DO NOT USE THIS PAGE: COMPILED SEPARATELY.

Proving Correctness for Randomized Distributed Algorithms

by

Alain Isaac Saias

Agrégation de Mathématiques, Paris, France (1984)
Ecole Normale Supérieure

Submitted to the Department of Mathematics
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

February 1995

DO NOT USE THIS PAGE: COMPILED SEPARATELY.

DO NOT USE THIS PAGE: COMPILED SEPARATELY.

Proving Correctness for Randomized Distributed Algorithms

by

Alain Isaac Saias

Submitted to the Department of Mathematics
on February 1995, in Partial Fulfillments of the
Requirements for the Degree of
Doctor of Philosophy

Abstract

Thesis Supervisor: Nancy Lynch

Title: Professor of Electrical Engineering and Computer Science

DO NOT USE THIS PAGE: COMPILED SEPARATELY.

Randomness versus Non-Determinism in Distributed Computing

by

Alain Isaac Saias

Submitted to the Department of Mathematics
on February 1995, in Partial Fulfillments of the
Requirements for the Degree of
Doctor of Philosophy

Abstract

In randomized distributed computing, executions encounter branch points resolved either randomly or non-deterministically. Random decisions and non-deterministic choices interact and affect each other in subtle ways. This thesis is devoted to the analysis and illustration of the effects of the interplay between randomness and non-determinism in randomized computing.

Using ideas from game theory, we provide a general model for randomized computing which formalizes the mutual effects of randomization and non-determinism. An advantage of this model over previous models is that it is particularly effective for expressing mathematical proofs of correctness in two difficult domains in randomized computing. The first domain is the analysis of randomized algorithms where non-deterministic choices are made based on a limited knowledge of the execution history. The second domain concerns the establishment of lower- bounds and proofs of optimality.

The advantage of this model are described in the context of three problems. First, we consider the classical randomized algorithm for mutual exclusion [49] of Rabin. This algorithm illustrates perfectly the difficulties encountered when the non-deterministic choices are resolved based on a limited knowledge of execution history.

We then analyze the Lehmann-Rabin Dining Philosophers algorithm (1981). Our analysis provides a general method for deriving probabilistic time bounds for randomized executions.

In the last part, we analyze a scheduling problem and give solutions in both the deterministic and the randomized cases. Lower bounds arguments show these solutions to be optimal. For the randomized case, we take full advantage of the game theoretic interpretation of our general model. In particular, the proof of optimality reflects Von-Neumann's duality for matrix games.

Thesis Supervisor: Nancy Lynch

Title: Professor of Electrical Engineering and Computer Science

Acknowledgments

My thanks go first to my thesis advisor Professor Nancy Lynch † for having introduced me to randomized computing. She was the main inspiration in what became Chapter 3 and 4 of this thesis. It is a pleasure to acknowledge Michel Goemans for his contribution in Chapter 5: he provided the technical key to the final solution. I also thank Roberto Segala who designed the model presented in Chapter 4. I acknowledge also with gratitude Ran Canetti who suggested a nice improvement in a computation of Chapter 3.

Most of the work presented in my chapters 6 and 7 was conducted in the past year while I was a member of the Computer Research and Applications Group of the Los Alamos National Laboratory. I found the atmosphere at the laboratory very propitious for research and I am very indebted to group leader Dr. Vance Faber who very generously hosted me and allowed this work to fructify.

In the last push Martin Müller helped me considerably through the arcane oddities of Latex: no special documentstyle command can hide from him. Many thanks also to Steven Lines who accomplished a similar task at MIT and was there to help through all these years.

I owe a lot to the kindness of Phyllis Ruby and it is a great pleasure to be able to acknowledge her. She consistently helps students and the mathematics department would lose a lot of its humanity without her. I also thank Joanne Talbot in Computer Science for the help she provided while I was away in Los Alamos.

Then I have to thank all the good friends that have been around and had a real impact on my life. In particular Mike Hawrylycz has always been a special friend and a true support to the alien froggy that I am. Then, as space is scarce, I will just mention the names of my friends, but it comes with pleasure: Ana-Lia Barrantez, Claude Crepeau, Alain Karma, Sacha Karpovsky, Marios Mavronicolas, Mohamed Mekias, Mike Murphy, Kumiko Nagashima, Mark Newman, Daniel Provost, Sergio Rajsbaum, Isabelle Riviere, Michel Sadelain, Tomasz Taylor. And of course Marianna Kantor and my far away parents who all waited many years for this moment.

† The work of Professor Lynch was supported by the grants DARPA N00014-89-J-1988, DARPA N00014-92-J-4033, ONR N00014-91-J-1046, NSF 8915206-CCR, NSF 9225124-CCR, and AFOSR-ONR F49620-94-1-0199.

Contents

Chapter 1

Introduction

For many distributed problems, it is possible to produce randomized algorithms that are better than their deterministic counterparts: they may be more efficient, have simpler structure, and even achieve correctness properties that deterministic algorithms cannot. One problem with using randomization is the increased difficulty in analyzing the resulting algorithms. This thesis is concerned with this issue and provides formal methods and examples for the analysis of randomized algorithms, the proof of their correctness, the evaluation of their performance and in some instance the proof of their optimality.

By definition a randomized algorithm is one whose code can contain random choices, which lead to probabilistic branch points in the tree of executions. In order to perform these random choices the algorithm is provided at certain points of the execution with random inputs having known distributions: these random inputs are often called coin tosses and we accordingly say that the algorithm flips a coin to make a choice.

A major difficulty in the analysis of a randomized algorithm is that the code of the algorithm and the value of the random inputs do not always completely characterize the execution: the execution sometimes branches according to some nondeterministic choices which are not in the control of the algorithm. Typical examples of such choices are the choices of the inputs of the algorithm (in which case the corresponding branch point is at the very beginning of the execution), the scheduling of the processes (in a distributed environment), the control of the faults (in a faulty environment) and the changes in topology (in a dynamic environment). For the sake of modeling we call *adversary* an entity controlling these choices. In a general situation an adversary has also access to random sources to make these choices. (In

this case the adversary decides non-deterministically the probability distribution of the coin.)

A randomized algorithm therefore typically involves two different types of nondeterminism – that arising from the random choices whose probability distributions is specified in the code, and that arising from an *adversary*, resolving by definition all the choices for which no explicit randomized mechanism of decision is provided in the code.

The interaction between these two kinds of nondeterminism complicates significantly the analysis of randomized algorithms and is at the core of many mistakes. To understand the issues at stake consider a typical execution of a randomized algorithm. The execution runs as prescribed by the code of the algorithm until a decision not in the control of the code has to be resolved. (For instance, in a distributed context, the adversary decides, among other things, the order in which processes take steps.) This part of the execution typically involves random choices and therefore, the state reached by the system when a decision of the adversary is required is also random. Generally, the decision of the adversary depends on the random value of the state, s_1, reached. Its decision, a_1, in turn, characterizes the probabilistic way the execution proceeds in its second part: the branch of the code then followed is specified by $s_1\, a_1$. The execution proceeds along that branch, branching randomly as specified by the code until a second non-deterministic branch point must be resolved. The adversary then makes a decision. This decision generally depends on the random value of the state, s_2, reached, and, in turn, characterizes the probabilistic way the execution proceeds in its third part ... The execution thus proceeds, in a way where the random inputs used by the algorithm influence the decisions of the adversary; decisions which in turn determine the probability distributions of the random inputs used by the algorithm subsequently.

The analysis and the measure of the performance of randomized algorithms is usually performed in the worst case setting. An algorithm "performs well" if it does so against all adversaries. For example, among the correctness properties one often wishes to prove for randomized algorithms are properties that state that a certain property of executions has a "high" probability of holding against all adversaries; or that a certain random variable depending on the executions, (e.g., a running time), has a "small" expected value for all adversaries.

The proof of such properties often entails significant difficulties the first of which is to make formal sense of the property claimed. Such statements often implicitly assume the existence of a probability space whose sample space is the set of executions, and whose probability distribution is induced by the distribution of the random inputs used by the algorithm. But what is "this" probability space? As we saw, the

random inputs used during an execution depend on the decisions made previously in the execution by the adversary. This shows that we do not have one, but instead a family of probability spaces, one for each adversary. For each fixed adversary, the coins used by the algorithm are well-defined and characterize the probabilistic nature of the executions.

The analysis of a given randomized algorithm π therefore requires one to model the set of adversaries to be considered with π: we refer to them as the set of *admissible* adversaries. We must then construct (if possible) the probability space $(\Omega_{\mathcal{A}}, \mathcal{G}_{\mathcal{A}}, P_{\mathcal{A}})$ corresponding to each adversary $\mathcal{A}$. It should be noted that the choice $\mathcal{A}$ of the adversary not only affects the distribution $P_{\mathcal{A}}$, if it exists, induced by the random inputs on the set of executions, but also the set of executions itself. This remark justifies that the analysis of a randomized algorithm requires one to consider a different sample space $\Omega_{\mathcal{A}}$ and a different σ-field $\mathcal{G}_{\mathcal{A}}$ for every adversary $\mathcal{A}$.

Most authors, including the pioneers in the area of randomized computing, are aware that the adversary influences the probability distribution on the set of executions. For instance, in an early paper [37] Lehmann and Rabin define a schedule[1] $\mathcal{A}$ to be

> "*a function which assigns to every past behavior of the n processes the process whose turn is next to be active ... Under past behavior we mean the complete sequence of atomic actions and random draws with their results, up to that time ... This captures the idea that, for any specific system, what will happen next depends on the whole history of past successes and failures of the processes ... as well as on what happened internally within the processes.* "

To each such $\mathcal{A}$, the same paper [37] associates a probability distribution on the set of executions. We quote:

> "*For a given schedule $\mathcal{A}$ and specific outcomes of the random draws ι,*[2] *we get a particular computation $\omega = COM(\mathcal{A}, \iota)$ ... On the space of all possible outcomes of random draws ι we impose the uniform distribution. The function COM then associates with every schedule $\mathcal{A}$ a probability distribution on the set of all computation, the probability of a set E of computations being defined as the probability of the set of sequences of random draws ι such that $COM(\mathcal{A}, \iota)$ is in E.* "

[1] In [37] a schedule corresponds to what we call an adversary.

[2] [37] uses the notation S in place of $\mathcal{A}$, D in place of ι and C in place of ω. We use $\mathcal{A}, \iota$ and ω to be consistent with the rest of our discussion.

This approach presents well the direction to be followed in a formal analysis. Nevertheless many of the models and analyses published so far suffer from various limitations, incompleteness and sometimes mistakes that we summarize now. We use the fact that most of the existing work on randomized distributed computing can be classified into one of two classes.

To begin, there is on the one hand a very rich body of work analyzing the *semantics* and *logics* of randomized computing (cf. [2, 18, 19, 23, 28, 29, 32, 38, 47, 51, 55, 58]). The emphasis in most of these papers is to provide a unified semantics or model of computation for the *description* of randomized algorithms; and to recognize some proof rules, methods and tools allowing the *automatic verification* of some specific subclasses of algorithms. The results thus obtained are typically limited in the following three ways. A major limitation is that very little in general can be said for properties of randomized algorithms that do not hold *with probability one*: such properties are usually too specific to hold in general situations, and also usually too hard to be simple consequences of general ergodic theory. For instance, the typical problem considered by Vardi in [58] is the "probabilistic universality problem", where one checks whether a formula belonging to some temporal logic holds with probability one. (This problem can be reformulated in an equivalent way using ω-automata.) To quote [58], these methods

> "*deal only with qualitative correctness. One has often quantitative correctness conditions such as bounded waiting time* [49] *or real-time responses* [52]. *Verifications of these conditions requires totally different techniques.*"

A second limitation of these methods is that they are directed at randomized algorithms whose correctness reflects only the asymptotic properties of infinite executions. A translation of this fact is their relative success in expressing liveness properties and their failure in expressing the short-term behavior and correctness of the studied algorithms. A third limitation is that, in their generality these methods are able to take into account only very marginally the exact numeric distributions used for the random inputs. These considerations lead the authors of [38] to say that, in their model,

> "*only very basic facts about probability theory are required to prove the properties needed. Essentially one does not need anything more than: "if I throw a coin an infinite number of times then it will fall an infinite number of times on heads.*"

A second big body of work in distributed randomized computing is devoted to the design and analysis of *specific* randomized algorithms.

These algorithms typically introduce randomness into algorithms for synchronization, communication and coordination between concurrent processes (cf. [3, 37, 49, 36, 48, 50, 14, 21, 25]). *If* correct (!), these algorithms solve problems that have been proven unsolvable by deterministic algorithms [3, 37, 21, 14]. Others improve on deterministic algorithms by various measures [49, 36, 50]. The analysis and proof of correctness of these algorithms is complex as is attested by the fact that subsequent papers were published providing new proofs of the same claims (cf. [29, 47]).

In spite of successive improvements most of the analyses presented in these papers suffer from the fact that the proofs and often even the statements are not expressed formally using the probability distributions $P_{\mathcal{A}}$ and that the proofs do not track precisely how the adversaries can influence the probabilities. The paper by Aspnes and Herlihy [3] is one of the few presenting formally the correctness statement claimed. Nevertheless no explicit use of the measures $P_{\mathcal{A}}$ is made during the proof. The claims and arguments in the papers [14, 37] and [49] similarly lack a truly formal treatment. As a consequence of this lack of formalism, many proofs are effectively unreliable to readers not willing solely to rely on claims and neither willing to spend the time to reach an intuitive understanding of the algorithm.

To summarize, the limitations encountered in the papers of the first class stem from the desire to develop automatic methods of verification for a wide class of randomized algorithms: the wider the class, the less intricate each algorithm can be. In contrast, from our point of view, the question of generating the proofs of correctness of randomized algorithms can be left to the ingenuity of each prover. Instead, our ambition is to derive a *probabilistic framework* within which most if not all *mathematical analyses* and proofs related to specific randomized algorithms can be expressed. This requires two types of construction. First, we need to develop a general *model* defining algorithms and adversaries and formalizing their interactions. (We call this a general model for randomized computing.) Then, for each algorithm/adversary structure (described within the general model for randomized computing), we need to characterize the *probability spaces* used in the analysis.

To understand better the nature of the work required we describe two situations that a general model for randomized computing ought to address. We begin with a situation very rarely addressed in the literature but whose consideration places the problems encountered in the right perspective.

Lower bounds, game theory. Most of the work existing on the analysis of distributed randomized algorithms and quoted above either provides an analysis of *one* existing algorithm or provides some tools geared at providing such an analysis. All these results can be called *upper-bound* results: they show that a given problem can be solved at a given level of performance by providing an algorithm having that level of performance. On the other hand very little work has been done in the direction of *exact* lower-bounds of a randomized distributed problem and in establishing the optimality of a randomized solution to this problem. The lack of formal development in this direction is mostly a reflection of the complications involved in randomized lower bounds and of the dearth of published work in this area.[3] One exception is provided by the combined work of Graham, Yao and Karlin [25, 33] who provide precise lower bounds and a proof of optimality for the randomized (3,1)-Byzantine Generals Problem: Byzantine broadcast with 3 processes, one of which is faulty. Also, in [35], Kushilevitz et al. present some asymptotic lower bounds on the size of the shared random variable for randomized algorithms for mutual exclusion.

A general model for randomized computing ought to provide a solid framework allowing also the formal analysis of such lower bounds. In contrast with upper bound proofs, the proofs of lower bounds require a model allowing the analyses and comparison of a *family* of algorithms. These algorithms must be evaluated in relation with a family of adversaries.

As mentioned in [3], page 443, [29], page 367, [46], page 145, the situation is actually best understood in the language of game theory. The "unified measure of complexity in probabilistic computations" proposed in 1977 in [61] by Yao also adopts (implicitly) this point of view and relies fundamentally on Von Neumann's theorem in game theory. We will adopt explicitly this point of view in the sequel and let $Player(1)$ be the entity selecting the algorithm and $Player(2)$ be the entity selecting the adversary. In this language, algorithms and adversaries are strategies of respectively $Player(1)$ and $Player(2)$.

If $Player(1)$ selects the algorithm π and if $Player(2)$ selects the adversary $\mathcal{A}$, the game played by the two players consists of the alternative actions of the algorithm and the adversary: $Player(1)$ takes all the actions as described by π until the first

[3] The extreme difficulty of proving lower bounds for randomized algorithms is reflected in the fact that [33] is one of the most referenced non-published papers!

Note also that lower bounds for on-line algorithms (cf., for instance, [7, 15]) are of a very different type. These algorithms are not evaluated by their *absolute* performance but instead by some *relative* measure, usually the ratio of their performance and the performance of the best off-line algorithm. In such problems the coupling between the adversary and the algorithm is often easily analyzable, using competitive analysis. By contrast, the coupling between adversary and algorithm is much harder to analyze in general situations where absolute measures of performance are used.

point where some choice has to be resolved by the adversary; *Player*(2) then takes actions to resolve this choice as described by $\mathcal{A}$ and *Player*(1) resumes action once the choice has been resolved ... In this game, the two players have conflicting objectives: *Player*(1) selects its strategy so as to heighten the performance studied whereas *Player*(2) selects its own strategy so as to lower it. The performance in question depends on the problem studied.

For instance, in [49] the performance expresses a measure of fairness among participating processes and *Player*(2) "tries" to be unfair to some process. In [37], the performance expresses that "progress occurs fast" and *Player*(2) tries to slow the occurrence of progress. In [3], the performance is measured by the expected time to consensus and *Player*(2) chooses inputs and schedules so as to increase that expected time. Also, let us mention that adopting a game point of view provides a unifying description of the complex situation studied in [25] by Graham and Yao. This paper studies the resiliency of a certain class of algorithms under the failure of a single process. There, *Player*(2) fulfills the manifold following functions. It receives some information about the algorithm selected, selects both the initial input and the identity of the faulty process, and then monitors the messages sent by the faulty process in the course of the execution.

To finish let us remark that considering the relation between randomized algorithms and adversaries as a game between *Player*(1) and *Player*(2) is also very useful when a fixed algorithm π_0 is analyzed.[4] This situation simply corresponds to the case where *Player*(1) has by assumption a single strategy π_0.

Formalizing the notion of knowledge. It should be intuitively clear that an optimal adversary is one that will optimize at each of its steps the knowledge it holds so as to take decisions most detrimental to the performance sought by *Player*(1). Similarly, in cases where *Player*(1) has more then one strategy π to choose from, (i.e., in cases where more then one algorithm can be considered), the best strategy is one that takes best advantage of the knowledge available about the past moves of *Player*(2) and about the strategy implemented by *Player*(2).

Establishing the performance of an algorithm is always tantamount to proving that "the optimal" adversary cannot reduce the performance below the level of performance claimed for that algorithm. Equivalently, a proof of correctness must in some way establish a bound on the usefulness of the knowledge available to *Player*(2).

This justifies that a general probabilistic model for randomized computing should formalize the notion of knowledge available to the players. And that it should

[4] This was actually the original insight of [3, 29, 46].

provide an *explicit mechanism of communication* between the players, formalizing the update of their knowledge as the execution progresses.

To illustrate this point of view note that in all the formal constructions quoted above and associated with the analysis of a specific algorithm (see for instance [3, 28, 29, 40, 47, 58]) the model proposed for an adversary formalizes that *Player*(2) makes its choices *knowing the complete past execution*: this was indeed the idea of Rabin in [37] quoted above. Nevertheless the correctness of some published algorithms depends critically on the fact that *Player*(2) is allowed only a partial knowledge of the state of the system and/or has only a finite memory of the past. The example of Rabin's randomized algorithm for mutual exclusion [49] is very interesting in this respect. ([49] is one of the few papers venturing into that area.) Due to the lack of model, Rabin resorts to intuition to present and establish the main correctness property. We quote:

> "*In the context of our study we do not say how the schedule arises, or whether there is a mechanism that imposes it ... We could have an adversary scheduler who tries to bring about a deadlock or a lockout of some process P_i. A special case of this evil scheduler is that some processes try to cooperate in locking out another process. Our protocol is sufficiently robust to have the desired property of fairness for every schedule S.*
>
> *We have sufficiently explained the notion of protocol to make it unnecessary to give a formal definition: given a protocol π we now have a natural notion of a run $\alpha = i_1 X_1 i_2 X_2 \ldots$, resulting from computing according to π. Again we do not spell out the rather straightforward definition. Note that since process i may flip coins, even for a fixed schedule S there may be many different runs α resulting from computing according to π.* "

As mentioned previously, the notion of *schedule* used in [49] corresponds to our notion of adversary. Also, the notion of *run* used in [49] corresponds to what we call the *knowledge* held by *Player*(2). As demonstrated in Chapter 3 of this thesis, a rigorous proof of the correctness of the algorithm presented in [49] should have actually required a formal model stating clearly "how the schedule arises" and also formalizing the interaction between *Player*(1) and *Player*(2): after having formalized the setting of [49], we establish that the knowledge at the disposal of *Player*(2) is much stronger then what was originally believed, and that the algorithm is *not* "sufficiently robust to have the desired property of fairness for every schedule S" as claimed in [49].

The reason for the failure of the proof given in [49] is that it does not take into

account the knowledge available to *Player*(2). Instead it argues solely in terms of the distribution of the random inputs; i.e., attempts to prove properties of the executions $\omega(\mathcal{A}, \iota)$ by simply considering the random inputs ι. Again, this means overlooking the power of *Player*(2) in influencing the distribution of $\omega(\mathcal{A}, \iota)$.

To summarize, we have argued that a general model for randomized computing should allow the simultaneous consideration of several algorithms: such situations are typically encountered while proving lower bounds. It is natural to view the situation as a game between *Player*(1), the algorithm designer, and *Player*(2) the adversary designer. The algorithms are the strategies of *Player*(1) and the adversaries are the strategies of *Player*(2). This model should provide an *explicit mechanism of communication* between the two players allowing them to update their knowledge as the execution progresses.

The goal of this thesis is to present such a model and to illustrate its generality on several examples. We now present the work done in the thesis.

