a cut C if for all $c \in C$ and all $d \in D$, we have $c \leq d$ or $c \operatorname{co} d$.

A predecessor-finite, acyclic net K is called an *occurrence net*, if its set of minimal conditions ${}^\circ K = \{b \in B \mid {}^\bullet b = \varnothing\}$ is finite, and for each condition b of K we have $|{}^\bullet b| \leq 1$, and for each event e of K, its preset ${}^\bullet e$ is a co-set. Let Σ be a system and let $K = (B, E, \lessdot)$ be an occurrence net. Let $l : B \cup E \to P \cup T$ be a labelling such that $l(B) \subseteq P$ and $l(E) \subseteq T$. For a co-set D, $l(D)$ denotes a marking defined by $l(D) = \sum_{b \in D} l(b)$. The pair $\pi = (K, l)$ is an *unfolding* of Σ if $l({}^\circ K) = M^0$, and for each event e of K, we have $l({}^\bullet e) = {}^\bullet l(e)$ and $l(e^\bullet) = l(e)^\bullet$, and for all events e_1, e_2 of K, we have ${}^\bullet e_1 = {}^\bullet e_2$ and $l(e_1) = l(e_2)$ implies $e_1 = e_2$. For two conditions b_1 and b_2 of an unfolding, exactly one of the following five relations holds: $b_1 = b_2$ or $b_1 < b_2$ or $b_2 < b_1$ or $b_1 \# b_2$ or $b_1 \operatorname{co} b_2$.

There is a natural prefix-order on all unfoldings of a system with a maximal unfolding, which is unique up to isomorphism. A conflict-free prefix of an unfolding π is called a *partial-order execution* or *run* of π. $\mathfrak{R}(\pi), \mathfrak{R}_{\max}(\pi)$, and $\mathfrak{R}_{\text{fin}}(\pi)$ denote the set of all runs, all maximal runs (with respect to the prefix order), and the set of all finite runs of π, respectively. Note that, in contrast to sequential executions, there are infinite partial-order executions that can be extended. For example, in the run of Σ_1 where l becomes critical infinitely often and r remains quiet, the initial *quiet*$_r$-condition has no successor, hence the run can be extended by an occurrence of transition a_r.

We use a simple temporal logic to specify temporal properties containing the operators *always* $\Box$, *eventually* $\Diamond$, and *leads-to* $\rhd$. If p is a place then p is also a *state formula*, which is *valid* in a marking that has at least one token on p. State formulas can also be obtained by combining state formulas with Boolean connectors with the usual meaning. A state formula φ is also a *temporal formula*, which is valid in a cut C if φ is valid in the marking $l(C)$. If Φ is a temporal formula, so is $\Box\,\Phi$, which is valid in a cut C if Φ is valid in all cuts that are reachable from C. Temporal formulas can also be obtained by combining temporal formulas with Boolean connectors with the usual meaning. We write $\Diamond\,\Phi$ for $\neg\,\Box\,\neg\Phi$ and $\Phi \rhd \Psi$ for $\Box(\Phi \Rightarrow \Diamond\,\Psi)$. We will also use universal quantification over finite sets as an abbreviation for finite conjunction. A temporal formula Φ is valid in a run $\rho = (K, l)$ if it is valid in the initial cut ${}^\circ K$ of ρ. For a given temporal formula Φ, the set of all runs ρ such that Φ is valid in ρ is called *temporal-logical property*.

Randomized systems. Fig. 10 shows an object flipping an n-sided coin. Each outcome of the coin flip is represented by a transition t_i which has a probability p_i associated with it such that $\sum_{i=1}^{n} p_i = 1$. We model coin flips as internal events of objects, i.e., each transition t_i is *free*, that is $|{}^\bullet t_i| = 1$. The t_i are called *probabilistic transitions* and are graphically distinguished by the symbol %.

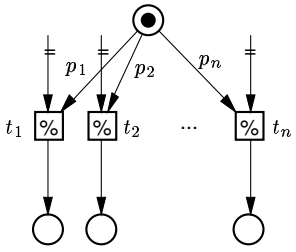


Fig. 10: An object flipping a coin

A *randomized system* consists of a system, a set $T^{\text{flip}} \subseteq T$ of free transitions, called *probabilistic transitions*, and a mapping $\mu : T^{\text{flip}} \to [0, 1]$ such that: each conflict with $C \subseteq T^{\text{flip}}$, called *probabilistic conflict*, is finite, $0 < \mu(t) < 1$ for all $t \in T^{\text{flip}}$, and for each maximal probabilistic conflict C, we have $\sum_{t \in C} \mu(t) = 1$.

In the classical sequential semantics for randomized systems, there is a canonical probability space associated with each *probabilistic computation tree*, that is a computation tree in which each branching represents a coin flip, i.e., there is no (pure) nondeterminism in a probabilistic computation tree. We use the partial-order counterpart of that semantics [17].

Let Σ be a randomized system and $\pi = (K, l)$ an unfolding of Σ. Let $E^{\text{flip}} = \{e \in E \mid l(e) \in T^{\text{flip}}\}$ be the set of *probabilistic events* of π and carry over probabilities by $\mu(e) = \mu(l(e))$ for each $e \in E^{\text{flip}}$. Call a probabilistic conflict C of π *complete* if $\sum_{e \in C} \mu(e) = 1$, i.e., if it contains all outcomes of the coin flip. An unfolding π of Σ is called *probabilistic* if each conflict in π is a complete probabilistic conflict. For each probabilistic unfolding π, there is a unique probability space $(\Omega_\pi, \mathcal{A}_\pi, P_\pi)$ such that $\Omega_\pi = \mathfrak{R}_{\max}(\pi)$, for each finite run α of π, its *cone* $K(\alpha) = \{\rho \in \mathfrak{R}_{\max}(\pi) \mid \alpha \text{ is a prefix of } \rho\}$ is in $\mathcal{A}_\pi$, and $P_\pi(K(\alpha)) = \prod_{e \in E_\alpha^{\text{flip}}} \mu(e)$ $(= 1$ if $E_\alpha^{\text{flip}} = \varnothing)$ where E_α^{flip} denotes the set of all probabilistic events of α.

Temporal-logical properties are measurable in that probability space and a temporal-logical property X is said to be *1-valid* in π if $P_\pi(X) = 1$ and *1-valid* in Σ if it is 1-valid in each *admissible* probabilistic unfolding π of Σ, where *admissible* will be defined by the liveness assumption of the particular model. As explained in the introduction, this semantics is more liberal than the classical sequential semantics, i.e., roughly speaking, if a property is 1-valid in all admissible computation trees then it is also 1-valid in all admissible probabilistic unfoldings but not necessarily vice versa (cf. [17]).

If X is a safety property that is 1-valid in Σ then X is satisfied in every run of Σ. (If X is violated then there is, by definition of a safety property, a finite run α such that all runs in $K(\alpha)$ violate X and we have $P_\pi(K(\alpha)) > 0$).

3 Progress

We present our first impossibility result in this section. We first formalize progress and define then which systems constitute a mutex solution.

A *progress system* consists of a system and a set of distinguished transitions $T^{\text{prog}} \subseteq T$, called *progress transitions*. A transition $t \in T \setminus T^{\text{prog}}$ is graphically distinguished by a grey shade. For a randomized progress system, we can assume $T^{\text{flip}} \subseteq T^{\text{prog}}$ without loss of generality.

Let Q be a set of places of a system Σ and let $\rho = (K, l)$ be a run of Σ. A co-set D of ρ is called a *Q-set* if $l(D) = Q$. Q is *persistent* in ρ if there is a Q-set D in ρ such that $D^\bullet = \varnothing$, i.e., it is a set of tokens that are never consumed in ρ. A run ρ *violates progress* with respect to t if ${}^\bullet t$ is persistent in ρ. A run ρ is *progressive* if there is no transition $t \in T^{\text{prog}}$ such that ρ violates progress with respect to t. An unfolding π is *progressive* if each run of π is progressive. Every (finite or infinite) unfolding can be extended to a progressive unfolding.

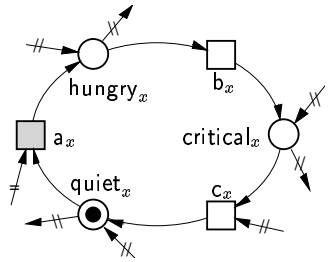


Fig. 11: Mutex structure Σ_x

We specify mutex in two steps (similar to Kindler and Walter's approach [7]). First we define *mutex structure* and then *mutex behavior*. Although we consider only two agents in this section, we define mutex structure for a finite set A of agents to reuse the definition later. A progress system Σ has *mutex structure for A* if for each agent $x \in A$, (1) there is exactly one progress system Σ_x as in Fig. 11 that is a subsystem of Σ such that all Σ_x are pairwise disjoint and (2) the connection of Σ_x with the rest of Σ is restricted as Fig. 11 indicates. The formalization of this is straight-forward [17]. Σ_x in Fig. 11 models a client x of a mutex algorithm. A client can inform the mutex algorithm about in which of the three states it is by outgoing arcs of a_x, b_x and c_x. The mutex algorithm can inform a client when it is allowed to enter the critical state by an ingoing arc to b_x. Note that a critical agent will always become quiet but a quiet agent may remain quiet forever. Note also that a critical agent need not wait for anything to leave the critical state.