We present in Chapter 2 a general model corresponding to the previous discussion. The model is simply constructed to formalize that both players take steps in turn having only a partial knowledge of the state of the system. This knowledge is updated at every move of either player. We then construct carefully the "natural" probability space $(\Omega_{\pi,\mathcal{A}}, \mathcal{G}_{\pi,\mathcal{A}}, P_{\pi,\mathcal{A}})$ obtained on the set of executions when the algorithm is a given π and when the adversary is a given $\mathcal{A}$. Our construction requires the technical (but fundamental) hypothesis that all the coins used in the game have at most countably many outcomes.

To our knowledge, this last probabilistic construction was similarly never conducted to completion. In [58] and [29] Vardi and Hart et al. present the σ-field $\mathcal{G}_{\pi,\mathcal{A}}$ that allows to study probabilistically events "depending on finitely many conditions". Nevertheless some additional work has to be devoted to justify the existence of a probability measure $P_{\pi,\mathcal{A}}$ on the set of executions. We prove this existence by a limiting argument using Kolmogorov's theorem.[5]

[5] Aspnes and Herlihy in [3] define formally the measure $P_{\mathcal{A}}$ to be the image measure of the measure on the random inputs under the mapping: $\iota \rightarrow \omega(\mathcal{A}, \iota)$, where ι denotes a generic sequence of random inputs and $\omega(\mathcal{A}, \iota)$ is the unique execution corresponding to a given adversary $\mathcal{A}$ and a given sequence of random inputs ι. Indeed, the σ-fields are defined precisely so that the mapping $\iota \rightarrow \omega(\mathcal{A}, \iota)$ is measurable. But this approach cannot be used in general situations as it presupposes a well defined probability measure on the set of random inputs ι. This is trivially the case when the sequence of inputs is obtained with *independent* coins, in which case the space of random draws ι is endowed with the product structure. This property holds for many published randomized algorithms such as those in [3, 37]. Nevertheless this is not the case in [4] which considers a situation where the weight of the coins used is modulated along the execution. Also, this is not

An important feature of our model is the symmetry existing between the notions of algorithm and of adversary. Eventhough this symmetry is rather natural in the light of the game theory interpretation it nevertheless seems to be rather novel with respect to the existing work. Most of the models presented so far are concerned with the analysis of a single algorithm, i.e., when *Player*(1) has a single strategy. This very specific (eventhough important) situation obscures the natural game theory structure that we reveal and led various authors to asymmetric models where the adversaries depend specifically on the algorithm considered. As discussed later, our Chapter 7 provides a striking illustration in favor of a symmetric model.

Our Chapter 3 analyzes Rabin's randomized algorithm for mutual exclusion [49]. This algorithm is one of the few in the literature whose correctness hinges critically on the limited knowledge available to *Player*(2).

As mentioned above, the study of such algorithms is rather complex and requires the exact formalization of the notion of knowledge. In the absence of a formal model for randomized computing Rabin resorted to intuition in his original paper [49] to express and establish the intended correctness statement. One year later, in 1983, as an illustration of their general method, Hart, Sharir and Pnueli provided in [29] some additional "justifications" to the correctness of the algorithm.

As part of our analysis we first show how the formal model of Chapter 2 allows one to formalize accurately the hypotheses. Our analysis then reveals that the correctness of the algorithm is tainted for two radically different reasons. We first show that the informal statement proposed by Rabin admits no natural "adequate" formalization. The problem is in essence the following. The statement involves proving that the probability of a certain event C is "high" against all adversaries if a certain precondition B holds. One natural way to formalize such a statement might seem to consider the expression

$$\inf_{\mathcal{A}} P_{\mathcal{A}}[C \mid B]$$

Nevertheless we show that this would be tantamount to giving the knowledge of B to *Player*(2). But *Player*(2) would not be able to derive the knowledge of B from the mere information sent to him in the course of the execution: *Player*(2) "learns"

---

the case in the complex paper [25] where a random adversary adapts its moves based on the past execution. Finally, this is not also the case in the general models in [29, 58], where the i-th coin used depends on the state s_i of the system at the time of the i-th flip.

Note nevertheless that the argument of Aspnes and Herlihy in [3] is now valid in the light of our construction in Chapter 2 which precisely justifies the existence of a probability distribution on the sequence of flips even when these flips are not independent.

some non-trivial fact from the conditioning on B. Not surprisingly $Player(2)$ can then defeat the algorithm if this measure is used.

This first problem is not related to the proof of the correctness statement but "merely" to its formalization. Our analysis shows that such formalization problems are general and proposes a partial method to formalize adequately informal high-probability correctness statements.

We then show that the algorithm suffers from a "real" flaw disproving a weaker but adequate (in the sense just sketched) correctness measure. The flaw revealed by our analysis is precisely based on the fact that the dynamics of the game between the two players allow $Player(2)$ to acquire more knowledge then a naive analysis suggests.

To finish we establish formally a correctness result satisfied by Rabin's algorithm. The method we apply is rather general and proceeds by successive inequalities until derivation of a probabilistic expression depending solely on the random inputs. Such an expression is independent of $Player(2)$ and its estimation provides a lower bound.

Our chapter 4 analyzes Lehmann-Rabin's Dining Philosophers algorithm [37]. In spite of its apparent simplicity this algorithm is not trivial to analyze as it requires, as usual, to unravel the complex dependencies between the random choices made by the algorithm and the non-deterministic choices made by $Player(2)$. The original proof given in [37] does not track explicitly how the probabilities depend on the adversary $\mathcal{A}$ and is therefore incomplete. In a subsequent paper [47], Pnueli and Zuck provides a proof of eventual correctness of the algorithm. The method used there shares the characteristics of most general semantic methods described on page 14: it establishes that some event eventually happens with probability one. Furthermore it does not take into account the specific probability distribution used for the random inputs.

We present instead a new method in which one proves auxiliary statements of the form $U \xrightarrow[p]{t} U'$, which means that whenever the algorithm begins in a state in set U, with probability p, it will reach a state in set U' within time t. A key theorem about our method is the composability of these $U \xrightarrow[p]{t} U'$ arrows allowing to use each of these results as building blocks towards a global proof. (This part was developed jointly with Roberto Segala.) Our method presents the two following advantages.

It first provides a tighter measure of progress then "eventual progress": we provide a finite time t and a probability p such that, for all adversaries, progress happens within time t. This trivially implies the eventual progress with probability-one.

As mentioned in page 14, eventual probability-one liveness properties are usually considered not because of their intrinsic relevance but because of the limitations of the general methods used. Our method uses more specific and refined tools then the very very basic facts about probability theory as "if I throw a coin an infinite number of times then it will fall an infinite number of times on heads" mentioned by Lehmann and Shelah in [38].[6] Nevertheless it is still general and simple enough to apply to a variety of situations and tighten the liveness measures usually derived.

An additional advantage of our method is that it does not require working with events belonging to the tail σ-field, an enterprise which can present some subtle complications. Recall in effect that the argument proposed by Lehmann-Rabin consisted in conditioning on the fact that no progress occurred and then deriving a contradiction. Our discussion at the beginning of Chapter 3 shows that conditioning on any event – let alone conditioning on an event from the tail σ-field – can be problematic.

Our Chapters 5 and 7 are concerned with a scheduling problem in presence of faults. Chapter 5 investigates the deterministic case – where algorithms do not use randomization – and Chapter 7 investigates the randomized case. In both situations we are interested in providing optimal algorithms. Both are rather complex even-though in very different ways. The solution of Chapter 5 is obtained by translating the problem into a graph problem. A key tool is Ore's Deficiency Theorem giving a dual expression of the size of a maximum matching in a bipartite graph.

In Chapter 7 we describe a randomized scheduling algorithm and establish formally its optimality. To our knowledge, Graham and Yao were before us the only ones providing the proof of the *exact* optimality of a randomized algorithm.

The method that we develop to prove optimality is rather general and in particular encompasses in its scope the specific proof strategy used by Graham and Yao in their paper [25]. This method is presented by itself in Chapter 6. It is in essence based on an application of a min-max equality reversing the roles of $Player(1)$ and $Player(2)$ in the scheduling game considered. This min-max method uses critically the natural symmetric structure between the two players. The success of this point of view also brings a striking illustration of the relevance of a general symmetric model for randomized computing as claimed at the beginning of this chapter. In particular, critical to the proof is the fact that adversaries are randomized and defined independently of any specific algorithm, much in the same way as algorithms are randomized and defined independently of any specific adversary.

[6] See page 14 of this chapter for a more complete quote.

Eventhough the proof of [25] and our proof are both applications of the general proof method presented in Chapter 6, the two proofs differ in an essential way. We comment on this difference to give an account of the complexities involved in such proofs. Critical to the proof of Graham and Yao [25] is the fact that, in their model, *Player*(2) knows explicitly the strategy (i.e., the algorithm) used by *Player*(1): their proof would not hold without this assumption. On the other hand, in the formal model that we use, *Player*(2) is *not* explicitly provided with the knowledge of the strategy used by *Player*(1), and our proof would not hold if it was.

Nevertheless, as is argued in Chapter 6, in both [25] and in our work, the optimality of an algorithm does *not* depend on the fact that *Player*(2) knows or not the algorithm under use. Such a fact merely affects the shape taken by the proof. This subtle point should make clear, we hope, that formal proofs of optimality are bound to be very complex.

We describe summarily the content of Chapter 6. This chapter presents a general proof method to attempt to prove that a given algorithm is optimal. At the core of the method is the use of the min-max equality

$$\max_{\pi} \inf_{\mathcal{A}} f(\pi, \mathcal{A}) = \min_{\mathcal{A}} \sup_{\pi} f(\pi, \mathcal{A}) \,,$$

where $f(\pi, \mathcal{A})$ is the performance of the algorithm π against the adversary $\mathcal{A}$. Fundamental to us are therefore situations where such an equality occurs: in each of these cases our method yields the possibility to prove the optimality of an algorithm. We show that this equality occurs in the two following cases. (In the first case the equality is a simple reinterpretation of Von Neumann's theorem.)

1. When the strategies of either player are the convex combinations of a finite set called the set of pure strategies (and when $f(\pi, \mathcal{A})$ is the expected value $E_{\pi,\mathcal{A}}[T]$ of some random variable T.)
2. When one player, typically *Player*(2), "knows" the strategy used by the other player.

The two settings yield different proof systems. Very interestingly our proof of optimality in Chapter 7 falls in case 1 whereas the proof of optimality [25] of Graham Yao falls in case 2.

To summarize, we present in this thesis a formal and general model for randomized computing. Under the assumptions that all the coins have at most countably many

outcomes we succeed in constructing formally the probability spaces on the set of executions to be used in the analysis of an algorithm. To our knowledge, our model is stronger then all models previously constructed in that 1) it accurately allows to formalize the precise nature of the knowledge held by both players (this is the critical point in Rabin's randomized algorithm for mutual exclusion); 2) it allows to conduct formal lower-bound proofs for randomized computing (no formal model existed so far allowing that[7]); and 3) it does not require as in the models presented by Vardi in [58] and Hart et al. in [29] that the algorithms be finite-state programs. (Actually our model allows for algorithms having a state space with the cardinality of the continuum). Our Chapters 3 and 7 illustrate the two first points. The last one is a simple consequence of the fact that our model allows the algorithm to have an infinite memory.

In Chapter 3 we present the notion of adequate performance measures for an algorithm: measures that are "naturally" attached to an algorithm and provide meaningful estimations. We illustrate this notion in our analysis of Rabin's algorithm for mutual exclusion [49]. We furthermore provide a rather general technique for proving rigorously high-probability statements holding against all adversaries.

In Chapter 4 we provide a rather general technique for proving upper bounds on time for randomized algorithms. We illustrate this technique in our analysis of Lehmann-Rabin's Dining Philosopher's algorithm [37].

In Chapter 6 we present a general proof methodology to attempt to prove that a given algorithm is optimal. We illustrate this method in Chapter 7 with a specific example.

[7] The model developed by Graham and Yao is an ad-hoc model for the specific situation pertaining to [25].

Chapter 2

A General Model for Randomized Computing

We argued in Chapter 1 that formal proofs of correctness of randomized algorithms required a formal model for randomized computing. Such a model should formalize the notion of algorithm and of adversary, and formalize how these two entities interact.

We also argued that the game-theory point of view was most appropriate to understand and model these notions. For this, we let $Player(1)$ be the entity selecting the algorithm and $Player(2)$ be the entity selecting the adversary. In this language, algorithms and adversaries are strategies of respectively $Player(1)$ and $Player(2)$. We therefore sometimes call $Player(1)$ the algorithm-designer and $Player(2)$ the adversary-designer. If $Player(1)$ selects the algorithm π and if $Player(2)$ selects the adversary $\mathcal{A}$, the game played by the two players consists of the alternative actions of the algorithm and the adversary: $Player(1)$ takes all the actions as described by π until the first point where some choice has to be resolved by the adversary; $Player(2)$ then takes actions to resolve this choice as described by $\mathcal{A}$ and $Player(1)$ resumes action once the choice has been resolved ... This means that the two players play sequentially.

The purpose of this chapter is to construct both 1) such a general model for randomized computing and 2) the associated probability spaces used for the analysis of randomized algorithms. Our model is presented in Section 2.3. The construction of the associated probability spaces is presented in Section 2.4. Section 2.1 investigates the features that a general model should be endowed with. Section 2.2 motivates the formal presentation given in Section 2.3.

2.1 Which Features should a General Model have?

We argue here that a model for randomized computing should have the following general features. 1) It should not only allow to analyze the performance of a given algorithm but the performance of a whole class of algorithms. 2) It should formalize the notions of both an adversary and of an algorithm: for emphasis, the algorithms and adversaries thus formalized are called *admissible*. 3) It should allow the adversaries to be randomized. 4) It should allow to formalize that both *Player*(1) and *Player*(2) have in general only a partial knowledge of the state during the execution. In particular it should provide an explicit mechanism of communication between the two players allowing them to update their knowledge as the execution progresses. And 5), an admissible adversary should be characterized independently of the choice of any specific admissible algorithm.

1. & 2. Concurrent analysis of several algorithms; Formalization of the notion of algorithm. As mentioned in Chapter 1 most of the existing models for randomized computing (e.g. [3, 28, 29, 47, 54, 58]) are implicitly designed for the analysis of *one* algorithm. In contrast we have in mind a general model in which all proofs and arguments about randomized algorithms could be expressed. Our model must in particular be suited for the formalization of lower-bound proofs. In that case the analysis considers not only one, but a whole family Π of algorithms. This family must be characterized much in the same way as the family A of adversaries must be characterized. Note that characterizing the family Π corresponds to modeling the notion of algorithm. In all the papers cited above and analyzing a fixed algorithm, the necessity to model an algorithm was not felt: more exactly the description of the algorithm analyzed was the only modelization required. This situation corresponds to the case where Π is reduced to a singleton $\{\pi\}$.

3. Randomized adversaries. We now turn to the third point and argue that a general model should allow the adversary to be randomized.

It is often claimed that, in the absence of cryptographic hypotheses, the analysis can "without loss of generality" consider only non-randomized adversaries. For instance in [29] the authors say and we quote:

> "*Note that the schedule's "decisions" are deterministic; that is, at each tree node a unique process is scheduled. One might also consider more general schedules allowing the schedule to "draw" the process to be scheduled at each node using some probability distribution (which may depend on the particular tree node). However ... there is no loss of generality in considering deterministic schedules only.*"

The reason is that, whenever making a random choice among a set D, an optimal *Player*(2) can instead analyze the outcomes corresponding to all the choices d in D, (more exactly to all the measurable choices), and choose one that best lowers the performance of the algorithm. (More exactly, can select one that brings the performance of the algorithm arbitrarily close to the infimum "$\inf_{d \in D}$*performance under choice of* d".)

The above argument shows that a model allowing the adversaries to be randomized does not make *Player*(2) more powerful. Nevertheless we do not share the more restrictive view expressed in [13]:

> "*The motivation for models where the adversary is assumed to exhibit certain probabilistic behavior is that the worst case assumptions are frequently too pessimistic, and phenomena like failures or delays are often randomly distributed.*"

Randomization can undoubtedly be useful in a setting where one wants to weaken the power of *Player*(2). But, even when considering an "unweakened" *Player*(2) allowing *Player*(2) to use randomization can be of inestimable help in the proof of optimality of a given (optimal) randomized algorithm π_0. Indeed, the proof methodology – for establishing the optimality of a randomized algorithm – presented in Chapter 6 of this thesis requires to provide a specific adversary and to prove that this adversary satisfies an optimal property[1]. (Recall that an adversary is a strategy of *Player*(2).) As just mentioned, randomization does not make *Player*(2) more powerful and the existence of such an optimal randomized adversary therefore implies the existence of a *deterministic* optimal adversary. But, in general, the sole description of such a deterministic optimal adversary can prove to be a very hard task (e.g., taking non constant space – even taking exponential space), therefore barring the possibility to establish the optimality of an algorithm – at least using our proof methodology. On the other hand, if the adversary is allowed to use randomization, we can in some instances provide the description of a "simple" optimal randomized adversary, thus also proving the optimality of the algorithm π_0.

An illustration of this phenomenon is given in the proof of optimality given by Graham and Yao in [25]. A close analysis of the proof of [25] shows that it follows the general methodology given in our Chapter 6, introduces a specific adversary $\mathcal{A}_0$ and proves that it verifies an optimal property. A fundamental assumption of the model used in [25] is that the *Player*(2) "knows" explicitly the algorithm under use. This knowledge is used critically to define the strategy $\mathcal{A}_0$: at every point of the execution, *Player*(2) determine its next step by emulating π under

[1] The desired notion of optimality for the adversary will be clarified in Chapter 6.

certain conditions. As π is randomized, the emulation of π also requires the uses of randomness and $\mathcal{A}_0$ is therefore a randomized adversary.

This discussion justifies that a general model should allow the adversary to use randomness. We will provide in Chapter 7 of this work another application of this fact.

4. Formalization of the notion of knowledge. We now turn to the fourth point and argue for a model formalizing the notion of *knowledge*. The model should also provide an *explicit mechanism of communication* between the algorithm and the adversary, allowing them to update their knowledge as the execution progresses.

It should be intuitively clear that an optimal strategy of *Player*(2) is one where *Player*(2) optimizes at each of its steps the knowledge it holds so as to take decisions most detrimental to the performance sought by *Player*(1). Similarly, in cases where *Player*(1) has more then one strategy π to choose from, (i.e., in cases where more then one algorithm can be considered), the best strategy is one that takes best advantage of the knowledge available about the past moves of *Player*(2) and about the strategy implemented by *Player*(2).

Establishing the performance of an algorithm is always tantamount to proving that *Player*(2) adopting an optimal strategy cannot reduce the performance below the level of performance claimed for that algorithm. Equivalently, a proof of correctness must in some way establish a bound on the usefulness of the knowledge available to *Player*(2). Similarly, to establish the optimality of an algorithm one is led to show that no other admissible algorithm can use more efficiently the knowledge available to *Player*(1).

This justifies that a general probabilistic model for randomized computing should formalize the notion of knowledge available to the players. And that it should provide an explicit mechanism of communication between the players, formalizing the update of their knowledge as the execution progresses.

5. An adversary can be associated to any algorithm. We now turn to the fifth point and argue that, in a general model for randomized computing, an admissible adversary should be characterized independently of any *specific* admissible algorithm. Note first that, by definition, an algorithm π is defined independently of a given adversary $\mathcal{A}$. On the other hand, an adversary might seem to be defined only in terms of a given algorithm: the adversary is by definition the entity that resolves all choices not in the control of the algorithm considered. We show over an example that this conception is incorrect and that a correct game theory interpretation yields

adversaries defined independently of the algorithm considered.

A situation where one can encounter several algorithms in the course of a correctness proof is one where, as in Chapters 3 and 4 of this thesis, a program $\mathcal{C}$ is studied for various initial conditions $s_i, i \in I$. One can then model an algorithm to be a couple $(\mathcal{C}, s_i)$: we have a different algorithm π_i for each different initial condition s_i.[2] The code $\mathcal{C}$ is considered to "be correct" (with respect to the specifications of the problem considered) if all algorithms behave well against *Player*(2). Note that, in this situation, one implicitly assumes that *Player*(2) "knows" which algorithm π_i is under use. A strategy $\mathcal{A}$ for *Player*(2) (i.e., an adversary) is then accurately defined to be a family $(\mathcal{A}_i)_{i\in I}$, one for each algorithm π_i. We will say that $\mathcal{A}_i$ is an *adversary specially designed for* π_i.[3] The adversary $\mathcal{A} = (\mathcal{A}_i)_{i\in I}$ is clearly not associated to a specific algorithm π_i.

More generally one will define an adversary to be a strategy of *Player*(2) taking into account *all* the information available during the execution. (In the previous example one assumed that *Player*(2) was told the algorithm selected by *Player*(1).) We thus obtain a symmetric model where algorithms π in Π (resp. adversaries $\mathcal{A}$ in A) are defined independently of any choice of $\mathcal{A}$ (resp. of any choice of π). Because they are dealing only with the special case where *Player*(1) has only one strategy, many of the models published have developed in ways obscuring this natural symmetry between *Player*(1) and *Player*(2), i.e., between adversaries and algorithms. (The model presented in [40] suffers of this flaw.) This represents more than an esthetical loss. For, as we show in Chapter 6 of this thesis, the symmetry of the two players is expressed by a max-min equality and plays a central role in the proof of optimality of a randomized algorithm.

Note that the properties 1, 2, 3 and 5 claimed for a general model – the possibility to analyze and compare several algorithms, the necessity to formalize the notion of an algorithm, the allocation of randomness to the adversary and the possibility to characterize an admissible adversary independently of any specific admissible algorithm – are useful mostly for lower-bounds. This explains why none of the models considered so far in the literature included these features.

On the other hand, the fourth claim – that the notion of knowledge is crucial for proofs of correctness and should be explicitly incorporated in a model – is very relevant to upper-bounds. Proofs that do not approach formally the notion of knowledge are walking a very slippery path, to use a term first coined in [37] and then often

[2] We will review this case in Section 2.2.

[3] We will come back in more detail to this notion in Chapter 6.

quoted. An illustration of this fact is, as we will see, the algorithm for mutual exclusion presented in [49] whose correctness property hinges critically on the limited knowledge available to *Player*(1).