A randomized system Σ has *probabilistic mutex behavior*, if the following two properties (1) and (2) are 1-valid in each progressive probabilistic unfolding of Σ:

$$\Box\, \neg(\mathit{critical}_l \wedge \mathit{critical}_r) \tag{1}$$

$$\forall x \in \{l, r\} : \mathit{hungry}_x \rhd \mathit{critical}_x \tag{2}$$

Theorem 1 There is no randomized progress system with mutex structure for $\{l, r\}$ and probabilistic mutex behavior.

Proof: To derive a contradiction, let Σ be a randomized progress system with mutex structure and probabilistic mutex behavior. There is a progressive probabilistic unfolding π_1 where r is never hungry but l is always eventually hungry: Start with the initial marking and let l become hungry and extend by progress, i.e., extend to a minimal progressive probabilistic unfolding. Let l become hungry everywhere (in every branch) where possible and repeat infinitely.

There is a progressive probabilistic unfolding π_2 that extends π_1 in which r becomes hungry: Let a_r occur and extend by progress. Since Σ has probabilistic mutex behavior, r becomes critical with probability 1 in π_2. Let b_r be a condition of π_2 such that $l(b_r) = \mathit{critical}_r$. We know by construction that b_r does not belong to π_1. Let C be a cut such that $b_r \in C$. In C, we have either quiet_l or hungry_l or $\mathit{critical}_l$, where $\mathit{critical}_l$ can be ruled out because that violates (1). Assume hunrgy_l. Each hunrgy_l-condition belongs to π_1 by construction. Therefore, there is a reachable $\mathit{critical}_l$-condition b_l that belongs to π_1. Obviously, we do not have $b_l = b_r$ since $l(b_l) \neq l(b_r)$. We also do not have b_r co b_l because of (1). We

have neither $b_l \# b_r$ nor $b_l < b_r$ because b_l is reachable from C. Finally, we also do not have $b_r < b_l$ since b_l belongs to π_1 and b_r does not belong to π_1. This contradicts π_1 being an unfolding. Therefore, we do not have $hungry_l$ but $quiet_l$ in C.

This $quiet_l$-condition has no event consuming it, because otherwise the subsequent $hungry_l$-condition would be together with b_r in some cut and we can argue as before. However, every $quiet_l$-condition is followed by a $hunrgy_l$-condition with probability 1 by construction—in π_1 as well as in π_2. Therefore the situation can only occur with probability 0 and hence r can be critical with at most probability 0 in π_2.

4 Fairness

In this section, we formalize fairness and present our second impossibility result.

A *fairness system* consists of a progress system and a set of distinguished transitions $T^{\text{fair}} \subseteq T^{\text{prog}}$, called *fairness transitions*. For a randomized fairness system we can assume $T^{\text{flip}} \cap T^{\text{fair}} = \varnothing$ without loss of generality. Fairness uses a notion of concurrent conditions being available at the same time. The conditions of a co-set D of a run are available at the same time if $e \in D^\bullet \Rightarrow D \subseteq {}^\bullet e$, i.e., if D is persistent or the conditions of D are synchronized by some event e.

A set Q of places of a system Σ is *strongly insistent* in a run ρ if for each cut C of ρ, there is a reachable Q-set D such that $e \in D^\bullet \Rightarrow D \subseteq {}^\bullet e$. A run ρ *violates fairness* with respect to t if there is a set $Q \subseteq {}^\bullet t$ such that Q is persistent in ρ, ${}^\bullet t \setminus Q$ is strongly insistent in ρ, and t occurs only finitely many times in ρ. Fairness requires a set of objects to wait for the synchronization and the rest to be always eventually available together at the same time. If a run ρ violates fairness with respect to t then t is enabled infinitely often in each sequentialization of ρ. A run is *fair* if there is no transition $t \in T^{\text{fair}}$ such that ρ violates fairness with respect to t. A probabilistic unfolding π is fair if each run of π is fair. Each finite probabilistic unfolding can be extended to a progressive and fair probabilistic unfolding.

We consider now the dining philosophers problem [3], a generalization of Dijkstra's ring of five philosophers. Let A be a finite set of agents and $N \subseteq A \times A$ be an irreflexive and symmetric relation, called *neighborhood relation*. A system Σ has *generalized mutex behavior* if mutex is satisfied for each pair of neighbors and starvation freedom for each agent, i.e., if the following two formulas are valid in each admissible run of Σ:

$$\forall (x,y) \in N : \Box\, \neg (critical_x \wedge critical_y) \quad (3)$$

$$\forall x \in A : hungry_x \rhd critical_x \quad (4)$$

Due to Theorem 1, fairness is needed for this problem and fairness is also sufficient [3]. However, consider now that agents may crash permanently. For our purposes, it suffices to allow that critical agents may crash, i.e., we do not require the transition c_x from Fig. 11 to be a progress transition anymore. We call the resulting notion of mutex structure a *crash-prone mutex structure*. For each agent $x \in A$, we define the predicate $crashed(x)$ to be valid in a cut C of a run ρ if there is a $critical_x$-condition $b \in C$ such that $b^\bullet = \varnothing$. When an agent crashes, we do not require neighboring agents to become critical again, i.e., we weaken the starvation-freedom (4) to

$$\forall x \in A : hungry_x \rhd (critical_x \vee crashed(x) \vee (\exists y : (x,y) \in N \wedge crashed(y))) \quad (5)$$

where $crashed(x)$ can also be omitted as we allow only critical agents to crash. A randomized fairness system Σ has *probabilistic crash-tolerant mutex behavior* for A and N if (3) and (5) are 1-valid in each progressive and fair probabilistic unfolding of Σ.

Theorem 2 Let A be a set of at least three agents. Then, there is a neighborhood relation N on A such that there is no randomized fairness system with a crash-prone mutex structure for A and probabilistic crash-tolerant mutex behavior for A and N.

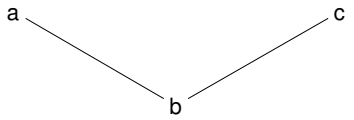


Fig. 12: Three agents in a neighborhood relation

Proof: Let A and N be as in Fig. 12. To derive a contradiction, let Σ be a randomized fairness system with crash-prone mutex structure for A and probabilistic crash-tolerant mutex behavior for A and N. We construct a fair and progressive probabilistic unfolding π such that $P_\pi((5)) < 1$. We will use a pattern of behavior that is known as *conspiracy due to race conditions* [1] where a and c conspire against b, that is, due to bad timing, there is always at least one of the two agents a and c in its critical state, hence b can never enter the critical state. We now start the construction of π in the initial marking of Σ.

1. Agent a becomes hungry. There is an extension π_0 where a becomes critical with probability 1. Then there is, for each $p_1 < 1$, a finite prefix π_1 of π_0 such that a becomes critical with at least probability p_1 in π_1 and that all $critical_a$-conditions have no successor. Note that, in π_1, there is no $critical_b$-condition.

2. Agent b becomes hungry.

3. Agent c becomes hungry. We extend to a finite probabilistic unfolding π_2 without using transition c_a (we pretend that a has crashed), such that c becomes critical with at least probability p_2 in π_2 and that all $critical_c$-conditions have no successor. In π_2, b becomes critical with at most probability $\epsilon_1 = 1 - p_1$ since b cannot, due to (3), be critical concurrently with a and we extended in this step concurrently to $critical_a$ with probability p_1, i.e., each condition of the extension is, with probability p_1, concurrent to a $critical_a$-condition.

4. Agent a becomes hungry in each run where it is not yet hungry. We extend to a finite probabilistic unfolding π_3 without using transition c_c such that $hungry_a \rhd critical_a$ is valid with at least p_3 and that all maximal $critical_a$-conditions have no successor. In π_3, b is critical with at most probability $\epsilon_1 + \epsilon_2$ where $\epsilon_2 = 1 - p_2$ since we extended with probability p_2 concurrently to $critical_c$ in this step.

5. Repeat the last two steps infinitely often.

We get an infinite progressive probabilistic unfolding where no agent crashes in any run of π; π can be constructed such that it is fair since we extend sequentially and whenever a fairness transition is enabled we can let it eventually occur. If there are conflicts between fairness transitions we can let the transitions of the conflict occur in round-robin fashion whenever the conflict is enabled. Agent b becomes critical with at most probability $\epsilon = \sum_{i=1}^{\infty} \epsilon_i$ in π and we can choose the p_i such that ϵ becomes arbitrarily small.