2.2 Towards the Model

The previous section outlined the features that a general model for randomized computing should be endowed with. This section analyzes in a first part various equivalent ways to construct a model and motivates in a second part the formal definition presented in Section 2.3.

2.2.1 Equivalent ways to construct a model

We illustrate here how equivalent models for randomized computing can be obtained by exchanging properties between the model used for an algorithm and the model used for an adversary.

A first difficulty encountered when modeling the *notions* of adversary and algorithm is that these two notions cannot be defined independently.[4] As a consequence, some properties can be exchanged between the model used for an algorithm and the model used for an adversary. This can lead to develop different but equivalent models.

We illustrate this fact with three examples. The first example expands on the example presented in Section 2.1 and shows that equivalent models can be achieved by allocating to *Player*(1) or to *Player*(2) the choice of the initial configuration. The second example shows that, when considering timed algorithms, equivalent models can be achieved by allocating to *Player*(1) or to *Player*(2) the control of the time. The third example shows that, in the case where *Player*(2) is assumed to know the past of an execution, equivalent models can be achieved by allocating to *Player*(1) or to *Player*(2) the control of the random inputs to be used next by *Player*(1).

Consider first the case of Lehmann-Rabin's algorithm presented in [37] and studied in Chapter 4 of this work.

The correctness property of the algorithm expresses that "progress occurs with

[4] This is not in contradiction with point 5 above stating that "an admissible adversary should be characterized independently of any specific admissible algorithm". We speak here of the way we define the sets Π and A of algorithms and adversaries. We spoke in point 5 of the way a given element $\mathcal{A}$ in A is characterized.

"high" probability whenever a process is in its Trying section". Let $\mathcal{C}$ be the program described in [37] and recalled in page 101. A possible way to model the situation is to consider that an algorithm π is defined by the conjunction $(\mathcal{C}, s_{init})$ of $\mathcal{C}$ and of the initial configuration s_{init}. We then derive a family of algorithms, one for each initial configuration s_{init}. In this case, an adversary is a strategy guiding the moves of *Player*(2) against all such possible algorithms $(\mathcal{C}, s_{init})$: by assumption *Player*(2) learns which algorithm $(\mathcal{C}, s_{init})$ is selected by *Player*(1) at the beginning of the execution. Another possible way to model the situation is to assume the existence of a single algorithm, namely $\mathcal{C}$, and to let *Player*(2) select the initial configuration.

This modeling duality is similarly encountered in the consensus problem studied in [3], in the mutual exclusion problem studied in [49] and more generally in all situations where the initial configuration or input is not a priori determined. In [61] Yao specifically considers this situation.

As a second example we now consider the timed version of the same Lehmann-Rabin's algorithm [37]. In this situation we restrict our attention only to executions having the property that "any participating process does not wait more then time 1 between successive steps". The control of the time can be allocated in (at least) two different ways resulting in two different models. One way is to allocate the time control to *Player*(1): each timing policy such that no participating process waits more then time 1 for a step corresponds to a different admissible algorithm. Another solution is to assume instead that *Player*(2) controls the passing of time: an adversary is admissible if it does not let time pass without allocating a step to a process having waited time 1. This last solution is the one adopted in Chapter 4 of this work.

We provide another less trivial example of the possible trade-off between the notion of adversary and the notion of algorithm. This example shows that the two models for randomized concurrent systems of Lynch et al. [40] and Vardi [58], page 334, are equivalent. We summarize here the model presented in [40]. (See Chapter 4 and [54] for more details.) In this model a randomized algorithm is modeled as a *probabilistic automaton*:

Definition 2.2.1 A *probabilistic automaton* M consists of four components:

- a set $states(M)$ of states
- a nonempty set $start(M) \subseteq states(M)$ of start states
- an action signature $sig(M) = (ext(M), int(M))$ where $ext(M)$ and $int(M)$ are disjoint sets of external and internal actions, respectively

- a transition relation $steps(M) \subseteq states(M) \times acts(M) \times Probs(states(M))$, where the set $Probs(states(M))$ is the set of probability spaces $(\Omega, \mathcal{G}, P)$ such that $\Omega \subseteq states(M)$ and $\mathcal{G} = 2^{\Omega}$.

An *execution fragment* α of a probabilistic automaton M is a (finite or infinite) sequence of alternating states and actions starting with a state and, if the execution fragment is finite, ending in a state, $\alpha = s_0 a_1 s_1 a_2 s_2 \cdots$, where for each i there exists a probability space $(\Omega, \mathcal{G}, P)$ such that $(s_i, a_{i+1}, (\Omega, \mathcal{G}, P)) \in steps(M)$ and $s_{i+1} \in \Omega$.

An *adversary* for a probabilistic automaton M is then defined to be a function $\mathcal{A}$ taking a finite execution fragment of M and giving back either nothing (represented as $\perp$) or one of the enabled steps of M if there are any.

A quick interpretation of this model is as follows. A step $(s, a, (\Omega, \mathcal{G}, P))$ in $steps(M)$ represents a step of the adversary. During an execution, a step $(s, a, (\Omega, \mathcal{G}, P))$ can be selected by the adversary as its t-th selection only if the state s_{t-1} of the underlying system is s. (Note that this condition implicitly assumes the existence of a mechanism allowing *Player*(2) to "know" precisely what the state of the system is. Furthermore the definition of the adversary as a function of the whole past execution fragment means that the *Player*(2) "remembers the past" and chooses the successive steps based on this knowledge. This precise model therefore does not apply to cases where, as in [49], *Player*(2) does not have access to the full knowledge of the state of the system.) The second field a and the third field $(\Omega, \mathcal{G}, P)$ of the step of the adversary characterize the step to be taken next by the algorithm: this step corresponds to the action a and consists in choosing randomly an element s_t in Ω according to the probability distribution P.

Consider the case where, for every (s, a) in $S \times A$, there is a *fixed* probability space $(\Omega_{s,a}, \mathcal{G}_{s,a}, P_{s,a})$ such that, for every step in $steps(M)$, if the state is s and the action is a, then the associated probability space is $(\Omega_{s,a}, \mathcal{G}_{s,a}, P_{s,a})$.[5] In this case, we can change the model of [40] into an equivalent model by making the following changes. We model a randomized algorithm to be a family $((\Omega_{s,a}, \mathcal{G}_{s,a}, P_{s,a}))_{(s,a) \in S \times A}$ of probability spaces. Following the idea presented in [58], page 334, we redefine

[5] The model presented in [40] does not always assume this fact: this model is a hybrid where the adversary *does* decide at each step what is the probability space to be used next by the algorithm, but where this space might not be uniquely characterized by the action a selected. This means in essence that the set of actions, $sig(M)$, is not big enough to describe accurately the set of decisions taken by *Player*(2). In order to do so we need to refine the set of actions. Doing this allows one to get a one-to-one correspondence $(s, a) \in S \times A \rightarrow (\Omega_{s,a}, \mathcal{G}_{s,a}, P_{s,a})$ and hence reduces the model of [40] to our situation.

the notion of adversary by saying that an adversary is a function $\mathcal{A}$ taking a finite execution fragment $s_0, a_1, s_1 \ldots$ of M and giving back either nothing (represented as $\perp$) or one of the enabled actions of M if there are any. (This action is said to be "decided" by the adversary.)[6] Furthermore, if *Player*(2) decides action a while the state is equal to s then the algorithm takes its next step by choosing randomly an element s in $\Omega_{s,a}$ according to the probability distribution $P_{s,a}$. Hence the family $((\Omega_{s,a}, \mathcal{G}_{s,a}, P_{s,a}))_{(s,a)\in S\times A}$ of probability spaces can be interpreted as being the *local dynamics* of the algorithm. In this model an algorithm is therefore identified with its local dynamics. We will expend on this theme later in Section 2.3.

It is easy to convince oneself that this model is equivalent to the one of [40]. This provides another example of a possible trade-off in the model of the algorithm and in the model of the adversary: in essence, the difference lies in whether *Player*(1) or *Player*(2) characterize the probability space $(\Omega, \mathcal{G}, P)$ – the local dynamics – to be used next by the algorithm. In [40] the local dynamics of the algorithm are specified in the steps taken by the adversary. In the alternative model that we just outlined these local dynamics are given separately and define the algorithm.

To summarize, the previous discussion illustrates the fact that a model for the analysis of randomized algorithms requires the simultaneous modeling of an algorithm and of an adversary; and that various equivalent models can be derived by trading some properties between the model of an algorithm and the model of an adversary. Furthermore, in our discussion about the model of [40], we showed how we could define an algorithm by its *local dynamics*. We also saw that a limitation of this model is that it pre-supposes that *Player*(2) "knows" completely the state of the system. This does not fit situations as the one encountered in Rabin's algorithm for mutual exclusion that we study in Chapter 3 of this work, where *Player*(2) has by assumption only a partial knowledge of the state of the system.

[6] The model considered by Vardi in [58] does not have a set of actions but solely a state space: the evolution of the system is defined by a concurrent Markov chain. We show here that our redefined notion of adversary coincides with the notion of *scheduler* in [58].

In [58], from a given state u, a step of the scheduler determines fully the next state v. This state determines in turn the probability distribution to be used for the next step.

On the other hand, in our modified version of [40], the adversary determines the next action a. This, along with the previous state s determines uniquely the probability distribution to be used next.

2.2.2 Motivations for Section 2.3

Recall that, if *Player*(1) selects the algorithm π and if *Player*(2) selects the adversary $\mathcal{A}$, the game played (i.e., the unfolding of the execution) consists of the alternative actions of the algorithm and the adversary: *Player*(1) takes all the actions as described by π until the first point where some choice has to be resolved by the adversary; *Player*(2) then takes actions to resolve this choice as described by $\mathcal{A}$ and *Player*(1) resumes action once the choice has been resolved... This means that the two players play sequentially, each move being randomized. Consider a player's random move and let R denote the outcome of this random move: R is a random variable. Let $(\Omega, \mathcal{G})$ be the space where R takes values and let P be the probability law $\mathcal{L}(R)$ of R.[7]

Recall also that our goal is not to construct a computational model that would describe the implementation of randomized algorithms, but, instead, to derive a probabilistic model allowing their analysis. In this perspective, two different implementations of a player's move leading to the same probabilistic output are indistinguishable: in probabilistic terms, random values having the same probabilistic law are indistinguishable. This means that the probability space $(\Omega, \mathcal{G}, P)$ gives all the probabilistic information about the move of the player and we can for instance assume that this move consists in drawing a random element of Ω with probability P.[8]

By definition, a strategy of a player is a description of how the player is to take all its moves. Each move being described by a probability space $(\Omega, \mathcal{G}, P)$, a strategy can be modeled as a family $(\Omega_x, \mathcal{G}_x, P_x)_{x \in X}$ of probability spaces. The set X is the set of different *views* of the system that the player can hold upon taking a move. This set depends on the assumptions done about the information conveyed to the player during the course of an execution, and about the memory of the past moves allowed to this player.

To motivate this notion we discuss quickly the case of Rabin's randomized algorithm for mutual exclusion presented in Chapter 3, the case of Lehmann-Rabin's randomized dining-philosophers algorithm presented in Chapter 4 and the case of the randomized scheduling problem presented in Chapter 7. The reader might not be familiar with these problems at this point. Nevertheless, even in a first reading,

[7] See Definition 8.1.2, Page 198, for a definition of the *law* of a random variable.

[8] There is a little subtlety here. "Drawing a random element ω of Ω with probability P" does *not* imply that, for every $\omega \in \Omega$, the singleton $\{\omega\}$ is measurable, i.e., is a set in $\mathcal{G}$. (For instance, in the case where $\emptyset$ is the only element of $\mathcal{G}$ having zero P-measure, if B is an atom of $\mathcal{G}$ (see Definition 8.1.1) and if ω is in B, then $\{\omega\} \in \mathcal{G}$ only if $B = \{\omega\}$.) Nevertheless this will be the case in the sequel as we will assume that Ω is countable and that $\mathcal{G}$ is the discrete σ-field $\mathcal{P}(\Omega)$.

the following description provides a good intuition of the issues involved with the notion of view.

In Lehmann-Rabin's algorithm [37], *Player*(2) knows at each time the full state of the system and remembers the past execution fragment. Its set X of views is therefore the set of execution fragments. (The notion of execution fragment is formally presented in Chapter 4.) By contrast, in Rabin's randomized algorithm for mutual exclusion [49], *Player*(2) does not see the full state of the system, but, instead, only sees the value of the program-counter pc_i of each process i: in particular *Player*(2) does not see the values of the various program variables.[9] Here also it remembers everything it learns. The set of possible views held by *Player*(2) upon making a move is therefore the set of finite sequences $(i_1, pc_{i_1}, \ldots, i_k, pc_{i_k})$, where $(i_1, \ldots, i_k)$ is a sequence of process-id's.

Consider now the case of *Player*(1) (for the same algorithm [49]). After each of its moves, *Player*(1) only remembers the current state s of the program: s is determined by the values of the program counters and of the program variables. Before each of its moves it furthermore learns from the adversary which process i is to take a next step. Its set of views is therefore the set $I \times S$, where I is the set of process-*id*'s and S is the state space of the system. (We can assume that the field i of the view of *Player*(1) is erased, i.e., reset to #, after each move of *Player*(1).)

In the scheduling problem of Chapter 7 the two players play the following game. At each (discrete) time t, *Player*(1) selects a set s_t of n elements from $\{1, \ldots, n\}$. Then *Player*(2) selects an element f_t from s_t. We assume that *Player*(2) learns the choices of *Player*(1) (and remembers everything). Its set of views is therefore the set of finite sequences $s_1, f_1, s_2, f_2, \ldots$ By contrast, *Player*(1) learns none of the moves of *Player*(2) and its set of views is the set of sequences $s_1, s_2, \ldots$

The intuition behind the notion of view should be clear: a player with a restricted knowledge of the state of the system can sometime hold the same view x of two different states. Being unable to distinguish between these two states the player therefore uses the same probability space $(\Omega_x, \mathcal{G}_x, P_x)$ to take its next move.

In the sequel we distinguish between the views of the two players: we let X denote the set of views of *Player*(1) and Y the set of views of *Player*(2).

The previous examples illustrate a general principle about the update mechanism of the views held by the two players: when taking a move a player changes the

[9] As shown in the quote presented in page 18, in [49] Rabin actually defines a schedule to be a function on the set of finite runs. Equivalently, the assumption of [49] is that *Player*(2) knows the past run of every execution. Nevertheless, a careful analysis of the algorithm shows that this is equivalent to having *Player*(2) knowing the values of the program counters of the processes.

state of the system. Both players in general only learn partially the nature of the change and update their views as prescribed by the specification of the problem. Our model will introduce two functions f and g formalizing how the move of either player modifies the state of the system and the views held by both players. Let us emphasize that f and g formalize the acquisition of all information by either player. This applies in particular to the situation considered by Graham and Yao in [25] where *Player*(2) learns the strategy π selected by *Player*(1).

The previous discussion allows us to present in the next section a general model for randomized computing.

2.3 A general Model for Randomized Computing

2.3.1 The model

We argued in Chapter 1 and in Section 2.2 that a formal model for randomized computing should model simultaneously the notion of algorithm and of adversary and should allow for the consideration of several algorithms; that the notion of adversary should be modeled independently of the specific algorithm chosen; that the adversary should be randomized (i.e., allowed to use randomization).

We argued also that the situation encountered while modeling algorithms and adversaries was best described in terms of game theory: *Player*(1) is the entity selecting an (admissible) algorithm. *Player*(2) is the entity selecting an (admissible) adversary. An algorithm is a strategy of *Player*(1), whereas an adversary is a strategy of *Player*(2). These two players take steps in turn, following their respective strategies.

Using this language of game theory, we argued that a precise mechanism should be introduced, describing how the state of the system evolves and how the views of the two players are affected when one of the two players takes a move.

This discussion leads to the following definition.

Definition 2.3.1 *An algorithm/adversary structure for randomized computing is a tuple*

$$(S,\ X,\ Y,\ y_{init},\ f,\ g,\ \Pi,\ A)$$

having the following properties:

- *S is a set called the set of states*

- *X is a set called the set of views of Player*(1)*. By assumption $\perp$ is an element not in X.*
- *Y is a set called the set of views of Player*(2)
- *y_{init} is the* initial view *of Player*(2)*. By assumption y_{init} is not in Y.*
- *$\Pi = \{\pi_i\}_{i\in I}$ is a family of elements called algorithms. Each algorithm π_i is itself a family of probability spaces, one for each view x in X: $\pi_i = (\Omega_{x,i}, \mathcal{G}_{x,i}, P_{x,i})_{x\in X}$.*
- *$A = \{\mathcal{A}_j\}_{j\in J}$ is a family of elements called adversaries. Each adversary $\mathcal{A}_j$ is itself a family of probability spaces, one for each view y in $Y \cup \{y_{init}\}$: $\mathcal{A}_j = (\Omega_{y,j}, \mathcal{G}_{y,j}, P_{y,j})_{y\in Y\cup\{y_{init}\}}$. By assumption, $\perp$ is an element of $\Omega_{y,j}$ for all $y \in Y$ and $j \in J$.*
- *$f : S \times X \times Y \times (\bigcup_{x\in X, i\in I} \Omega_{x,i}) \to S \times X \times Y$ is the update function associated to Player*(1)*.*
- *$g : S \times X \times (Y \cup \{y_{init}\}) \times (\bigcup_{y\in Y, j\in J} \Omega_{y,j}) \to S \times X \times Y$ is the update function associated to Player*(2)*. The function g is such that, for every s and s' in S, x and x' in X, and a in $\bigcup_{x\in X, i\in I} \Omega_{x,i}$, we have $g(s, x, y_{init}, a) = g(s', x', y_{init}, a)$. For every s, x and y, $g(s, x, y, \perp) = (\perp, \perp, \perp)$.*

Abusing language, we will sometimes find it convenient to refer to a (S, X, Y, y_{init}, f, g, Π, A) structure as simply a Π/A-structure. This is very similar to the abuse of language committed by probabilists when speaking of a probability P without mentioning the underlying σ-field.

We now provide further justifications and reiterate some comments to the previous definition.

We will describe in Section 2.4 how an algorithm/adversary structure for randomized computing defines a set of executions. In doing so we will assume without loss of generality that *Player*(2) takes the first move and that its initial view is an element y_{init} not in Y: we can assume this without loss of generality because we can always add some dummy moves at the beginning of the game if necessary. The condition $g(s, x, y_{init}, a) = g(s', x', y_{init}, a)$ for every s and s' in S, x and x' in X, and a in $\bigcup_{x\in X, i\in I} \Omega_{x,i}$, ensures that the first move of *Player*(2) is independent of the values the state s and the view x might originally hold. For convenience we will let $g(y_{init}, a)$ the value common to all $g(s, x, y_{init}, a)$.

Let us emphasize that, eventhough seemingly restrictive, our choice of initial conditions is very general. In particular our model allows to express that "a randomized

algorithm must behave correctly for a family *Init* of different initial states". (This is in particular the case of Lehmann-Rabin's algorithm which we analyze in Chapter 4.) Indeed we model such a situation by enforcing that the first move of *Player*(2) consists in choosing one of the states s in *Init*. The subsequent moves then correspond to the "normal" exchanges between algorithm and adversary for the initial state s selected by *Player*(2). Hence, in such a model, the fact an algorithm "performs well" against all adversaries encompasses that it does so for all initial inputs. This example illustrates the power of the game theory setting that we adopt.

As discussed in Section 2.2, the purpose we seek in a model for randomized computing is to *analyze* randomized algorithms and not to *describe* them. As a consequence, eventhough a move of a player could be implemented in a variety of ways and involve several sequential instructions, we are only interested in the probability distribution of its output. This explains why we can assimilate a move to a probability space $(\Omega, \mathcal{G}, P)$ and assume that, in order to take this move, the player draws a random element of Ω with probability P.

A randomized algorithm is a strategy of *Player*(1), i.e., a description of how *Player*(1) is to take all its moves. Each move being described by a probability space $(\Omega, \mathcal{G}, P)$, a strategy π_i can be modeled as a family $(\Omega_{x,i}, \mathcal{G}_{x,i}, P_{x,i})_{x \in X}$ of probability spaces. X is the set of views that *Player*(1) can have of the system: *Player*(1) can act differently in two moves only if its views are different. This explains why a strategy π_i is associated to a different probability space for each different view x. This justifies the definition of an algorithm given in Definition 2.3.1.

As discussed in Section 2.2, an adversary should similarly be randomized and, in general, similarly allowed to have only a partial view of the state of the system. This justifies the definition of an adversary $\mathcal{A}_j$ as a family of probability spaces $(\Omega_{y,j}, \mathcal{G}_{y,j}, P_{y,j})_{y \in Y}$. The randomization of the adversary will be used crucially in Chapter 7. The restricted view of the adversary is a crucial feature of Rabin's algorithm for mutual exclusion studied in Chapter 3.

As mentioned in Page 32, the symbol $\perp$ is meant to signify that *Player*(2) delays forever taking its next move and that the execution stops. We check that our formalism is accurate. Assume that *Player*(2) selects $\perp$. By assumption $g(s, x, y, \perp) = (\perp, \perp, \perp)$ so that the view of *Player*(1) is overwritten with $\perp$. As $\perp$ is not in X, *Player*(1) does not take any move and the execution stops.

For emphasis, we sometimes call A the set of *admissible* adversaries. Similarly Π is the set of *admissible* algorithms. In the case where we analyze a single algorithm π_0 we have $\Pi = \{\pi_0\}$. This is the case of Chapter 3 and Chapter 4 of this work. On the other hand, in Chapter 7, we will prove the optimality of an algorithm within

an infinite class Π of algorithms.

The function f (resp. g) is the update function expressing how the state and the views of the two players evolve when *Player*(1) (resp. *Player*(2)) takes a move: if the state is s, if the views of the two players are x and y, and if a move a in $\Omega_{x,i}$ is selected by an algorithm π_i, then the new state is s' and the views of the two players are changed into x' and y', where s', x' and y' are defined by $f(s,x,y,a) = (s',x',y')$. Similarly, if the state is s', if the views of the two players are x' and y', and if a move a' in $\Omega_{y,j}$ is selected by an adversary $\mathcal{A}_j$, then the new state is s and the views of the two players are changed into x and y, where s, x and y are defined by $g(s',x',y',a') = (s,x,y)$.