5 Hyperfairness

This section briefly introduces hyperfairness as a way to solve crash-tolerant generalized mutex. In contrast to fairness, hyperfairness does not *require* that all objects of a synchronization have to be ready at the same time—it *assumes* that all objects will eventually be ready at the same time provided that all objects return to their ready state independently from each other.

A set Q of places of a system Σ is *insistent* in a run ρ of Σ if for each cut of ρ, there is a reachable Q-set. A run violates *hyperfairness* with respect to t if ${}^\bullet t$ is insistent in ρ and t occurs only finitely many times in ρ. Hyperfairness just excludes the *conspiracies* that we used in the proof of Theorem 2.

Crash-tolerant generalized mutex can be solved by hyperfairness: Place a key between each pair of neighbors. For the resulting system, hyperfairness can be implemented by fairness, randomization, and partial synchrony with probability 1 [16]. Also message-passing consensus can be

solved with hyperfairness. We will treat hyperfairness in detail elsewhere.

6 Conclusion

It is easy to see that both presented results can be strengthened in the following way. If we want the safety specification of the problem to be 1-valid, which is equivalent to it being valid in all executions, then the liveness specification cannot be satisfied with high probability.

As we can see in randomized solutions to the consensus [2], leader election [6], and choice coordination [13] problems, randomization helps to coordinate causally independent decisions. That independence allows crash-tolerance in the case of the consensus problem.

We presented two examples in this paper where introduction of randomization did not help to solve the problems because it did not increase the ability of the model to synchronize independent objects. Note that our results establish that all depicted relationships in Fig. 7 are strict.

References

1. Paul C. Attie, Nissim Francez, and Orna Grumberg. Fairness and hyperfairness in multi-party interactions. *Distributed Computing*, 6:245–254, 1993.
2. Michael Ben-Or. Another advantage of free choice: Completely asynchronous agreement protocols. In *Proceedings of the 2nd Annual ACM Symposium on Principles of Distributed Computing*, pages 27–30. ACM, 1983.
3. K. Mani Chandy and Jayadev Misra. *Parallel Program Design: A Foundation*. Addison-Wesley, 1988.
4. Michael J. Fischer, Nancy A. Lynch, and Michael S. Paterson. Impossibility of distributed consensus with one faulty process. *Journal of the ACM*, 32(2):374–382, April 1985.
5. Sergiu Hart, Micha Sharir, and Amir Pnueli. Termination of probabilistic concurrent programs. *ACM Transactions on Programming Languages and Systems*, 5(3):356–380, July 1983.
6. Alon Itai and Michael Rodeh. Symmetry breaking in distributive networks. In *Proc. 22nd Annual Symposium on Foundations of Computer Science*, pages 150–158. IEEE, 1981.
7. Ekkart Kindler and Rolf Walter. Mutex needs fairness. *Information Processing Letters*, 62:31–39, 1997.
8. Leslie Lamport. A new solution of Dijkstra's concurrent programming problem. *Communications of the ACM*, 17(8):453–455, August 1974.
9. Leslie Lamport. Time, clocks, and the ordering of events in a distributed system. *Communications of the ACM*, 21(7):558–565, July 1978.
10. Leslie Lamport. Fairness and hyperfairness. *Distributed Computing*, 13(4):239–245, 2000.
11. Mogens Nielsen, Gordon Plotkin, and Glynn Winskel. Petri nets, event structures and domains, part I. *Theoretical Computer Science*, 13:85–108, 1981.
12. James L. Peterson. *Petri net theory and the modeling of systems*. Prentice-Hall, 1981.
13. Michael O. Rabin. The choice coordination problem. *Acta Informatica*, 17:121–134, 1982.
14. Wolfgang Reisig. *Petri nets : an introduction*, volume 4 of *EATCS monographs on theoretical computer science*. Springer-Verlag, 1985.
15. Walter Vogler. Efficiency of asynchronous systems and read arcs in Petri nets. In *Proceedings ICALP'97*, volume 1256 of *LNCS*, pages 538–548. Springer, 1997.
16. Hagen Völzer. *Fairneß, Randomisierung und Konspiration in Verteilten Algorithmen*. PhD thesis, Humboldt-Universität zu Berlin, Institut für Informatik, December 2000. in German, available via `http://dochost.rz.hu-berlin.de/dissertationen/voelzer-hagen-2000-12-08`.
17. Hagen Völzer. Randomized non-sequential processes. In *Proc. CONCUR 2001 – 12th International Conference on Concurrency Theory, Aalborg, Denmark*, volume 2154 of *LNCS*, pages 184–201. Springer, August 2001.