Note that we imposed the function f to be defined on the cartesian product $S \times X \times Y \times (\bigcup_{x\in X, i\in I} \Omega_{x,i})$ only for simplicity of the exposition. We could for instance reduce its domain of definition to the subset $\{(s,x,y,a);\, s \in S,\, x \in X,\, y \in Y,\, a \in \cup_{i\in I}\Omega_{x,i}\}$. The domain of definition of g could similarly be reduced. As we will see in the next section, these variations do not affect the probabilistic structure on the set of executions derived from the model. Also, we could easily generalize our model to the case where, for every (s,x,y), a move a would lead to a randomized new configuration (s',x',y'). This situation would correspond to an environment with randomized dynamics. For simplicity we consider here only situations where the environment has deterministic dynamics.

The model of Definition 2.3.1 expends on the idea presented in Page 33 and defines algorithms and adversaries by providing their *local dynamics*. Indeed each probability space $(\Omega_{x,i}, \mathcal{G}_{x,i}, P_{x,i})$ describes how the algorithm π_i selects its next move when its view is x. And the function f describes what the state and the views evolve into after each such move. The adversary is defined through symmetric structures.

2.3.2 Special cases

We now mention two special cases which will play an important role in the sequel.

The analysis of a single algorithm π_0 corresponds to the case where Π is equal to the singleton $\{\pi_0\}$: *Player*(1) has only one strategy.

A second important case is when the strategies of *Player*(2) are all deterministic. This corresponds to the situation where every admissible adversary $\mathcal{A}_j = (\Omega_{y,j}, \mathcal{G}_{y,j}, P_{y,j})$ is such that all the measures $P_{y,j}$ are Dirac measures. In the case where the σ-fields $\mathcal{G}_{y,j}$ are discrete this means that for all j and y, there exists some point $\omega_{y,j}$ in $\Omega_{y,j}$ such that $P_{y,j}[\omega_{y,j}] = 1$.

As mentioned in Chapter 1, randomization does not make the adversary more powerful and we can without loss of generality assume that all admissible adversaries are deterministic when a single algorithm is considered. (Or more generally, when Π is finite.) This fact is largely acknowledged and explains that only deterministic adversaries are considered in the literature interested in the analysis of a given algorithm. (Hart et al. [29], page 359 and Aspnes-Waarts [4], section 5 mention explicitly that fact. See also, section 4 of [58] devoted to probabilistic concurrent programs, where Vardi models schedulers as deterministic entities.)

We are now ready to address the question raised in Section 2.2 and characterize the global probability space(s) whose sample space is the set executions and whose probability distribution is induced by the local dynamics of the two players.

2.4 The Probability Schema associated to a Π/A-Structure

Consider an algorithm/adversary structure (in short, a Π/A-structure) (S, X, Y, y_{init}, f, g, Π, A) as presented in Definition 2.3.1.

The purpose of this section is to define for each algorithm π in Π and each adversary $\mathcal{A}$ in A a probability space $(\Omega_{\pi,\mathcal{A}}, \mathcal{G}_{\pi,\mathcal{A}}, P_{\pi,\mathcal{A}})$ whose sample space $\Omega_{\pi,\mathcal{A}}$ is the set of executions generated by π and $\mathcal{A}$ and whose probability measure $P_{\pi,\mathcal{A}}$ is "induced" by the local dynamics of π and $\mathcal{A}$.

We need first to define the sample space $\Omega_{\pi,\mathcal{A}}$ and the σ-field $\mathcal{G}_{\pi,\mathcal{A}}$. The precise sample space $\Omega_{\pi,\mathcal{A}}$ we have in mind is the set of "maximal" executions, a maximal execution being either an infinite execution or a finite execution such that the last move of $Player(2)$ is $\perp$.

The σ-field $\mathcal{G}_{\pi,\mathcal{A}}$ we have in mind is the smallest one allowing to measure (in a probabilistic sense) all the sets of executions "defined by finitely many conditions".

In order to carry out our construction we need to make the following assumption.

Assumption: *The probability spaces* $(\Omega_{x,i}, \mathcal{G}_{x,i}, P_{x,i})$ *and* $(\Omega_{y,j}, \mathcal{G}_{y,j}, P_{y,j})$ *defining the algorithms in* Π *and the adversaries in* A *have all countable sample spaces. The associated* σ*-fields* $\mathcal{G}_{x,i}$ *and* $\mathcal{G}_{y,j}$ *are the discrete* σ*-fields* $\mathcal{P}(\Omega_{x,i})$ *and* $\mathcal{P}(\Omega_{y,j})$.

(Let us mention that the requirement that all the σ-fields are discrete is not restrictive: if this were not the case we would adopt an equivalent probabilistic model by

changing all the sample spaces and taking only one point in every atom.[10] Also, we could relax the hypothesis of countability of the sample spaces by requiring instead that all the probability measures $P_{x,i}$ and $P_{y,j}$ have countable support.)

Let $\pi_i = (\Omega_{x,i}, \mathcal{G}_{x,i}, P_{x,i})_{x\in X}$ and $\mathcal{A}_j = (\Omega_{y,j}, \mathcal{G}_{y,j}, P_{y,j})_{y\in Y}$ be two elements in Π and A respectively. A $(\pi_i, \mathcal{A}_j)$-execution ω is a (finite or infinite) sequence of alternating actions a and triples (s, x, y) of states and views ending, if the execution is finite, with the state-view triplet $(\perp, \perp, \perp)$,

$$\omega = \underbrace{a_1\,(s_1, x_1, y_1)}_{Player(2)}\ \underbrace{a_2\,(s_2, x_2, y_2)}_{Player(1)}\ \underbrace{a_3\,(s_3, x_3, y_3)}_{Player(2)} \ldots$$

and having the following properties:

1. $a_1 \in \Omega_{y_{init},j}$ and $(s_1, x_1, y_1) = g(y_{init}, a_1)$ [11]
2. for every even $k, k > 1$, $a_{k+1} \in \Omega_{y_k,j}$ and $(s_{k+1}, x_{k+1}, y_{k+1}) = g(s_k, x_k, y_k, a_{k+1})$
3. for every odd $k, k > 1$, $a_{k+1} \in \Omega_{x_k,i}$ and $(s_{k+1}, x_{k+1}, y_{k+1}) = f(s_k, x_k, y_k, a_{k+1})$.

A $(\pi_i, \mathcal{A}_j)$-execution-fragment is a finite execution-prefix terminating with a state-view triplet. For every $(\pi_i, \mathcal{A}_j)$-execution-fragment ω, we define the length $|\omega|$ of ω to be the number of actions a present in ω. For instance the length of $a_1\,(s_1, x_1, y_1)\,a_2\,(s_2, x_2, y_2)$ is 2. We define:

$$\Omega_{\pi_i,\mathcal{A}_j} = \{\omega\,;\,\omega \text{ is a } (\pi_i, \mathcal{A}_j)\text{-execution}\}.$$

We now turn to the formal definition of $\mathcal{G}_{\pi_i,\mathcal{A}_j}$. We endow the sets S, X and Y with the discrete σ-fields $\mathcal{P}(S)$, $\mathcal{P}(X)$ and $\mathcal{P}(Y)$. We define on $\Omega_{\pi_i,\mathcal{A}_j}$ the functions $A_k, S_k, X_k, Y_k; k \geq 1$, by setting $A_k(\omega) = a_k$, $S_k(\omega) = s_k$, $X_k(\omega) = x_k$ and $Y_k(\omega) = y_k$ for every $\omega = a_1\,(s_1, x_1, y_1)\,a_2\,(s_2, x_2, y_2)\ldots$ of length at least j. We extend the definition and set $A_k(\omega) = S_k(\omega) = X_k(\omega) = Y_k(\omega) = \perp$ if $|\omega| < k$.

We define $\mathcal{G}_{\pi_i,\mathcal{A}_j}$ to be the smallest σ-field making all the functions $A_k, S_k, X_k, Y_k; k \geq 1$ measurable:

$$\mathcal{G}_{\pi_i,\mathcal{A}_j} = \sigma\left(A_k, S_k, X_k, Y_k;\, k \geq 1\right).$$

Note that, as for every k the triple S_k, X_k, Y_k is defined deterministically in terms of A_k, S_{k-1}, X_{k-1} and Y_{k-1}, we also have that

$$\mathcal{G}_{\pi_i,\mathcal{A}_j} = \sigma\left(A_k; k \geq 1\right).$$

[10] See the Definition 8.1.1, page 198, for a definition of an atom.

[11] See the discussion on page 37 for a discussion of the special case $y = y_{init}$.

For the same reason we could have more simply defined an execution ω to be the sequence

$$\omega = a_1\, a_2\, a_3\, \ldots$$

in place of the sequence

$$\omega = a_1\,(s_1, x_1, y_1) a_2\,(s_2, x_2, y_2) a_3\,(s_3, x_3, y_3)\, \ldots$$

We will adopt this simplified definition in Chapter 7. Our slightly redundant definition has the advantage of making explicit states and views which are important parameters.

Recall that, by assumption, all the σ-fields $\mathcal{G}_{x,i}$ and $\mathcal{G}_{y,j}$ are the discrete σ-fields $\mathcal{P}(\Omega_{x,i})$ and $\mathcal{P}(\Omega_{y,j})$. This implies that for every a in $\Omega_{x,i}$ or $\Omega_{y,j}$ the singleton $\{a\}$ is in $\mathcal{G}_{x,i}$ or $\mathcal{G}_{y,j}$, respectively. Hence, for every sequence $(a_1, \ldots, a_k)$, the set $\{A_1 = a_1, \ldots, A_k = a_k\}$ is measurable in $\mathcal{G}_{\pi_i, \mathcal{A}_j}$. This allows us to present the following equivalent definition of $\mathcal{G}_{\pi_i, \mathcal{A}_j}$. For every $(\pi_i, \mathcal{A}_j)$-execution-fragment α we define the *rectangle* R_α to be the set of $(\pi_i, \mathcal{A}_j)$-executions having α as their initial prefix. Then

$$\mathcal{G}_{\pi_i, \mathcal{A}_j} = \sigma\big(R_\alpha;\, \alpha \text{ is a } (\pi_i, \mathcal{A}_j)\text{-execution-fragment}\big)\,.$$

This formalizes the claim formulated at the beginning of this section that $\mathcal{G}_{\pi_i, \mathcal{A}_j}$ is the σ-field allowing to measure probabilistically all the sets of executions defined "by finitely many conditions".

We now turn to the definition of the global probability measure $P_{\pi_i, \mathcal{A}_j}$. We want this measure, *if* it exists, to be compatible with the local dynamics defining π_i and $\mathcal{A}_j$. This means that, if $\alpha = a_1\,(s_1, x_1, y_1)\, a_2\,(s_2, x_2, y_2)\, \ldots\, a_k\,(s_k, x_k, y_k)$ is a $(\pi_i, \mathcal{A}_j)$-execution-fragment, then

$$P_{\pi_i, \mathcal{A}_j}[R_\alpha] = P_{y_{init}, j}[a_1]\, P_{x_1, i}[a_2]\, P_{y_2, j}[a_3]\, \ldots P_{x_{k-1}, i}[a_k]\,,$$

where, for the sake of exposition, we assumed that k was even. We now define a filtration $(\mathcal{G}_k)_{k \geq 1}$ of $(\Omega_{\pi_i, \mathcal{A}_j}, \mathcal{G}_{\pi_i, \mathcal{A}_j})$, i.e., an increasing sequence of sub-σ-fields of $\mathcal{G}_{\pi_i, \mathcal{A}_j}$. For every $k, k \geq 1$, we set

$$\mathcal{G}_k = \sigma\big(R_\alpha;\, \alpha \text{ is a } (\pi_i, \mathcal{A}_j)\text{-execution-fragment}, |\alpha| \leq k\big)\,.$$

The σ-field $\mathcal{G}_k$ should be more appropriately called $\mathcal{G}_{i,j,k}$. We drop the indices i and j for simplicity. Then, for every $k \geq 1$, setting

$$\mathbf{P}_k[R_\alpha] = P_{y_{init}, j}[a_1]\, P_{x_1, i}[a_2]\, P_{y_2, j}[a_3]\, \ldots P_{x_{k'-1}, i}[a_{k'}]$$

for every $(\pi_i, \mathcal{A}_j)$-execution-fragment α of even length $k' \leq k$, and similarly setting

$$\mathbf{P}_k[R_\alpha] = P_{y_{init},j}[a_1]\, P_{x_1,i}[a_2]\, P_{y_2,j}[a_3]\, \ldots P_{y_{k'-1},j}[a_{k'}]$$

for every $(\pi_i, \mathcal{A}_j)$-execution-fragment α of odd length $k' \leq k$, defines a probability measure $\mathbf{P}_k$ on $\mathcal{G}_k$. The probability measures $(\mathbf{P}_k)_{k\geq 1}$ are compatible in the sense that, if $k \leq l$ are two integers then $\mathbf{P}_k[R_\alpha] = \mathbf{P}_l[R_\alpha]$ for every $(\pi_i, \mathcal{A}_j)$-execution-fragment α of length at most k. Equivalently, the measures $\mathbf{P}_k$ and $\mathbf{P}_l$ coincide on $\mathcal{G}_k$. Therefore, by Kolmogorov's extension theorem, (see for instance [56], page 161–163), there exists a (unique) probability measure $\mathbf{P}$ defined on the σ-field $\sigma(\cup_k \mathcal{G}_k)$. As $\sigma(\cup_k \mathcal{G}_k) = \mathcal{G}_{\pi_i,\mathcal{A}_j}$ we have therefore established the existence of a probability measure $\mathbf{P}$ defined on $(\Omega_{\pi_i,\mathcal{A}_j}, \mathcal{G}_{\pi_i,\mathcal{A}_j})$ and compatible with the local dynamics defining π_i and $\mathcal{A}_j$. This measure $\mathbf{P}$ is the measure $P_{\pi_i,\mathcal{A}_j}$ we were after.

This finishes to characterize the global probability space $(\Omega_{\pi_i,\mathcal{A}_j}, \mathcal{G}_{\pi_i,\mathcal{A}_j}, P_{\pi_i,\mathcal{A}_j})$ compatible with the local dynamics attached to an algorithm π_i in Π and an adversary $\mathcal{A}_j$ in A as defined in a Π/A structure (S, X, Y, y_{init}, f, g, Π, A). The construction required that the probability spaces $(\Omega_{x,i}, \mathcal{G}_{x,i}, P_{x,i})$ and $(\Omega_{y,j}, \mathcal{G}_{y,j}, P_{y,j})$ defining the algorithm π_i and the adversary $\mathcal{A}_j$ have all countable sample spaces.

The analysis of a Π/A structure requires the simultaneous consideration of all the probability spaces $(\Omega_{\pi_i,\mathcal{A}_j}, \mathcal{G}_{\pi_i,\mathcal{A}_j}, P_{\pi_i,\mathcal{A}_j})$. Correspondingly, the notion of event has to be modified to take into account the various choices of strategy made by *Player*(1) and *Player*(2). The next definition summarizes these facts.

Definition 2.4.1 *The family* $(\Omega_{\pi,\mathcal{A}}, \mathcal{G}_{\pi,\mathcal{A}}, P_{\pi,\mathcal{A}})_{(\pi,\mathcal{A})\in\Pi\times A}$ *is called the* probability schema *attached to the algorithm/adversary structure* (S, X, Y, y_{init}, f, g, Π, A). *An* event schema B *is a family* $B = (B_{\pi,\mathcal{A}})_{(\pi,\mathcal{A})\in\Pi\times A}$, *where* $B_{\pi,\mathcal{A}} \in \mathcal{G}_{\pi,\mathcal{A}}$ *for every* $(\pi, \mathcal{A}) \in \Pi \times A$. *A* variable-schema X *is a family* $X = (X_{\pi,\mathcal{A}})_{(\pi,\mathcal{A})\in\Pi\times A}$, *where, for every* $(\pi, \mathcal{A}) \in \Pi \times A$, $X_{\pi,\mathcal{A}}$ *is a random variable measurable with respect to* $\mathcal{G}_{\pi,\mathcal{A}}$.

Traditionally Kolmogorov's theorem is stated for $\mathbb{R}^\infty$ endowed with its Borel σ-field $\mathcal{B}(\mathbb{R}^\infty)$. We therefore explain and justify here our use of Kolmogorov's theorem in the previous construction. This discussion is technical and not fundamental for the rest of the work.

Recall that, by assumption, all the sample spaces $\Omega_{x,i}$ and $\Omega_{y,j}$ are countable and endowed with their discrete σ-fields $\mathcal{P}(\Omega_{x,i})$ and $\mathcal{P}(\Omega_{y,j})$. By relabelling we can therefore take all these spaces to be equal to $\mathbb{N}$ endowed with the discrete σ-field

$\mathcal{P}(\mathbb{N})$. Hence we can take $\Omega_{\pi_i,\mathcal{A}_j} = \mathbb{N} \times \mathbb{N} \times \ldots$ endowed with the product σ-field $\mathcal{P}(\mathbb{N}) \otimes \mathcal{P}(\mathbb{N}) \otimes \ldots$ For each k, the σ-field $\mathcal{G}_k$ then corresponds to the set $\{A \times \mathbb{N} \times \mathbb{N} \times \ldots ; A \in \mathcal{P}(\mathbb{N}^k)\}$. For each k the space $\mathbb{N}^k$ is a complete (a Cauchy-sequence in $\mathbb{N}^k$ is constant after a certain rank and hence obviously converging) separable metric space (for the topology induced from $\mathbb{R}^k$). As every point in $\mathbb{N}^k$ is open, the σ-field $\mathcal{P}(\mathbb{N}^k)$ is trivially generated by the open sets. Therefore the extension of Kolmogorov's theorem given in [56], page 163, applies.

Chapter 3

Rabin's Algorithm for Mutual Exclusion

3.1 Introduction

In this chapter we study the correctness of Rabin's randomized distributed algorithm [49] implementing mutual exclusion for n processes using a read-modify-write primitive on a shared variable with $O(\log n)$ values. As we are concerned with a single algorithm, the remarks made in Section 2.3.2 of Chapter 2 allow us to consider only deterministic adversaries throughout the analysis. Rabin's algorithm differs markedly with most other work in randomized computing in the three following ways.

1. The correctness statement is expressed in terms of a property holding with "high" probability for all adversaries. Formally, such a statement is of the form
$$\inf_{\mathcal{A} \in A'} P_{\mathcal{A}}[W_{\mathcal{A}} \mid I_{\mathcal{A}}] \geq \alpha \,,$$
where W and I are event schemas[1] and A' is a subset of the set of admissible adversaries A. Nevertheless, in contrast with much previous works, "high" does not mean here "with probability one", i.e., α is not equal to one.

2. This correctness property is relative to the short term behavior of the algorithm and depends critically on the specific probability distribution of the random inputs.

[1] See Definition 2.4.1, page 43.

3. The adversary is assumed to have only a partial knowledge of the past execution.

We comment here on properties 1–3. Properties 1 and 2 are related. Property 3 is of another nature.

As discussed in Chapter 1, most correctness statements for randomized algorithms are either expressed in terms of a high expected performance or in terms of a property holding with probability one. For instance the algorithms presented in [3] and [4] are shown to have a low worst case expected running time.[2] On the other hand, the original property claimed in [37] for Lehmann-Rabin's algorithm is that "progress occurs with probability one".

A reason for considering probability-one statements is that they arise naturally from considerations of ergodic theory and zero-one laws. In particular, a feature of these statements is that the events whose probability is claimed to be one belong to the tail σ-field of the executions, i.e., depend (only) on the asymptotic properties of the executions. Also, these properties usually do not depend on the precise numeric probability values associated to each random input X_n. Instead, they typically depend only on the asymptotic properties of the sequence $(X_n)_n$.[3]

Such a setting has been exploited by authors interested in the logic and semantic of randomized computing. (See for instance [19, 28, 29, 47, 58].) The methods presented in these papers are directed at randomized algorithms whose correctness reflects the asymptotic properties of infinite executions, depends only marginally on the numeric specifications of the random inputs and, is expressed as a probability-one statement.

These methods therefore do not apply to algorithms as Rabin's algorithm whose correctness, as we will see, is relative to the short term behavior of the algorithm, depends critically on the specific probability distribution(s) of the random inputs and is expressed by a non trivial high-probability statement.

[2] The term "worst case" refers to the adversary. The precise measure used in [4] is the worst case expected running time *per processor*.

[3] The Borel-Cantelli Lemma provides a good example of a classical result of probability theory which depends only on the tail of the sequence of events considered. This property states that, if $(A_n)_{n\in\mathbb{N}}$ is a sequence of independent events, then infinitely many events A_n occur with probability one if and only if the sum $\sum_n P(A_n)$ is infinite. Assume for instance that $A_n = \{X_n = H\}$ where $(X_n)_{n\in\mathbb{N}}$ is a sequence of flips made with independent coins. (The coins are possibly all different.) Then Head appears infinitely often if and only if $\sum_n P[X_n = H] = \infty$. This depends only on the asymptotic property of the coins and depends on the bias of the coins only through the sum $\sum_n P[X_n = H]$.

These difficulties significantly complicate the analysis of any randomized algorithm. But one of the most significant challenges encountered in the proof of correctness of Rabin's algorithm [49] is to account formally for the limited knowledge granted to the adversary. As mentioned in Chapter 1, to our knowledge, our model of Chapter 2 is the first to present a general and formal framework allowing for adversaries with limited knowledge. In the absence of such a framework, the analyses of non trivial algorithms as the one presented in [49] have been fraught with mistakes in two rather incomparable domains.

Mistakes can happen of course in the proof of the property claimed. Such mistakes are not uncommon, as it is very hard to formally disentangle in a proof the combined effects of randomness and of non-determinism. In particular, recall from Chapter 2 that the formal analysis of a randomized algorithm requires to work within a class of different probability spaces $(\Omega_{\mathcal{A}}, \mathcal{G}_{\mathcal{A}}, P_{\mathcal{A}})$, one for each admissible adversary $\mathcal{A}$. A proof is therefore not truly formal unless it explicitly records the presence of the adversary $\mathcal{A}$ in the probabilistic expressions involved.

But the complications in the analysis of a randomized algorithm sometimes begin with the formalization of the intended correctness property, i.e., even before the proof itself begins. Indeed, as we will see in Section 3.2, even a simple statement including a precondition can lead to very delicate problems of formalization. Using an image one might describe this problem as a Solomon dilemma trying to ascertain "which of the two players should bear the responsibility of the precondition".

The example of Rabin's randomized algorithm for mutual exclusion illustrates perfectly the two types of difficulties and the dangers encountered with arguments not fully formalized. In [49], Rabin claimed that the algorithm satisfies the following correctness property: for every adversary, any process competing for entrance to the critical section succeeds with probability $\Omega(1/m)$, where m is the number of competing processes. In [29], Sharir et al. gave another discussion of the algorithm as an illustration of their formal Markov chains model and argued about its correctness.

However, both papers did not write formally the correctness property claimed and did not make explicit in their arguments the influence of the adversary on the probability distribution on executions.

We show in this chapter that the correctness of the algorithm is tainted for the two different reasons described above. We first show that the informal correctness statement claimed by Rabin admits no natural "adequate" formalization.[4] We then show that the influence of the adversary is much stronger than previously thought, and in fact, the high probability correctness result claimed in [49] does not hold.

[4] The notion of "adequate" formalization is discussed in Section 3.2.

3.2 Formalizing a High-Probability Correctness Statement

As mentioned above, the complications in the analysis of a randomized algorithm sometimes begin with the formalization of the intended correctness property, i.e., even before the proof itself begins. Consider for instance a correctness property described informally in the following general format: "for all adversaries, if property B holds then property C holds with probability at least $1/2$". (The probability $1/2$ is chosen only for the sake of illustration.) How can we formalize such a statement? In the sequel we will refer to C as the target property and to B as the precondition.

Again, as discussed in Chapter 2, we know that the property B is actually formalized to be a family of events $(B_{\mathcal{A}})_{\mathcal{A}\in A}$ (each $B_{\mathcal{A}}$ is an element of the σ-field $\mathcal{G}_{\mathcal{A}}$), and that, similarly, C is a family of events $(C_{\mathcal{A}})_{\mathcal{A}\in A}$. Also, the probability referred in the informal correctness statement depends on $\mathcal{A}$ and corresponds to different probability distributions $P_{\mathcal{A}}$. In spite of this comment we will conform to the tradition, and write for simplicity B and C instead of $B_{\mathcal{A}}$ and $C_{\mathcal{A}}$ and emphasize only when necessary the dependence in $\mathcal{A}$. (The dependence in $\mathcal{A}$ will nevertheless be everywhere implicit.) On the other hand, we will find it useful to emphasize the dependence of the probability $P_{\mathcal{A}}$ on the adversary $\mathcal{A}$. This being clarified, how do we formalize the "*if* B then C" clause of the statement?

Following well-anchored reflexes in probability theory we are naturally led to translate this statement into a conditional probability and say that, for every $\mathcal{A}$, we compute the probability of the event $C_{\mathcal{A}}$ conditioned on $B_{\mathcal{A}}$. Indeed, conditioning on an event B exactly formalizes that we restrict the probabilistic analysis to within the set B. (The elementary definition of conditioning in terms of Bayes' rule expresses that we restrict our attention to within B and consider the trace $C \cap B$ of C in B. The denominator $P[B]$ is there to normalize the restricted measure $C \to P[C \cap B]$ into a probability measure.[5]) We can then formalize the previous informal correctness statement into

$$\boxed{\inf_{\mathcal{A}\in A;\, P_{\mathcal{A}}[B]>0} P_{\mathcal{A}}[C \mid B] \geq 1/2} \quad .$$

[5] A more general definition of conditioning is as follows. In that case the "notion of restricting the analysis" is formalized as an orthogonal projection in an adequate setting.

Let $\mathcal{G}, \mathcal{G}'$; $\mathcal{G}' \subseteq \mathcal{G}$ be two σ-fields, and let $(\Omega, \mathcal{G}, P)$ be a measured space. For every function f in $L^2(dP)$, we define the conditional expectation $E[f \mid \mathcal{G}']$ to be the orthogonal projection (in the L^2-sense) of the $\mathcal{G}$-measurable function f onto the subset of $L^2(dP)$ composed of $\mathcal{G}'$-measurable functions. We recover the elementary definition in the case where f is the indicator function of a set C and $\mathcal{G}' = \{\emptyset, B, \overline{B}, \Omega\}$: in that case, the orthogonal projection of f onto $\mathcal{G}'$ is coincides with the intersection $C \cap B$.

The restriction $P_{\mathcal{A}}[B] > 0$ expresses that we consider only adversaries $\mathcal{A}$ for which the precondition B is of probabilistic relevance.

An example.
We now show that, in some (frequent) cases, the dependence on $\mathcal{A}$ of the correctness statement can have far reaching consequences and negate intuitive properties. Consider for instance the following example proposed in a footnote of [11].

> Imagine performing a random walk on a line, but where the adversary can stop you at any time $t \leq 100$. One might like to say that: "if the adversary allows you to make 100 steps, then with probability at least a half you will have made 40 steps to the right".

We now formalize this statement. For all adversaries, the sample space can be taken to be $\Omega = \{H, T\}^{100}$. For all adversaries, we also consider the associated discrete σ-field 2^{Ω}. The probability distributions $P_{\mathcal{A}}$ are defined as follows. Let α be an execution-fragment (i.e., a sequence of draws). If $\mathcal{A}$ does not allow the occurrence of α (i.e., if there is a strict prefix α_1 of α after which $\mathcal{A}$ allocates no steps) then $P_{\mathcal{A}}[\alpha] = 0$. If $\mathcal{A}$ allows the occurrence of α but blocks the execution at α then $P_{\mathcal{A}}[\alpha] = 2^{-|\alpha|+1}$, where $|\alpha|$ denotes the length of α. If $\mathcal{A}$ allows the occurrence of α and does not block the execution at α then $P_{\mathcal{A}}[\alpha] = 2^{-|\alpha|}$. Hence the support of the measure $P_{\mathcal{A}}$ is exactly the set of *maximal* executions allowed (with non-zero probability) by $\mathcal{A}$. We argue that this construction corresponds to an equivalent form of the general construction given in Chapter 2. In Chapter 2 we (construct and) consider a different probability space $(\Omega_{\mathcal{A}}, \mathcal{G}_{\mathcal{A}}, P_{\mathcal{A}})$ for every adversary $\mathcal{A}$: $\Omega_{\mathcal{A}}$ is the set of maximal executions under $\mathcal{A}$. Here we consider instead a measurable space $(\Omega, \mathcal{G})$ common for all the adversaries. Note that the two measurable structures are related: $(\Omega_{\mathcal{A}}, \mathcal{G}_{\mathcal{A}}) \subseteq (\Omega, \mathcal{G})$ for every $\mathcal{A}$. (This means that $\Omega_{\mathcal{A}} \subseteq \Omega$ and that $\mathcal{G}_{\mathcal{A}} \subseteq \mathcal{G}$ for every $\mathcal{A}$.) The equivalence between the two models is ensured by the fact that, in both cases, for every $\mathcal{A}$ the support of $P_{\mathcal{A}}$ is equal to $\Omega_{\mathcal{A}}$.

We define B to be the event "the adversary allows you to make 100 steps" and let C to be "Head comes up at least 40 times in the experiment". Is it true that $P_{\mathcal{A}}[C \mid B] \geq 1/2$ for all adversaries $\mathcal{A}$ allowing 100 steps with non-zero probability ? The answer is no. For instance, let $\mathcal{A}$ be the adversary that stops the process as soon as a Head comes up. This adversary allows 100 steps with some non-zero probability, namely when all draws are Tail. On the other-hand, for this adversary, conditioned on B, the number of Heads is at most 1 with probability 1.

But this argument *only* shows that the correctness property proposed does not fit

the algorithm, not that the algorithm is "not correct"[6]: the algorithm must be evaluated by other means. For instance, as we now show, the algorithm satisfies

$$\boxed{\inf_{\mathcal{A}\in A;\, P_{\mathcal{A}}[B]=1} P_{\mathcal{A}}[C] \geq 1/2}$$

and also satisfies

$$\boxed{\inf_{\mathcal{A}\in A} P_{\mathcal{A}}[B \Rightarrow C] \geq 1/2} \quad .$$

In this last statement $B \Rightarrow C$ denotes the event $\overline{B} \cup C$. ($\overline{B}$ denotes the complement of B.) The first statement is easily proven, as, by definition, "$P_{\mathcal{A}}[B] = 1$" means that the adversary allocates 100 steps and that, correspondingly, 100 independent coin tosses are performed:

$$\begin{aligned} \inf_{\mathcal{A},\, P_{\mathcal{A}}[B]=1} P_{\mathcal{A}}[C] &= \text{Prob[100 independent flips result in at least 40 Heads]} \\ &\geq 1/2\,. \end{aligned}$$

The second statement is harder to prove and is actually a consequence of the equality $\inf_{\mathcal{A}\in A} P_{\mathcal{A}}[B \Rightarrow C] = \inf_{\mathcal{A}\in A;\, P_{\mathcal{A}}[B]=1} P_{\mathcal{A}}[C]$, which we now establish. Let us emphasize that, as we will later argue, this equality does not hold in general. The idea of the proof is that an adversary providing less then 100 steps in some executions yields a higher or equal probability $P_{\mathcal{A}}[B \Rightarrow C]$ then some other adversary always allocating (at least) 100 steps.

To argue this formally, consider an adversary $\mathcal{A}$ for which $P_{\mathcal{A}}[B \Rightarrow C]$ is minimized. Assume the existence of an execution-fragment α (i.e., a sequence of draws) having length k for some k less then 100, and such that 1) $P_{\mathcal{A}}[\alpha] > 0$ and 2), $\mathcal{A}$ does not allocate any steps after α. (We will call $Stop_{\mathcal{A}}$ the set of such execution fragments α.) Consider the adversary $\mathcal{A}'$ which is equal to $\mathcal{A}$ except that $\mathcal{A}'$ always allocates a total of 100 steps if the execution begins with α. Let D denote the event "the first k draws do not yield α". Note that, by definition of D and $\mathcal{A}$, $P_{\mathcal{A}}[\overline{D} \subseteq \overline{B}] = 1$, and hence:

$$P_{\mathcal{A}}[B \Rightarrow C] = P_{\mathcal{A}}[B \Rightarrow C, \overline{D}] + P_{\mathcal{A}}[B \Rightarrow C, D] = P_{\mathcal{A}}[\overline{D}] + P_{\mathcal{A}}[B \Rightarrow C, D].$$

We have:

$$P_{\mathcal{A}'}[B \Rightarrow C] \quad = \quad P_{\mathcal{A}'}[B \Rightarrow C \mid \overline{D}]\, P_{\mathcal{A}'}[\overline{D}] + P_{\mathcal{A}'}[B \Rightarrow C \mid D]\, P_{\mathcal{A}'}[D]$$

[6] The algorithm considered here is the trivial algorithm associated to the random walk and whose code is "flip a coin": whenever the adversary allocates a step that unique instruction is performed.

$$
\begin{aligned}
&= P_{\mathcal{A}'}[B \Rightarrow C \mid \overline{D}]\, P_{\mathcal{A}'}[\overline{D}] + P_{\mathcal{A}}[B \Rightarrow C \mid D]\, P_{\mathcal{A}}[D] && (3.1)\\
&= P_{\mathcal{A}'}[B \Rightarrow C \mid \overline{D}]\, P_{\mathcal{A}}[\overline{D}] + P_{\mathcal{A}}[B \Rightarrow C, D] && (3.2)\\
&\leq P_{\mathcal{A}}[\overline{D}] + P_{\mathcal{A}}[B \Rightarrow C, D]\\
&= P_{\mathcal{A}}[B \Rightarrow C]\,.
\end{aligned}
$$

Equations 3.1 and 3.2 come from the fact that $\mathcal{A}'$ behaves as $\mathcal{A}$ in D and during the execution fragment α: $\mathcal{A}'$ behaves differently only if and once α appears, i.e., in $\overline{D}$.

We have therefore constructed an adversary $\mathcal{A}'$ such that $|Stop_{\mathcal{A}'}| < |Stop_{\mathcal{A}}|$, i.e., which brings to termination "more" executions then $\mathcal{A}$ and which is such that $P_{\mathcal{A}'}[B \Rightarrow C] \leq P_{\mathcal{A}}[B \Rightarrow C]$. By iteration, we can therefore produce an adversary that we will still denote $\mathcal{A}'$ which allocates always 100 steps, i.e., such that $P_{\mathcal{A}'}[B] = 1$, and whose associated probability $P_{\mathcal{A}'}[B \Rightarrow C]$ is no bigger then $P_{\mathcal{A}}[B \Rightarrow C]$. This justifies the following first equality.

$$
\begin{aligned}
\inf_{\mathcal{A}} P_{\mathcal{A}}[B \Rightarrow C] &= \inf_{\mathcal{A},\, P_{\mathcal{A}}[B]=1} P_{\mathcal{A}}[B \Rightarrow C]\\
&= \inf_{\mathcal{A},\, P_{\mathcal{A}}[B]=1} P_{\mathcal{A}}[C]\\
&\geq 1/2\,.
\end{aligned}
$$

Remark that the proof of the equality $\inf_{\mathcal{A}} P_{\mathcal{A}}[B \Rightarrow C] = \inf_{\mathcal{A},\, P_{\mathcal{A}}[B]=1} P_{\mathcal{A}}[C]$ was only possible because of the special nature of the problem considered here. In particular we found it very useful that all the spaces $(\Omega_{\mathcal{A}}, \mathcal{G}_{\mathcal{A}})$ were finite and equal for all adversaries and that the events B and C were true events (and not only general event-schemas). Also, the proof uses that, for every adversary $\mathcal{A}$, it is possible to construct an adversary $\mathcal{A}'$ having the following two properties:

1. The adversary $\mathcal{A}'$ gives probability one to B: $P_{\mathcal{A}'}[B] = 1$.
2. The adversary $\mathcal{A}'$ "makes the same decisions as the adversary $\mathcal{A}$ along the executions ending in B under the control of $\mathcal{A}$". More formally, the probability measure $P_{\mathcal{A}'}$ coincides with the probability measure $P_{\mathcal{A}}$ on the set $\{\omega; \omega \in B \text{ and } P_{\mathcal{A}}[\omega] > 0\}$.[7]

The following modification on our example illustrates the importance of the first previous property. (An example illustrating the importance of the second property could similarly be constructed.) Assume that the game is changed so that, now, an execution can also stop with some probability ε at each step of the random

[7] This property was the key for the derivation of Equations 3.1 and 3.2.

walk. Then, obviously, no adversary can ensure termination with probability one i.e., for all $\mathcal{A}$, $P_{\mathcal{A}}[B] < 1$. Hence $\inf_{\mathcal{A}\in A;\, P_{\mathcal{A}}[B]=1} P_{\mathcal{A}}[C] = \infty$.[8] On the other hand $\inf_{\mathcal{A}\in A} P_{\mathcal{A}}[B \Rightarrow C]$ is obviously bounded (by 1!) and hence $\inf_{\mathcal{A}\in A} P_{\mathcal{A}}[B \Rightarrow C] \leq \inf_{\mathcal{A}\in A;\, P_{\mathcal{A}}[B]=1} P_{\mathcal{A}}[C]$. This inequality is general as the following proposition demonstrates. (We emphasize there for formality the dependence on $\mathcal{A}$.)

Proposition 3.2.1 *Let $B = (B_{\mathcal{A}})_{\mathcal{A}\in A}$ and $C = (C_{\mathcal{A}})_{\mathcal{A}\in A}$ be two event schemas. Assume that $\{\mathcal{A} \in A;\, P_{\mathcal{A}}[B] > 0\} \neq \emptyset$. Then*

$$\inf_{\mathcal{A}\in A;\, P_{\mathcal{A}}[B_{\mathcal{A}}]>0} P_{\mathcal{A}}[C_{\mathcal{A}} \mid B_{\mathcal{A}}] \leq \inf_{\mathcal{A}\in A} P_{\mathcal{A}}[B_{\mathcal{A}} \Rightarrow C_{\mathcal{A}}] \leq \inf_{\mathcal{A}\in A;\, P_{\mathcal{A}}[B_{\mathcal{A}}]=1} P_{\mathcal{A}}[C_{\mathcal{A}}]\,.^{8}$$

Proof. We abuse notations slightly by writing B instead of $B_{\mathcal{A}}$ and C instead of $C_{\mathcal{A}}$. Obviously, $A_1 \stackrel{\text{def}}{=} \{\mathcal{A} \in A;\, P_{\mathcal{A}}[B] = 1\}$ is a subset of A. Also, for every $\mathcal{A}$ in A_1, we have $P_{\mathcal{A}}[B \Rightarrow C] = P_{\mathcal{A}}[C]$. This establishes that

$$\inf_{\mathcal{A}\in A} P_{\mathcal{A}}[B \Rightarrow C] \leq \inf_{\mathcal{A}\in A;\, P_{\mathcal{A}}[B]=1} P_{\mathcal{A}}[C]\,.$$

Let A_2 be the set $\{\mathcal{A} \in A;\, P_{\mathcal{A}}[B] > 0\}$. If $\mathcal{A} \in A - A_2$ then, by definition, $P_{\mathcal{A}}[B] = 0$ and hence $P_{\mathcal{A}}[B \Rightarrow C] = 1$. As by assumption A_2 is not empty, this trivially implies that $\inf_{\mathcal{A}\in A_2} P_{\mathcal{A}}[C \mid B] \leq \inf_{\mathcal{A}\in A-A_2} P_{\mathcal{A}}[B \Rightarrow C]$.[8]

Consider now an adversary $\mathcal{A}$ in A_2. We have:

$$\begin{aligned}
P_{\mathcal{A}}[B \Rightarrow C] &= P_{\mathcal{A}}[\overline{B} \cup C] \\
&= P_{\mathcal{A}}[\overline{B} \cup (C \cap B)] \\
&= P_{\mathcal{A}}[\overline{B}] + P_{\mathcal{A}}[C \cap B] \\
&= P_{\mathcal{A}}[\overline{B}] + P_{\mathcal{A}}[C \mid B]\, P_{\mathcal{A}}[B] \\
&= (1 - P_{\mathcal{A}}[C \mid B])P_{\mathcal{A}}[\overline{B}] + P_{\mathcal{A}}[C \mid B]\, (P_{\mathcal{A}}[\overline{B}] + P_{\mathcal{A}}[B]) \\
&\geq P_{\mathcal{A}}[C \mid B]
\end{aligned}$$

This immediately implies that $\inf_{\mathcal{A}\in A_2} P_{\mathcal{A}}[C \mid B] \leq \inf_{\mathcal{A}\in A_2} P_{\mathcal{A}}[B \Rightarrow C]$. This inequality along with the one proved above establishes that

$$\inf_{\mathcal{A}\in A_2} P_{\mathcal{A}}[C \mid B] \leq \inf_{\mathcal{A}\in A} P_{\mathcal{A}}[B \Rightarrow C]\,.$$

□

[8] Recall that $\inf_{x\in X} f(x) = \infty$ if X is the empty set. This is justified by the fact that, by definition, $\inf_{x\in X} f(x)$ is the biggest lower bound of the set $\{f(x); x \in X\}$. Hence, if X is empty, all numbers are lower bounds and the infimum is therefore infinite.

Note also the interesting general probabilistic relation, which is easily derived using Bayes' rule: $P[B \Rightarrow C] = P[C \mid B]$ if and only if $P[B \cap C'] = P[B]\,P[C']$, where we let C' denote the event $B \Rightarrow C$. In particular, $P[B \Rightarrow C] = P[C \mid B]$ if $(B \Rightarrow C)$ is independent of B.[9]

Which correctness measure is adequate.
The previous discussion shows that "adequately" formalizing the correctness statement of a randomized algorithm is a not trivial problem. We proposed three possible ways to formalize a correctness statement presented in the format: "for all adversaries, if property B holds then property C holds with probability at least ε." It is natural to wonder which of the three is most "adequate" in practice, whether they have different domains of application and whether some other good or even better measures exist. The answer to these questions has significant implications as it determines the benchmark to which randomized algorithms are evaluated and against which their correctness is decided.

What do we mean by an "adequate measure"? The example discussed in page 49 provides a good illustration of the problem. We showed that, for this game and the choices B = "100 steps are allocated", C = "At least 40 Heads turn up" the measure $\inf_{\mathcal{A} \in A;\, P_{\mathcal{A}}[B]>0} P_{\mathcal{A}}[C \mid B]$ is equal to 0. The intuition behind this result is that the use of this measure provides *Player*(2) with the implicit knowledge of B. (See page 207 for a discussion on implicit knowledge.) Using this knowledge, *Player*(2) is then able to select a specific strategy $\mathcal{A}$ so as to annulate the probability of occurrence of C.

To justify why this measure is not adequate we have to return to our original intention. We can assume that we are provided with a Π/A structure $(S, X, Y, y_{init}, f, g, \Pi, A)$ as in Definition 2.3.1, page 36, formalizing how the game is played between *Player*(1) and *Player*(2). (As we are analyzing a single algorithm π_0, Π is by definition equal to the singleton $\{\pi_0\}$. Also, as mentioned at the beginning of the chapter, the set A of admissible adversaries contains only deterministic adversaries.) We would ideally like to play the role of a passive observer, able to analyze the game without interfering with it. In this perspective, a measure is deemed adequate if its use does not perturb the experiment, i.e., the game between *Player*(1) and *Player*(2).

It is clear from our discussion that the measure $\inf_{\mathcal{A} \in A;\, P_{\mathcal{A}}[B]>0} P_{\mathcal{A}}[C \mid B]$ greatly affects the game: we arrive at the scene, impose the unnatural condition B and let

[9] The relation $P[B \cap C'] = P[B]\,P[C']$ does not imply the independence of B and C', because this independence also involves similar relations with the complements of B and C'.

Player(2) use it to its fullest. By contrast, an adequate measure would be one that would derive probabilistic facts about the Π/A structure that would hold similarly if no measure was conducted. (This will constitute our definition of *adequacy*. As in physics, it is hard to formalize how a measure influences the ambient structure. The reason for the difficulty is by essence that we have access to the structure studied only by measuring it.) With this in mind we can now come back to the analysis of our three measures.

Probabilistic conditioning. First note that the adversary is restricted in a minimal way by the precondition B when the measure $\inf_{\mathcal{A}\in A;\, P_{\mathcal{A}}[B]>0} P_{\mathcal{A}}[C \mid B]$ is used: it must just not disallow B probabilistically. Furthermore, as is discussed in Chapter 8, page 207, *Player*(2) *learns implicitly* that the execution is restricted to within the set B. *Player*(2) can take selective actions (i.e., design some specific strategy $\mathcal{A}$) so as to take advantage of this knowledge.

Conditioning by adversary. By contrast, if the measure $\inf_{\mathcal{A}\in A;\, P_{\mathcal{A}}[B]=1} P_{\mathcal{A}}[C]$ is used, *Player*(2) is in some sense "fully responsible" to ensure that the precondition B happens. The only strategies of *Player*(2) (i.e., the only adversaries) that are retained are those that ensure with probability one that B happens.

These two measures correspond therefore to two extremes cases. In the setting imposed by the first measure, *Player*(2) can use without any restriction the information that the executions take place in B. In the second setting *Player*(2) is most restricted and must select strategies ensuring that the executions happen in B with probability one.

The third measure $\inf_{\mathcal{A}\in A} P_{\mathcal{A}}[B \Rightarrow C]$ does not define so precisely the role played by *Player*(2) in bringing the event B or in taking advantage of it. The fact that an execution falls in $(B \Rightarrow C)$ is controlled in a mixed fashion by both the random choices used by the algorithm and by the choices made by *Player*(2). As discussed in the example presented in page 51, this measure corresponds to the measure $\inf_{\mathcal{A}\in A;\, P_{\mathcal{A}}[B]=1} P_{\mathcal{A}}[C]$ only under very specific circumstances. On the other hand neither of the two values $\inf_{\mathcal{A}\in A} P_{\mathcal{A}}[B \Rightarrow C]$ and $\inf_{\mathcal{A}\in A;\, P_{\mathcal{A}}[B]>0} P_{\mathcal{A}}[C \mid B]$ is uniformly bigger than the other.

These considerations show that the first measure $\inf_{\mathcal{A}\in A;\, P_{\mathcal{A}}[B]>0} P_{\mathcal{A}}[C \mid B]$ using probabilistic conditioning is an adequate measure in situations where the "*natural dynamics of the game*" (i.e., the dynamics described by the Π/A structure $(S, X, Y, y_{init}, f, g, \Pi, A)$) are such that the precondition B is *part of the knowl-*

edge of Player(2).[10] Indeed, as we just argued, this measure implicitly gives the knowledge of B to *Player*(2). The measure is adequate, i.e., passive, exactly when the dynamics ensure naturally that *Player*(2) holds that knowledge. A typical example of this situation is obtained when B represents a knowledge that *Player*(2) can have acquired during an execution.

Similarly, the second measure $\inf_{\mathcal{A}\in A;\, P_{\mathcal{A}}[B]=1} P_{\mathcal{A}}[C]$ using conditioning by adversary is adequate in situations where the dynamics of the Π/A structure are such that the precondition B *depends solely* of *Player*(2).[11] This was the situation in the example presented previously in this section: *Player*(2) was "naturally" the only entity deciding the schedule. This situation – where B represents a scheduling decision depending solely of *Player*(2) – provides a typical example where the measure $\inf_{\mathcal{A}\in A;\, P_{\mathcal{A}}[B]=1} P_{\mathcal{A}}[C]$ is adequate.

On the other hand, as we saw, the fact that an execution falls in $(B \Rightarrow C)$ is controlled in a mixed fashion both by the random choices used by the algorithm and by the choices made by *Player*(2). Our third measure $\inf_{\mathcal{A}\in A} P_{\mathcal{A}}[B \Rightarrow C]$ is therefore adequate under only very special dynamics. For all practical purposes, we will deem it inadequate.

The notions of "sole dependence" and "knowledge of player(2)".
We introduced in page 54 the notions of event-schemas that *depend solely* of *Player*(2) and which are *part of the knowledge of Player*(2). We now formalize these notions. Both require to use with precision the model of a Π/A structure presented in Definition 2.3.1, page 36, and the description of the associated probability schema $(\Omega_{\mathcal{A}}, \mathcal{G}_{\mathcal{A}}, P_{\mathcal{A}})_{\mathcal{A}\in A}$ presented in page 41. Recall in particular from the construction given in page 41 that the random variables $A_1, A_3, A_5, \ldots$ are the actions taken by *Player*(2) following its strategy $\mathcal{A}$, and that the random variables $A_2, A_4, \ldots$ are the actions taken by *Player*(1) following its strategy π.[12]

Definition 3.2.1 *An event schema $B = (B_{\mathcal{A}})_{\mathcal{A}\in A}$ is said to depend solely on Player*(2) *if, for every $\mathcal{A} \in A$, conditioned on the σ-field $\sigma(A_1, A_3, A_5, \ldots)$,*[13] *$B_{\mathcal{A}}$ is*

[10] We will formalize this statement later in Definition 3.2.2.

[11] We will formalize this statement later in Definition 3.2.1.

[12] Eventhough sharing the same notation A, the notion of actions $A_1, A_2, \ldots$ is distinct from the notion of set of admissible adversaries $A = \{\mathcal{A}_j\}_{j\in J}$. Note also that, eventhough there is no subscript $\mathcal{A}$ to emphasize that fact, the random variables $A_k, S_k, X_k, Y_k; k \geq 1$ defined in page 41 are defined with respect to a given adversary $\mathcal{A}$.

[13] We cannot say "conditioned on the values taken by $A_1, A_2, \ldots$" because there are in general *infinitely many* A_k and we cannot fix *all of them* when conditioning. We therefore have to resort to the more general notion of conditioning with respect to a σ-field. (See [56], page 211. See also

independent of the σ-field $\sigma(A_2, A_4, A_4, \ldots)$.

Recall that $\mathcal{G}_{\mathcal{A}}$ is by definition equal to $\sigma(A_k; k \geq 1)$.[14] Hence the previous condition is equivalent to: "conditioned on $\sigma(A_1, A_3, A_5, \ldots)$, $B_{\mathcal{A}}$ is independent of $\mathcal{G}_{\mathcal{A}}$".

Note also that the independence condition of this definition is trivially verified if $B_{\mathcal{A}}$ can be characterized solely in terms of the actions of *Player*(2) i.e., if $B_{\mathcal{A}} \in \sigma(A_1, A_3, A_5, \ldots)$. Indeed, in that case, when conditioned on $\sigma(A_1, A_3, A_5, \ldots)$, $B_{\mathcal{A}}$ "is a constant" and hence independent of $\mathcal{G}_{\mathcal{A}}$.[15] This is a satisfactory fact: we would expect that events that depend syntactically only on the actions $A_1, A_3, \ldots$ of *Player*(2) do also "depend solely on *Player*(2)" in the sense of Definition 3.2.1! Our more complex definition takes into account that some events B might not be expressible uniquely in terms of the random variables $A_1, A_3, \ldots$ but still result in adequate measures of the type $\inf_{\mathcal{A} \in A;\, P_{\mathcal{A}}[B]=1} P_{\mathcal{A}}[C]$ when enforcing B.[16] We therefore provide now some motivations and justifications at how our definition captures that fact.

As we just recalled, we introduced the notion of an event B "depending solely on" *Player*(2) to justify the adequacy of measures where *Player*(2) enforces the precondition B. The conditional independence of B and of the random choices of the algorithm ensures that, eventhough the definition of B might also involve the choices of the algorithm, $A_2, A_4, \ldots$, these choices do not influence whether B occurs or not. Hence the occurrence of B depends only on the way values are allocated to $A_1, A_3, \ldots$, i.e., on the adversary. (Recall that, in Definition 2.3.1, page 36, we defined an adversary to be a family $\mathcal{A} = (\Omega_y, \mathcal{G}_y, P_y)_{y \in Y}$ i.e., the family of laws[17] of the random variables $A_1, A_3, \ldots$) Therefore restricting the analysis to within B (the precondition of "if B then C") corresponds to considering only adversaries ensuring that B occurs i.e., such that $P_{\mathcal{A}}[B] = 1$.

This shows that the measure $\inf_{\mathcal{A} \in A;\, P_{\mathcal{A}}[B]=1} P_{\mathcal{A}}[C]$ is an adequate formalization of "the probability of C under condition B" if B depends solely on *Player*(2).

---

the footnote 3 above.)

[14] See page 41.

[15] A formal proof of this fact is as follows. The formal definition of conditional expectations recalled in footnote 3 expresses that, for every $f \in L^2(dP)$, the conditional expectation $E[f \mid \mathcal{G}']$ is characterized by the property: $\forall \phi \in L^2(dP) \cap \mathcal{G}'$, $E[\phi f] = E\big[\phi\, E[f \mid \mathcal{G}']\big]$. Hence, for every $\phi \in L^2(dP) \cap \mathcal{G}'$ and every $g \in L^2(dP)$, $E[\phi f g] = E\big[\phi\, E[fg \mid \mathcal{G}']\big]$. On the other hand, if by assumption f is in $\mathcal{G}'$ we also have $E[\phi f g] = E\big[\phi f\, E[g \mid \mathcal{G}']\big]$. Hence, in that case, $\forall \phi \in L^2(dP) \cap \mathcal{G}'$, $E\big[\phi E[fg \mid \mathcal{G}']\big] = E\big[\phi f E[g \mid \mathcal{G}']\big]$. This implies that $E[fg \mid \mathcal{G}'] = fE[g \mid \mathcal{G}'] = E[f \mid \mathcal{G}']E[g \mid \mathcal{G}']$ P-almost surely. As g is arbitrary in $L^2(dP)$ this precisely means that, conditioned on $\mathcal{G}'$, f is independent of $\mathcal{G}$.

[16] See our discussion about adequate measures in page 53.

[17] See Definition 8.1.2, page 198, for a definition of of the notion of law of a random variable.

We conclude this discussion about events "depending solely" of *Player*(2) with two caveats. To begin, note that even if an event-schema depends solely on *Player*(2), there might exist no strategy $\mathcal{A}$ such that $P_{\mathcal{A}}[B] = 1$. The reason is that the actions A_k are random variables and hence not in the full control of *Player*(2).

This brings us to the second point. A special but important case is when, as is the assumption in this chapter, the strategies of *Player*(2) are all deterministic. In that case, each action A_k taken by *Player*(2) depends deterministically on the view Y_k: Y_k represents the knowledge held by *Player*(2) at level k. Hence, in this case, the condition presented in Definition 3.2.1 is equivalent to $Y_1, Y_2, \ldots$ determining completely B. Nevertheless it is possible that B be not characterized by a single variable Y_k. In that case, there is *no* point k at which *Player*(2) *knows* B, eventhough B depends solely on that player. A justification for this apparent paradox is that the sequence of views of *Player*(2) characterizes B, but *Player*(2) lacks at each single point the possibility to assess more then one value of the sequence $(Y_k)_{k\in\mathbb{N}}$. We can encounter such situations – where some B depends solely on the views of *Player*(2) but where *Player*(2) does not "know" it – if, for instance, *Player*(2) does not remember completely the past (i.e., if the the past values $Y_1, \ldots, Y_{k-1}$ are not recorded in the view Y_k) or if B depends on infinitely many views Y_k.

Definition 3.2.2 *An event schema $(B_{\mathcal{A}})_{\mathcal{A}\in A}$ is said to be part of the knowledge of Player*(2) *if there exist a (random) step number $k_{\mathcal{A}}$ such that $B_{\mathcal{A}}$ is measurable with respect to the view $Y_{k_{\mathcal{A}}}$.*

By definition, $B_{\mathcal{A}}$ is measurable with respect to $Y_{k_{\mathcal{A}}}$ if there is a (deterministic) function $f_{\mathcal{A}}$ such that the indicator of $B_{\mathcal{A}}$ is equal to $f_{\mathcal{A}}(Y_{k_{\mathcal{A}}})$. As mentioned in the caveat above, the knowledge accessible to *Player*(2) at each point k is completely described by the variable Y_k. Our definition therefore formalizes well that B is part of the knowledge of *Player*(2). We allow $k_{\mathcal{A}}$ to be random to take into account the fact that information becomes available to the players at random times.

Composition of adequate correctness-measures.
The two types of measure $\inf_{\mathcal{A}\in A;\, P_{\mathcal{A}}[B]>0} P_{\mathcal{A}}[C \mid B]$ and $\inf_{\mathcal{A}\in A;\, P_{\mathcal{A}}[B]=1} P_{\mathcal{A}}[C]$ can be combined to yield an adequate composite measure [18] in the following situations. *Player*(2) holds some specific knowledge B_1 and uses this knowledge to enforce a condition B_2, which depends solely on him, trying to minimize the probability

[18] See our discussion about adequate measures in page 53.

of occurrence of an event C. In this case, the probability of occurrence of C is adequately[18] measured by:

$$\inf_{\substack{\mathcal{A};P_{\mathcal{A}}[B_2|B_1]=1 \\ P_{\mathcal{A}}[B_1]>0}} P_{\mathcal{A}}[C|B_1]\,.$$

We actually have to be more precise to ensure that this measure is adequate[18]. In Definition 3.2.1, the formalization of "B depends solely of *Player*(2)" is made (implicitly) in terms of the probability schema $(\Omega_{\mathcal{A}}, \mathcal{G}_{\mathcal{A}}, P_{\mathcal{A}})_{\mathcal{A}\in A}$.[19] In contrast, the formalization of "B_2 depends solely on *Player*(2)" is made conditioned on B_1, i.e., is made in terms of the probability schema

$$(B_{1,\mathcal{A}},\, \mathcal{G}_{\mathcal{A}} \cap B_{1,\mathcal{A}},\, P_{\mathcal{A}}[\,\cdot\,]/P_{\mathcal{A}}[B_{1,\mathcal{A}}])_{\mathcal{A}\in A'}$$

where $B_1 = (B_{1,\mathcal{A}})_{\mathcal{A}\in A}$, $\mathcal{G}_{\mathcal{A}} \cap B_1 \stackrel{\text{def}}{=} \{C \cap B_1; C \in \mathcal{G}_{\mathcal{A}}\}$ and where $A' \stackrel{\text{def}}{=} \{\mathcal{A} \in A;\, P_{\mathcal{A}}[B_1] > 0\}$.

We now generalize this construction. Our discussion involves a sequence of event-schemas $B_1, B_2, \ldots$ To justify the validity of our argument we make the following assumption formalizing that, for every k, B_k happens before B_{k+1} in the execution. (This requirement might possibly be relaxed and the argument generalized.)

Assumption. There exists an increasing sequence of (random) values $n_1, n_2, \ldots$ such that, for every k, $B_k \in \sigma(Y_{n_k}, Y_{n_k+1}, \ldots, Y_{n_{k+1}-1})$.

Assume that the natural dynamics[20] of the Π/A structure ensure the following. *Player*(2) holds the knowledge of B_1. Using this knowledge *Player*(2) decides single-handedly on B_2: by assumption B_2 depends solely on *Player*(2). (As above the formal translation of this fact requires the use of the probability schema $(B_{1,\mathcal{A}}, \mathcal{G}_{\mathcal{A}} \cap B_1, P_{\mathcal{A}}[\,\cdot\,]/P_{\mathcal{A}}[B_{1,\mathcal{A}}])_{\mathcal{A}\in A'}$.) *Player*(2) then observes B_3 and uses this knowledge to decide on some B_4 that depends solely on him ... We symbolically let

$$B_1 \mid B_2 \rightarrow B_3 \mid B_4 \rightarrow \ldots$$

denote such scenarios. To each scenario, we can associate an adequate correctness-measure by iterating the previous procedure. For instance, an adequate correctness

[19] The notion of probability schema is presented in Definition 2.4.1.
[20] The notion of natural dynamics is presented in page 54.

measure associated to a scenario $B_1 \mid B_2 \rightarrow B_3$ is given by:

$$\inf_{\substack{\mathcal{A};P_{\mathcal{A}}[B_3|B_1,B_2]>0 \\ P_{\mathcal{A}}[B_2|B_1]=1 \\ P_{\mathcal{A}}[B_1]>0}} P_{\mathcal{A}}[C|B_1,B_3]\,.$$

We thus obtain a whole family of (rather complex) adequate high-probability correctness measures. A special but important case where we can easily write such adequate measures is when, for (each) odd index $2i-1$, the precondition B_{2i-1} is part of the knowledge of *Player*(2) and when B_{2i} is described in terms of the action taken next by *Player*(2).[21] Indeed, in that case, the precondition B_{2i} depends solely on *Player*(2) (conditioned on $B_1, \ldots, B_{2i-1}$).

Summary and open questions about adequate measures.
We saw that there are two extremes when formalizing a high-probability statement of the form "for all adversaries, if property B holds then property C holds with probability at least ε." In one extreme, we restrict the analysis to within B, leaving *Player*(2) free to select the most damaging strategy it wishes. This leads to the measure

$$\inf_{\mathcal{A}\in A} P_{\mathcal{A}}[C \mid B]\,.$$

This measure might be "unfair" to *Player*(1) as it gives to *Player*(2) some knowledge that it might not receive otherwise according to the natural dynamics of the Π/A structure considered. The other extreme is to require that the adversary ensures B with probability one. This leads to the measure

$$\inf_{\mathcal{A}\in A;\, P_{\mathcal{A}}[B]=1} P_{\mathcal{A}}[C]\,.$$

This measure can be "unfair" to *Player*(2) as it forces that player to guaranty alone an event which might depend on both players. Any other high probability measure M formalizing the statement above must fall between these two measures:

$$\inf_{\mathcal{A}\in A;\, P_{\mathcal{A}}[B]>0} P_{\mathcal{A}}[C \mid B] \leq M \leq \inf_{\mathcal{A}\in A;\, P_{\mathcal{A}}[B]=1} P_{\mathcal{A}}[C]\,.$$

[21] We formalize here this statement. We will use the convenient labeling presented on page 65 to describe an execution: $\omega = \underbrace{A_1\,(S_1,X_1,Y_1)}_{Player(2)}\ \underbrace{A'_1\,(S'_1,X'_1,Y'_1)}_{Player(1)}\ \underbrace{A_2\,(S_2,X_2,Y_2)}_{Player(2)}\ldots$. With these conventions a formal translation of our statement is that, for every $\mathcal{A}$ in A, there exists a (random) index $k_{\mathcal{A}}$ such that $B_{2i-1,\mathcal{A}} \in \sigma(Y'_{k_{\mathcal{A}}})$ and such that $B_{2i,\mathcal{A}} \in \sigma(A_{k_{\mathcal{A}}})$

It would be very interesting to be able to characterize the set of adequate measures of a given randomized algorithm.

Existence. In particular, an interesting question is whether there always exists an adequate[22] formalization of the expression "the highest probability of C under condition B". If not this would mean that some correctness statements are by nature ill-formed. In particular, we would like to know whether Rabin's property quoted in page 61 can be adequately formalized in connection with the Π/A structure presented on page 64.

Completeness. We saw that the set of actions of *Player*(2) depended solely[23] of her. Combining such actions with events that are part of her knowledge[23] allowed us to derive (infinitely many) adequate measures associated to a randomized algorithm. Are all the adequate measures of this type? This would show that the method of conditioning by adversary and of probabilistic conditioning[24] are "basis" for all other adequate measures.

3.3 Proving Lower Bounds

The previous section was devoted to formalizing adequately a statement informally stated. We consider in this section the technical problem to provide a lower bound for an expression already formalized into the form $\inf_{\mathcal{A}\in A} P_{\mathcal{A}}[W \mid I]$. In general it is difficult to estimate directly this expression. Indeed recall that W and I are event schemas and that we actually have to estimate $\inf_{\mathcal{A}\in A} P_{\mathcal{A}}[W_{\mathcal{A}} \mid I_{\mathcal{A}}]$.

Let $(\Omega, \mathcal{G}, P)$ be the probability space associated to the random inputs ι used by the algorithm. (Rabin's algorithm is using two independent coins so that this probability space is easily defined in that case.) Our method for proving a high probability correctness property of the form $P_{\mathcal{A}}[W|I]$ consists of proving successive lower bounds:

$$\begin{aligned} P_{\mathcal{A}}[W \mid I] &\geq P_{\mathcal{A}}[W_1 \mid I_1] \\ &\vdots \\ &\geq P_{\mathcal{A}}[W_r \mid I_r], \end{aligned}$$

where all the W_i and I_i are event schemas, and where the last two event schemas, W_r and I_r, are true events in $\mathcal{G}$ and do not depend on $\mathcal{A}$. The final term, $P_{\mathcal{A}}[W_r \mid I_r]$,

[22] See our discussion about adequate measures in page 53.
[23] See Definitions 3.2.1 and 3.2.2.
[24] See page 54

is then evaluated (or bounded from below) using the distribution P. This method can be in practice rather difficult to implement as it involves disentangling the ways in which the random choices made by the processes affect the choices made by the adversary.

3.4 Rabin's Algorithm

The problem of *mutual exclusion* [16] involves allocating an indivisible, reusable resource among n competing processes. A mutual exclusion algorithm is said to guarantee *progress*[25] if it continues to allocate the resource as long as at least one process is requesting it. It guarantees *no-lockout* if every process that requests the resource eventually receives it. A mutual exclusion algorithm satisfies *bounded waiting* if there is a fixed upper bound on the number of times any competing process can be bypassed by any other process. In conjunction with the progress property, the bounded waiting property implies the no-lockout property. In 1982, Burns et al.[12] considered the mutual exclusion algorithm in a distributed setting where processes communicate through a shared read-modify-write variable. For this setting, they proved that any *deterministic* mutual exclusion algorithm that guarantees progress and bounded waiting requires that the shared variable take on at least n distinct values. Shortly thereafter, Rabin published a *randomized* mutual exclusion algorithm [49] for the same shared memory distributed setting. His algorithm guarantees progress using a shared variable that takes on only $O(\log n)$ values.

It is quite easy to verify that Rabin's algorithm guarantees mutual exclusion and progress; in addition, however, Rabin claimed that his algorithm satisfies the following informally-stated *strong no-lockout* property[26].

> "*If process i participates in a trying round of a run of a computation by the protocol and compatible with the adversary, together with $0 \leq m-1 < n$ other processes, then the probability that i enters the critical region at the end of that round is at least c/m, $c \sim 2/3$.*" (*)

This property says that the algorithm guarantees an approximately equal chance of success to all processes that compete at the given round. Rabin argued in [49] that

[25] We give more formal definitions of these properties in Section 3.5.

[26] In the statement of this property, a "trying round" refers to the interval between two successive allocations of the resource, and the "critical region" refers to the interval during which a particular process has the resource allocated to it. A "critical region" is also called a "critical section".

a good randomized mutual exclusion algorithm should satisfy this strong no-lockout property, and in particular, that the probability of each process succeeding should depend inversely on m, the number of *actual* competitors at the given round. This dependence on m was claimed to be an important advantage of this algorithm over another algorithm developed by Ben-Or (also described in [49]); Ben-Or's algorithm is claimed to satisfy a weaker no-lockout property in which the probability of success is approximately c/n, where n is the total number of processes, i.e., the number of *potential* competitors.

Rabin's algorithm uses a randomly-chosen round number to conduct a competition for each round. Within each round, competing processes choose lottery numbers randomly, according to a truncated geometric distribution. One of the processes drawing the largest lottery number for the round wins. Thus, randomness is used in two ways in this algorithm: for choosing the round numbers and choosing the lottery numbers. The detailed code for this algorithm appears in Figure 3.1.

We begin our analysis by presenting three different formal versions of the no-lockout property. These three statements are of the form discussed in the introduction and give lower bounds on the (conditional) probability that a participating process wins the current round of competition. They differ by the nature of the events involved in probabilistic conditioning, those involved in conditioning by adversary and by the values of the lower bounds.

Described in this formal style, neither of the two forms of conditioning – probabilistic conditioning and conditioning by adversary – provides an adequate formalization[27] of the fact that m processes participate in the round. We show in Theorem 3.6.1 that, if probabilistic conditioning is selected, then the adversary can use this fact in a simple way to lock out any process during any round.

On the other hand, the *weak* c/n no-lockout property that was claimed for Ben-Or's algorithm involves only conditioning over events that describe the knowledge of the adversary at the end of previous round. We show in Theorems 3.6.2 and 3.6.4 that the algorithm suffers from a different flaw which bars it from satisfying even this property.

We discuss here informally the meaning of this result. The idea in the design of the algorithm was to incorporate a mathematical procedure within a distributed context. This procedure allows one to select with high probability a unique random element from any set of at most n elements. It does so in an efficient way using a distribution of small support ("small" means here $O(\log n)$) and is very similar to the approximate counting procedure of [20]. The mutual exclusion problem in a

[27] "adequate" in the sense of Section 3.2.

distributed system is also about selecting a unique element: specifically the problem is to select in *each* trying round a unique process among a set of competing processes. In order to use the mathematical procedure for this end and select a true random participating process at each round and for *all choices of the adversary*, it is necessary to discard the old values left in the local variables by previous calls of the procedure. (If not, the adversary could take advantage of the existing values.) For this, another use of randomness was designed so that, with high probability, at each new round, all the participating processes would erase their old values when taking a step.

Our results demonstrate that this use of randomness did not actually fulfill its purpose and that the adversary is able in some instances to use old lottery values and defeat the algorithm.

In Theorem 3.6.6 we show that the two flaws revealed by our Theorems 3.6.1 and 3.6.2 are at the center of the problem: if one restricts attention to executions where program variables are reset, and if we condition by adversary on the number m of participating processes then the strong bound does hold. Our proof, presented in Proposition 3.6.7, highlights the general difficulties encountered in our methodology when attempting to disentangle the probabilities from the influence of $\mathcal{A}$.

The algorithm of Ben-Or which is presented at the end of [49] is a modification of Rabin's algorithm that uses a shared variable of *constant* size. All the methods that we develop in the analysis of Rabin's algorithm apply to this algorithm and establish that Ben-Or's algorithm is similarly flawed and does not satisfy the $1/2en$ no-lockout property claimed for it in [49]. Actually, in this setting, the shared variables can take only two values, which allows the adversary to lock out processes with probability one, as we show in Theorem 3.6.9.

In a recent paper [36], Kushilevitz and Rabin use our results to produce a modification of the algorithm, solving randomized mutual exclusion with $\log_2{}^2 n$ values. They solve the problem revealed by our Theorem 3.6.1 by conducting *before* round k the competition that results in the control of $Crit$ by the end of round k. And they solve the problem revealed by our Theorem 3.6.2 by enforcing in the code that the program variables are reset to 0.

The remainder of this chapter is organized as follows. Section 3.5 contains a description of the mutual exclusion problem and formal definitions of the strong and weak no-lockout properties. Section 3.6 contains our results about the no-lockout properties for Rabin's algorithm. It contains Theorems 3.6.1 and 3.6.2 which disprove in different ways the strong and weak no-lockout properties and Theorem 3.6.6 whose proof is is a model for our methodology: a careful analysis of this proof reveals ex-

actly the origin of the flaws stated in the two previous theorems. One of the uses of randomness in the algorithm was to disallow the adversary from knowing the value of the program variables. Our Theorems 3.6.2 and 3.6.8 express that this objective is not reached and that the adversary is able to infer (partially) the value of *all* the fields of the shared variable. Theorem 3.6.9 deals about the simpler setting of Ben-Or's algorithm.

Some mathematical properties needed for the constructions of Section 3.6 are presented in an appendix (Section 3.7).

3.5 The Mutual Exclusion Problem

The problem of *mutual exclusion* is that of continually arbitrating the exclusive ownership of a resource among a set of competing processes. The set of competing processes is taken from a universe of size n and changes with time. A solution to this problem is a distributed algorithm described by a program (code) $\mathcal{C}$ having the following properties. All involved processes run the same program $\mathcal{C}$. $\mathcal{C}$ is partitioned into four regions, Try, $Crit$, $Exit$, and Rem which are run cyclically in this order by all processes executing $\mathcal{C}$. A process in $Crit$ is said to hold the resource. The indivisible property of the resource means that at any point of an execution, at most one process should be in $Crit$.

3.5.1 Definition of Runs, Rounds, and Adversaries

In this subsection, we define the notions of *run*, *round*, *adversary*, and *fair* adversary which we will use to define the properties of *progress* and *no-lockout*.

A *run* ρ of a (partial) execution ω is a sequence of triplets $\{(p_1, old_1, new_1), (p_2, old_2, new_2), \ldots (p_t, old_t, new_t) \ldots\}$ indicating that process p_t takes the t^{th} step in ω and undergoes the region change $old_t \rightarrow new_t$ during this step (e.g., $old_t = new_t = Try$ or $old_t = Try$ and $new_t = Crit$). We say that ω is *compatible* with ρ.

An *adversary* for the mutual exclusion problem is a mapping $\mathcal{A}$ from the set of finite runs to the set $\{1, \ldots, n\}$ that determines which process takes its next step as a function of the current partial run. That is, the adversary is *only* allowed to see the changes of regions. For every t and for every run $\rho = \{(p_1, old_1, new_1), (p_2, old_2, new_2), \ldots\}$, $\mathcal{A}[\{(p_1, old_1, new_1), \ldots, (p_t, old_t, new_t)\}] = p_{t+1}$. We then say that ρ and $\mathcal{A}$ are *compatible*.

The associated Π/A-structure. We show how these definitions can be formalized into a Π/A-structure within the general model presented in Definition 2.3.1, page 36. Recall that we consider here only deterministic adversaries, i.e., adversaries for which, for every k, the k^{th} action of $Player(2)$, A_k, is a deterministic function of the view Y_k.

To simplify the exposition we will slightly change the notations of Chapter 2, page 41, in the following way. We write here a random execution ω as

$$\omega = \underbrace{A_1\,(S_1, X_1, Y_1)}_{Player(2)}\ \underbrace{A'_1\,(S'_1, X'_1, Y'_1)}_{Player(1)}\ \underbrace{A_2\,(S_2, X_2, Y_2)}_{Player(2)} \ldots$$

A_k is the k^{th} action taken by $Player(2)$. S_k, X_k and Y_k are respectively the state of the system, the view of $Player(1)$ and the view of $Player(2)$ resulting after this action. Similarly we let A'_k, S'_k, X'_k and Y'_k denote the k^{th} action taken by $Player(1)$, the state of the system, and the views of the two players resulting after this action.

- The set of states, S, is the set of tuples containing the value of the program counters $pc_1, \ldots, pc_n$, the values of all the local variables and the value of the shared variable.
- The actions A_k take values in $\{1, \ldots, n\}$.
- The actions A'_k are described by the code of the algorithm given in Figure 3.1, page 74. (We will not make these actions more explicit.)
- The views X_k take value in $S \times \{1, \ldots, n\}$. (If the view of $Player(1)$ is (s, i), then i represents the process to take a step next.)
- The views Y_k take value in $\{runs\ of\ length\ (k-1)\} \times \{1, \ldots, n\}$. ($Y_k = (y, i)$ means that the previous view of $Player(2)$ was $Y'_{k-1} = y$ and $Player(2)$ just selected i.)
- The set of views X'_k is equal to S. ($Player(1)$ just remembers the state of the system after its step.)
- The views Y'_k take value in $\{runs\ of\ length\ k\}$.

The update functions are described as follows. Assume that $(S_k, X_k, Y_k) = (s, (x, i), (y, i))$. Then $(S'_k, X'_k, Y'_k) = f(S_k, X_k, Y_k, A'_k) = (s', s', (y, i, new_i))$,[28] where s' is the state of the system after the k^{th} move of $Player(1)$ and new_i is the region reached by

[28] For simplicity we do not recall old_i which is recorded in y.

process i. Similarly assume that $(S'_k, X'_k, Y'_k) = (s, x, y)$. Then $(S_{k+1}, X_{k+1}, Y_{k+1}) = g(s, x, y, i) = (s, (x, i), (y, i))$, where i is the process selected by $Player(2)$ for round $k + 1$.

This defines the Π/A-structure associated to Rabin's randomized algorithm for mutual exclusion. The construction given in Chapter 2, page 36, defines the associated probability schema $(\Omega_{\mathcal{A}}, \mathcal{G}_{\mathcal{A}}, P_{\mathcal{A}})_{\mathcal{A} \in A}$ over which we will conduct the analysis.

In this model $Player(1)$ "forgets" systematically all the past, knows the current state and learns what the last action of $Player(2)$ is. By contrast, we will consider in Chapter 7 an example where $Player(1)$ learns nothing about the moves of $Player(2)$, consequently knows only partially the state, and remembers everything about its past actions.

For every adversary $\mathcal{A}$, an execution ω in $\Omega_{\mathcal{A}}$ is in $Fair_{\mathcal{A}}$ if every process i in Try, $Crit$, or $Exit$ is eventually provided by $\mathcal{A}$ with a step. This condition describes "normal" executions of the algorithm and says that processes can quit the competition only in Rem. The number of states, actions and views being finite we can express $Fair_{\mathcal{A}}$ as an expression involving (only) countably many rectangles.[29] This establishes that $Fair_{\mathcal{A}} \in \mathcal{G}_{\mathcal{A}}$ and that the family $Fair = (Fair_{\mathcal{A}})_{\mathcal{A} \in A}$ is an event-schema. An adversary $\mathcal{A}$ is *fair* if the executions produced under the control of $\mathcal{A}$ are in *Fair* with probability one, i.e., if $P_{\mathcal{A}}[Fair_{\mathcal{A}}] = 1$. This definition was also given in Vardi [58], page 334. $Player(2)$ is *fair* if every $\mathcal{A} \in A$ is *fair*.[30]

In this chapter, we choose the set A of admissible adversaries to be the set of fair adversaries.

A *round* of an execution is the part between two successive entrances to the critical section (or before the first entrance). More specifically, it is a maximal execution fragment of the given execution, containing one transition $Try \to Crit$ at the end of this fragment and no other transition $Try \to Crit$. The round of a run is defined

[29] See page 42 for a definition of a rectangle and page 43 for a definition of an event-schema.

[30] The probabilistic notion of fairness allows more flexibility in the definition of adversaries then requiring that *all* executions be in *Fair*. The following example illustrates that fact. Consider two processes each running the simple code: "Flip a fair coin". This means that process i ($i = 1$ or 2) flips a coin whenever allocated a step by $Player(2)$. An execution is in *Fair* if both processes take infinitely many steps. Consider the adversary $\mathcal{A}$ defined by: "**I**. Allocate steps repeatedly to process 1 until Head comes up. Then allocate steps repeatedly to process 2 until Head comes up. Then go back to **I** and repeat." This adversary is fair in the probabilistic sense: $P_{\mathcal{A}}[Fair] = 1$. Nevertheless it produces (with probability zero) some executions that are not in *Fair*.

similarly. For every k, we let $round(k)$ denote the k^{th} round. Formally, $round(k)$ is a variable-schema[31] $round(k) = (round_{\mathcal{A}}(k))_{\mathcal{A} \in A}$: $round_{\mathcal{A}}(k)$ is the random k^{th} round of the generic execution obtained when the adversary is $\mathcal{A}$. (For completeness we write $round_{\mathcal{A}}(k) = \perp$ if the execution has less then k rounds.)

A process i *participates* in a round if i takes a step while being either in its trying section *Try* or at rest in its section *Rem*. Hence, for a participating process $old_i \in$ *Rem, Try* and $new_i \in$ *Try, Crit*.

3.5.2 The Progress and No-Lockout Properties

Definition 3.5.1 *An algorithm $\mathcal{C}$ that solves mutual exclusion guarantees* progress *if for all fair*[32] *adversaries there is no infinite execution in which, from some point on, at least one process is in its Try region (respectively its $Exit$ region) and no transition $Try \rightarrow Crit$ (respectively $Exit \rightarrow Rem$) occurs.*

Recall that the notion of fair adversary is probabilistic. But the notion of mutual exclusion is not probabilistic: we require that, for all fair adversaries and *all* executions ω in $Fair_{\mathcal{A}}$, ω have the property enunciated in Definition 3.5.1.

We now turn towards the no-lockout property. This property is probabilistic. Its formal definition requires the following notation:

For every adversary $\mathcal{A}$, let $X_{\mathcal{A}}$ denote any generic quantity whose value changes as the execution unfolds under the control of $\mathcal{A}$ (e.g., the value of a program variable). We let $X_{\mathcal{A}}(k)$ denote the value of $X_{\mathcal{A}}$ *just prior* to the last step $(Try \rightarrow Crit)$ of the kth round of the execution. As a special case of this general notation, we define the following.

- $\mathcal{P}_{\mathcal{A}}(k)$ is the set of participating processes in round k. (Set $\mathcal{P}_{\mathcal{A}}(k) = \emptyset$ if ω has fewer then k rounds.) The notation $\mathcal{P}_{\mathcal{A}}(k)$ is consistent with the general notation because the set of processes participating in round k is updated as round k progresses: in effect the definition of this set is complete only at the end of round k. (This fact is at the heart of our Theorem 3.6.1).

- $t_{\mathcal{A}}(k)$ is the total number of steps that are taken by all the processes up to the end of round k.

[31] See Definition 2.4.1.

[32] The mention of the fairness is put here just as a reminder: recall that, in this chapter, all the admissible adversaries are fair.

- $\mathcal{N}_{\mathcal{A}}(k)$ is the set of executions in which all the processes j participating in round k reinitialize their program variables B_j with a new value $\beta_j(k)$ during round k. ($\mathcal{N}$ stands for New-values.) $\beta_j(k)$; $k = 1, 2, \ldots$, $j = 1, \ldots, n$ is a family of iid [33] random variable whose distribution is geometric truncated at $\log_2 n + 4$ (see [49]).

- For each i, $W_{i,\mathcal{A}}(k)$ denotes the set of executions in which process i enters the critical region at the end of round k.

We consistently use the probability theory convention according to which, for any property $\mathcal{S}_{\mathcal{A}}$, the set of executions $\{\omega \in \Omega_{\mathcal{A}} : \omega \text{ has property } \mathcal{S}_{\mathcal{A}}\}$ is denoted as $\{\mathcal{S}_{\mathcal{A}}\}$. Then:

- For each step number t and each execution $\omega \in \Omega_{\mathcal{A}}$ we let $\pi_{t,\mathcal{A}}(\omega)$ denote the run compatible with the first t steps of ω. For any t-steps run ρ, $\{\pi_{t,\mathcal{A}} = \rho\}$ represents the set of executions compatible with ρ. ($\{\pi_{t,\mathcal{A}} = \rho\} = \emptyset$ if ρ has fewer then t steps.) We will use $\pi_{k,\mathcal{A}}$ in place of $\pi_{t(k),\mathcal{A}}$ to simplify notation.

  Note that the definition of $\pi_{t,\mathcal{A}}$ can be made independently of any adversary $\mathcal{A}$, justifying the simpler notation π_t. We nevertheless keep the subscript $\mathcal{A}$ to emphasize that, for every $\mathcal{A}$, $\pi_{t,\mathcal{A}}$ is defined on $(\Omega_{\mathcal{A}}, \mathcal{G}_{\mathcal{A}})$ which depends on $\mathcal{A}$.

- Similarly, for all $m \leq n$, $\{|\mathcal{P}_{\mathcal{A}}(k)| = m\}$ represents the set of executions having m processes participating in round k.

For every $\mathcal{A}$, The quantities $\mathcal{N}_{\mathcal{A}}(k), \{\pi_{t,\mathcal{A}} = \rho\}$, $W_{i,\mathcal{A}}(k)$, $\{|\mathcal{P}_{\mathcal{A}}(k)| = m\}$, $\{i \in \mathcal{P}_{\mathcal{A}}(k)\}$ are sets of executions. We actually easily check that they are all events in the σ-field $\mathcal{G}_{\mathcal{A}}$. The care that we showed by keeping the reference to $\mathcal{A}$ in $\pi_{t,\mathcal{A}}$ is justified here by the fact that $\{\pi_{t,\mathcal{A}} = \rho\}$ is an event in $\mathcal{G}_{\mathcal{A}}$ which does depend on $\mathcal{A}$. These families of events naturally lead to event-schemas $\mathcal{N}(k), \{\pi_t = \rho\}, W_i(k)$, $\{|\mathcal{P}(k)| = m\}$, $\{i \in \mathcal{P}(k)\}$. For instance, $\mathcal{N}(k) = (\mathcal{N}_{\mathcal{A}}(k))_{\mathcal{A} \in A}$ and $\{\pi_t = \rho\} = (\{\pi_{t,\mathcal{A}} = \rho\})_{\mathcal{A} \in A}$. The analysis will consider these event schemas in connection with the probability schema $(\Omega_{\mathcal{A}}, \mathcal{G}_{\mathcal{A}}, P_{\mathcal{A}})_{\mathcal{A} \in A}$.

We now present the various no-lockout properties that we want to study. All are possible formalizations of statement (*) given on page 61. Of great significance to us is their adequacy[34]: a measure which is not adequate does not confer valuable information about the algorithm studied. To simplify the discussion we will sometimes

[33] Recall that iid stands for "independent and identically distributed".

[34] See our discussion about the notion of *adequate* measures on page 53. We also review that notion shortly here.

use the notation C, B_1, B_2, B_3 and B_4 in place of $W_i(k)$, $\{i \in \mathcal{P}(k)\}$, $\{\pi_{k-1} = \rho\}$, $\{|\mathcal{P}(k)| = m\}$, and $\mathcal{N}(k)$, respectively. As suggested by our notation, we will consider C to be the target property [35] and B_1, B_2, B_3 and B_4 to be the preconditions. The target property can be considered only in conjunction with the precondition B_1. Hence all the measures we will consider will have B_1 as a precondition. The other preconditions can be introduced or omitted, each choice corresponding to a different measure (actually to two different measures as we now discuss).

We argued at the beginning of this chapter that a measure reflects an actual property of the Π/A-structure only if it does not perturb the dynamics of the game between *Player*(1) and *Player*(2). We then say that the measure is adequate. All the measures that we will consider formalize the preconditions using either probabilistic conditioning or conditioning by adversary.[36] (At this point we know of no other method to construct adequate measures.) As we will see neither of these two methods allows to treat adequately the precondition B_3. This is unfortunate because, as is mentioned in [49], a "good" measure of no-lockout should be expressed in terms of m, the actual number of participating processes in a round. In spite of this fact we will treat the preconditions in the most adequate way, envisioning various alternatives when no adequate formulation is available.

We have actually in mind to compute the probability of the target $W_i(k)$ at different (random) points s_k of the execution. The execution-fragment previous to s_k is (obviously) determined at the point s_k. The definition of Rabin's Π/A-structure implies that *Player*(2) then knows the run associated to that execution-fragment. As is explained on page 54, the natural way to account for this situation is to use probabilistic conditioning on the knowledge held by *Player*(2) at that point s_k. We will consider two cases, when s_k is the beginning of the execution and when it is the beginning of round k. In the latter case the knowledge held by *Player*(2)is the past run ρ. Hence the adequate formalization of this case is obtained by probabilistically conditioning on $B_2 = \{\pi_{k-1} = \rho\}$. In the former case – when s_k is the beginning of the execution – there is no past and the corresponding adequate formalization consists in simply omitting B_2 from the measure.

We now turn to $B_1 = \{i \in \mathcal{P}(k)\}$. As mentioned above the target $C = W_i(k)$ can be considered only when this precondition is considered. In this case the situation is not as clear as for B_2: the fact that a process i participates in round k depends on both the strategy $\mathcal{A}$ used by *Player*(2) and on the random values drawn by the algorithm during the round. Indeed, on the one hand i can participates only if

[35] The notions of "target property" and "precondition" are defined on page 48.
[36] These notions are defined on page 54.

Player(2) plans to schedule it. On the other hand, i can also participate only if no process scheduled before its turn comes does succeed into *Crit*. This depends (in part) on the random lottery values drawn by these processes.

There is one process though, for which the situation is unambiguous: the first one scheduled in round k by *Player*(2). This one depends solely on *Player*(2) (in the sense of Definition 3.2.1). When proving negative results (i.e., disproving the correctness of the algorithm) we can therefore consider that process: if correct, the algorithm should in particular ensure a good success rate for this specific process.

On the other hand, when proving positive results (i.e., proving that the algorithm satisfies some correctness-measure) we will, by default of an adequate measure, give *Player*(2) more power then it would in any adequate situation. Indeed, the correctness of the algorithm in that case implies the correctness for any adequate measure, if any exists. We achieve this by letting *Player*(2) *know* the identity of the process i with respect to which the test is conducted. Formally, this means that we use probabilistic conditioning on $\{i \in \mathcal{P}(k)\}$.

We now turn to B_3, the most important character in the cast of B's. (Recall again that the whole purpose of the algorithm is to achieve a measure of fairness with respect to the m currently participating processes.) Unfortunately, in that case, we have no means in our panoply to interpret adequately this precondition ... unless $m = 1$, in which case statement (*) of page 61 is vacuously true. Indeed, consider first probabilistically conditioning on B_3. This means, as we saw, letting *Player*(2) know the number of participating processes and then letting it act as it wishes (provided that *Player*(2) allows with non-zero probability that $|\mathcal{P}(k)| = m$). This gives a definite power to *Player*(2): *Player*(2) is in fact the main power in the determination of m. Our Theorem 3.6.1 expresses that fact and shows that the algorithm is incorrect ... for the inadequate correctness measure considered.

The other possibility at our disposal is to condition by adversary[37] on B_3. This does not provide an adequate measure either. As we will show in Lemma 3.6.5 this constrains in essence *Player*(2) to give steps to all participating processes when the critical section is still closed. This implies in particular that *Player*(2) cannot play on the order according to which it schedules the processes: that is precisely the weapon used by *Player*(2) to defeat the previous measure. This restriction can be viewed as "unfair" to *Player*(2): it ties its hands in a way that does not correspond to the natural dynamics of Rabin's Π/A-structure. Nevertheless our Theorem 3.6.2 shows that this over-constrained *Player*(2) still manages to defeat the algorithm.

[37] See page 54.

The measure considered being the most restrictive towards *Player*(2), any measure formalizing adequately that $|\mathcal{P}(k)| = m$ (if any exists) would similarly yield a defeat of the algorithm.

This brings us to the last of the B event-schemas: $B_4 = \mathcal{N}(k)$. This event is similarly neither part of the knowledge of *Player*(2)[38] at the beginning of round k nor depending solely on him.[39] Thus, as we already saw several times, both methods of probabilistic conditioning and conditioning by adversary are inadequate. The most unfavorable to *Player*(1) is the probabilistic conditioning method. A correctness result in that case will therefore imply correctness for any other adequate formalization of $\mathcal{N}(k)$, if any exits. This is why probabilistic conditioning is used in Theorem 3.6.6.

We are now at last in a position to present the various measures. We use the following notations. The term *weak* refers to the fact that a $1/n$ lower bound on the probability is sought. The term *strong* refers to a $1/m$ lower bound. The terms/notations i, *run*, m and *renew* refer to B_1, B_2, B_3 and B_4 respectively. The term *knowing* refers to a probabilistic conditioning. For instance (*run*, i, m and *renew*)-knowing summarizes that probabilistic conditioning is performed on B_1, B_2, B_3 and B_4. The term *imposing* refers to a conditioning by adversary. For instance *i-imposing* summarizes that conditioning by adversary is performed on B_1.

The first two definitions involve evaluating the probabilities "at the beginning of round k". The measure $\inf_{\mathcal{A}\in A'} P_{\mathcal{A}}[W_i(k) \mid \pi_{k-1} = \rho]$ used in the first definition is adequate (in the sense defined on page 53). It corresponds to a probabilistic measuring obtained for *Player*(2) sitting at the beginning of round k (and hence knowing the past $(k-1)$-run ρ) and scheduling the first step of round k to process i. As discussed on page 70, this measure is too specific to provide a relevant measure of performance of the algorithm. Nevertheless it is a good measure to establish a negative result: if correct, the algorithm should in particular ensure a good success rate for the specific process i. The measures used in the other two definitions are not adequate (in the sense defined on page 53). We argue this point after the definitions.

Definition 3.5.2 (Weak, Run-knowing, i-imposing, Probabilistic no-lockout) *A solution to the mutual exclusion problem satisfies* weak, run-knowing, **i**-imposing *probabilistic no-lockout whenever there exists a constant c such that, for*

[38] See Definition 3.2.2.
[39] See Definition 3.2.1.

every $k \geq 1$, every $(k-1)$-round run ρ and every process i,

$$\inf_{\substack{\mathcal{A}: P_{\mathcal{A}}\left[\, i \in \mathcal{P}(k) \,\middle|\, \pi_{k-1} = \rho \,\right] = 1 \\ P_{\mathcal{A}}[\pi_{k-1} = \rho] > 0}} P_{\mathcal{A}}\left[\, W_i(k) \mid \pi_{k-1} = \rho \,\right] \geq c/n \,.$$

Definition 3.5.3 (Strong, Run & m-knowing, i-imposing, Probabilistic no-lockout) *The same as in Definition 3.5.2 except that:*

$$\inf_{\substack{\mathcal{A}: P_{\mathcal{A}}\left[\, i \in \mathcal{P}(k) \,\middle|\, \pi_{k-1} = \rho,\ |\mathcal{P}(k)| = m \,\right] = 1 \\ P_{\mathcal{A}}[\pi_{k-1} = \rho,\ |\mathcal{P}(k)| = m] > 0}} P_{\mathcal{A}}\left[\, W_i(k) \;\middle|\; \pi_{k-1} = \rho,\ |\mathcal{P}(k)| = m \,\right] \geq c/m \,.$$

The next definition is the transcription of the previous one for the case where the probability is "computed at the beginning of the execution" (i.e., $s_k = 0$ for all k).

Definition 3.5.4 (Strong, m-knowing, i-imposing, Probabilistic no-lockout) *The same as in Definition 3.5.2 except that:*

$$\inf_{\substack{\mathcal{A}: P_{\mathcal{A}}\left[\, i \in \mathcal{P}(k) \,\middle|\, |\mathcal{P}(k)| = m \,\right] = 1 \\ P_{\mathcal{A}}[|\mathcal{P}(k)| = m] > 0}} P_{\mathcal{A}}\left[\, W_i(k) \;\middle|\; m = |\mathcal{P}(k)| \,\right] \geq c/m \,.$$

We argue that the last two definitions are not adequate: both involve probabilistically conditioning on the number m of participating processes in round k. As mentioned above, the two definitions correspond to *Player*(2) sitting either at the beginning of round k or at the beginning of the execution. In both situations, the value of $|\mathcal{P}(k)|$ is not part of the knowledge[40] of *Player*(2). Therefore probabilistic conditioning on the value m of $|\mathcal{P}(k)|$ yields an inadequate measure of performance.

By integration over ρ we see that an algorithm having the property of Definition 3.5.3 is stronger then one having the property of Definition 3.5.4. Equivalently, an adversary able to falsify Property 3.5.4 is stronger then one able to falsify Property 3.5.3.

3.6 Our Results

Here, we give a little more detail about the operation of Rabin's algorithm than we gave earlier in the introduction. At each round k a new *round number* R is

[40] See Definition 3.2.2.

selected at random (uniformly among 100 values). The algorithm ensures that any process i that has already participated in the current round has $R_i = R$, and so passes a test that verifies this. The variable R acts as an "eraser" of the past: with high probability, a newly participating process does not pass this test and consequently chooses a new random number for its *lottery value* B_i. The distribution used for this purpose is a geometric distribution that is truncated at $b = \log_2 n + 4$: $P[\,\beta_j(k) = l\,] = 2^{-l}$ for $l \leq b-1$. The first process that checks that its lottery value is the highest obtained so far in the round, at a point when the critical section is unoccupied, takes possession of the critical section. At this point the shared variable is reinitialized and a new round begins.

The algorithm has the following two features. First, any participating process i reinitializes its variable B_i at most once per round. Second, the process winning the competition takes at most two steps (and at least one) after the point f_k of the round at which the critical section becomes free. Equivalently, a process i that takes two steps after f_k and does not win the competition cannot hold the current maximal lottery value. (After having taken a step in round k a process i must hold the current round number i.e., $R_i(k) = R(k)$. On the other hand, the semaphore S is set to 0 after f_k. If i held the highest lottery value at its second step after f_k it would pass all three tests in the code and enter the critical section.) We will take advantage of this last property in our constructions.

We are now ready to state our results. The first result states that the strong m-knowing correctness property does not hold unless $n \leq 2$.

Theorem 3.6.1 *The algorithm does not have the strong no-lockout property of Definition (3.5.4) (and hence of Definition 3.5.3). Indeed, if $n \geq 3$, there is an adversary $\mathcal{A}$ such that, for all rounds k, for all $m, 2 \leq m \leq n$,*

$$\begin{cases} P_{\mathcal{A}}[\,W_1(k) \mid m = |\mathcal{P}(k)|\,] = 0 \\ P_{\mathcal{A}}[\,1 \in \mathcal{P}(k) \mid m = |\mathcal{P}(k)|\,] = 1 \\ P_{\mathcal{A}}[\,m = |\mathcal{P}(k)|\,] > 0\,. \end{cases}$$

PROOF. Assume first that $2 \leq m \leq n-1$. Consider the following adversary $\mathcal{A}$. $\mathcal{A}$ does not use its knowledge about the past run ρ (which is granted to *Player*(2) by the Π/A-dynamics), gives one step to process 1 while the critical section is occupied, waits for *Exit* and then adopts the schedule $2, 2, 3, 3, \ldots, n, n, 1$. This schedule brings round k to its end, because of the second property mentioned above (i.e., all processes are scheduled for two steps, one of which when the critical section is empty). This adversary is such that m wins with non zero probability i.e., $P_{\mathcal{A}}[m = |\mathcal{P}(k)|] > 0$. Also 1 is scheduled first so that, obviously, $P_{\mathcal{A}}[1 \in \mathcal{P}(k) \,|\, m = |\mathcal{P}(k)|] =$

Shared variable: $V = (S, B, R)$, where:
$S \in \{0, 1\}$, initially 0
$B \in \{0, 1, \ldots, \lceil \log n \rceil + 4\}$, initially 0
$R \in \{0, 2, \ldots, 99\}$, initially *random*

Code for i**:**
Local variables:
$B_i \in \{0, \ldots, \lceil \log n \rceil + 4\}$, initially 0
$R_i \in \{0, 1, \ldots, 99\}$, initially $\perp$
Code:
while $V \neq (0, B_i, R_i)$ **do**
if $(V.R \neq R_i)$ **or** $(V.B < B_i)$ **then**
$B_i \leftarrow random$
$V.B \leftarrow max(V.B, B_i)$
$R_i \leftarrow V.R$
unlock; lock;
$V \leftarrow (1, 0, random)$
unlock;
** Critical Region **
lock;
$V.S \leftarrow 0$
$R_i \leftarrow \perp$
$B_i \leftarrow 0$
unlock;
** Remainder Region **
lock;

Figure 3.1: Rabin's Algorithm

1. But, for this adversary, $|\mathcal{P}(k)| = m$ happens exactly when process m wins so that $P_{\mathcal{A}}[W_1(k) \mid m = |\mathcal{P}(k)|] = 0$.

Consider now the case where $m = n$. We consider then the adversary which gives one step to process 2 while the critical section is occupied, waits for $Exit$ and then adopts the schedule $1, 1, 3, 3, \ldots, n, n, 2$. As above, this schedule brings round k to its end. Similarly, $P_{\mathcal{A}}[|\mathcal{P}(k)| = n] > 0$, namely when 2 holds the highest lottery value. Also, 1 is scheduled with certainty. We now show that 1 must have a smaller lottery then 2 and hence cannot win access to *Crit*. Indeed, otherwise 1 would win and $|\mathcal{P}(k)|$ would be equal to 2. This is a contradiction as we assume that $|\mathcal{P}(k)|$ is n and as, by assumption, n is bigger then 2. □

The previous result is not too surprising in the light of our previous discussion. The measure

$$\inf_{\substack{\mathcal{A}: P_{\mathcal{A}}\left[1 \in \mathcal{P}(k) \,\middle|\, m = |\mathcal{P}(k)|\right] = 1 \\ P_{\mathcal{A}}[|\mathcal{P}(k)| = m] > 0}} P_{\mathcal{A}}[W_1(k) \mid m = |\mathcal{P}(k)|]$$

is not adequate and $Player(2)$ punishes us for using it: in "normal times", i.e., under the dynamics of the Π/A structure, $Player(2)$ is provided with only incomplete information about the past, and definitely no information about the future. But our inadequate measure gives her the future information $|\mathcal{P}(k)| = m$, allowing her to target a specific strategy against any process.

We now give in Theorem 3.6.2 the more damaging result, stating (1) that, in spite of the randomization introduced in the round number variable R, $Player(2)$ is able to infer the values held in the local variables and (2) that it is able to use this knowledge to lock out a process with probability exponentially close to 1. This result is truly damaging because the measure used is adequate: our result expresses that the algorithm is "naturally" incapable to withstand the machinations of $Player(2)$.

Theorem 3.6.2 *There exists a constant $c < 1$, an adversary $\mathcal{A}$, a round k and a $k-1$-round run ρ such that:*

$$\begin{cases} P_{\mathcal{A}}[W_1(k) \mid \pi_{k-1} = \rho] \leq e^{-32} + c^n \\ P_{\mathcal{A}}[1 \in \mathcal{P}(k) \mid \pi_{k-1} = \rho] = 1 \\ P_{\mathcal{A}}[\pi_{k-1} = \rho] > 0\,. \end{cases}$$

We need the following definition in the proof.

Definition 3.6.1 *Let l be a round. Assume that, during round l, $Player(2)$ adopts the following strategy. It first waits for the critical section to become free, then gives*

one step to process j and then two *steps (in any order) to s other processes. (We will call these* test-processes*.) Assume that at this point the critical section is still available (so that round l is not over). We then say that process j is an s-survivor (at round l).*

The idea behind this notion is that, by manufacturing survivors, *Player*(2)is able to select processes having high lottery values. We now describe in more detail the selection of survivors and formalize this last fact.

In the following we will consider an adversary constructing sequentially a family of s-survivors for the four values $s = 2^{\log_2 n+t}$; $t = -1, \ldots, -5$. Whenever the adversary manages to select a new survivor it stores it, i.e, does not allocates it any further step until the selection of survivors is completed. ($\mathcal{A}$ actually allocates steps to selected survivors, but only very rarely, to comply with fairness. Rarely means for instance once every nT^2 steps, where T is the expected time to select an $n/2$-survivor.) By doing so, $\mathcal{A}$ reduces the pool of test-processes still available. We assume that, at any point in the selection process, the adversary selects the test-processes *at random* among the set of processes still available. (The adversary could be more sophisticated then random, but this is not needed.) Note that a new s-survivor can be constructed with probability one whenever the available pool has size at least $s + 1$: it suffices to reiterate the selection process until the selection completes successfully.

Lemma 3.6.3 *There is a constants d such that for any $t = -5, \ldots, -1$, for any $2^{\log_2 n+t}$-survivor j, for any $a = 0, \ldots, 5$*

$$P_{\mathcal{A}}[\, B_j(l) = \log n + t + a \,] \geq d.$$

PROOF. Let s denote $2^{\log n+t}$. Let j be an s-survivor and $i_1, i_2, \ldots, i_s$ be the test-processes used in its selection. Assume also that j drew a new value $B_j(l) = \beta_j(l)$ (this happens with probability $q_1 = .99$.) Remark that $B_j(l) = \text{Max}\{\, B_{i_1}(l), \ldots, B_{i_s}(l), B_j(l)\}$: if this were not the case, one of the test-processes would have entered $Crit$. As the test processes are selected at random, each of them has with probability .99 a round number different from $R(l)$ and hence draws a new lottery number $\beta_j(l)$. Hence, with high probability $q_2 > 0$, 90% of them do so. The other of them keep their old lottery value $B_j(l-1)$: this value, being old, has lost in previous rounds and is therefore stochastically smaller [41] then a new value $\beta_j(l)$. (An application

[41] A real random variable X is stochastically smaller then another one Y (we write that: $X \leq_{\mathcal{L}} Y$) exactly when, for all $x \in \mathbb{R}$, $P[\, X \geq x \,] \leq P[\, Y \geq x \,]$. Hence, if $X \leq Y$ in the usual sense, it is also stochastically smaller.

of Lemma 8.1.5 formalizes this.) Hence, with probability at least $q_1 q_2$ we have the following stochastic inequality:

$$\mathrm{Max}\{\beta_1(l), \ldots, \beta_{s \cdot 90/100}\} \leq_{\mathcal{L}} B_j(l) \leq_{\mathcal{L}} \mathrm{Max}\{\beta_1(l), \ldots, \beta_{s+1}(l)\}.$$

Corollary 3.7.4 then shows that, for $a = 0, \ldots, 5$, with probability at least $q_1 q_2$, $P_{\mathcal{A}}[B_j(l) = \log_2 s + a] \geq q_3$ for some constant q_3 (q_3 is close to 0.01). Hence, with probability at least $d \stackrel{\text{def}}{=} q_1 q_2 q_3$, $B_j(l)$ is equal to $\log_2 s + a$. □

Proof of Theorem 3.6.2. The adversary uses a preparation phase to select and store some processes having high lottery values. We will, by abuse of language, identify this phase with the run ρ which corresponds to it. When this preparation phase is over, round k begins.

Preparation phase ρ: For each of the five values $\log_2 n + t$, $t = -5, \ldots, -1$, $\mathcal{A}$ selects in the preparation phase many ("many" means $n/20$ for $t = -5, \ldots, -2$ and $6n/20$ for $t = -1$) $2^{\log_2 n + t}$-survivors. Let S_1 denote the set of all the survivors thus selected. (Note that $|S_1| = n/2$ so that we have enough processes to conduct this selection). By partitioning the set of $2^{\log_2 n - 1}$-survivors into six sets of equal size, for each of the ten values $t = -5, \ldots, 4$, $\mathcal{A}$ has then secured the existence of $n/20$ processes whose lottery value is $\log_2 n + t$ with probability bigger then d. (By Lemma 3.6.3.)

Round k: While the critical section is busy, $\mathcal{A}$ gives a step to each of the $n/2$ processes from the set S_2 that it did not select in phase ρ. (We can without loss of generality assume that process 1 is in that set S_2: hence $P_{\mathcal{A}}[i \in \mathcal{P}(k)] = 1$ which was to be verified.) When this is done, with probability at least $1 - 2^{-32}$ (see Corollary 3.7.2) the program variable B holds a value bigger or equal then $\log_2 n - 5$. The adversary then waits for the critical section to become free and gives steps to the processes of S_1 it selected in phase ρ. A process in S_2 can win access to the critical section only if the maximum lottery value $B_{S_2} \stackrel{\text{def}}{=} \mathrm{Max}_{j \in S_2} B_j$ of all the processes in S_2 is strictly less then $\log_2 n - 5$ or if no process of S_1 holds both the correct round number $R(k)$ and the lottery number B_{S_2}. This consideration gives the bound predicted in Theorem 3.6.2 with $c = (1 - d/100)^{1/20}$. □

The lesson brought by this last proof is that the variable R does not act as an eraser of the past as it was originally believed and that the adversary can correspondingly use old values to defeat the algorithm.

Furthermore, our proof demonstrates that there is an adversary that can lock out, with probability exponentially close to 1, an arbitrary set of $n/2$ processes during *some* round. With a slight improvement we can derive an adversary that will succeed in locking out (with probability exponentially close to 1) a given set S_3 of, for

example, $n/100$ processes at *all rounds*: we just need to remark that the adversary can do without this set S_3 during the preparation phase ρ. The adversary would then alternate preparation phases $\rho_1, \rho_2, \ldots$ with rounds $k_1, k_2, \ldots$ The set S_3 of processes would be given steps only during rounds $k_1, k_2, \ldots$ and would be locked out at each time with probability exponentially close to 1.

In view of our counterexample we might think that increasing the size of the shared variable might yield a solution. For instance, if the geometric distribution used by the algorithm is truncated at the value $b = 2\log_2 n$ instead of $\log_2 n + 4$, then the adversary is not able as before to ensure a lower bound on the probability that an $n/2$-survivor holds b as its lottery value. (The probability is given by Theorem 3.7.1 with $x = \log n$.) Then the argument of the previous proof does not hold anymore. Nevertheless, the next theorem establishes that raising the size of the shared variable does not help as long as the size stays sub-linear. But this is exactly the theoretical result the algorithm was supposed to achieve. (Recall the n-lower bound of [12] in the deterministic case.) Furthermore, the remark made above applies here also: a set of processes of linear size can be locked out at each time with probability arbitrarily close to 1.

Theorem 3.6.4 *Suppose that we modify the algorithm so that the set of possible round numbers used has size r and that the set of possible lottery numbers has size b ($\log_2 n + 4 \leq b \leq n$). Then there exists positive constants c_1 and c_2, an adversary $\mathcal{A}$, and a run ρ such that*

$$\begin{cases} P_{\mathcal{A}}[\, W_1(k) \mid \pi_{k-1} = \rho,\ 1 \in \mathcal{P}(k)\,] \leq e^{-32} + e^{-c_1 n/r} + c_2 \frac{r}{n^2} \\ P_{\mathcal{A}}[1 \in \mathcal{P}(k) \mid \pi_{k-1} = \rho] = 1 \\ P_{\mathcal{A}}[\pi_{k-1} = \rho] > 0\,. \end{cases}$$

Proof. We consider the adversary $\mathcal{A}$ described in the proof of theorem 3.6.2: for $t = -5, \ldots, -2$, $\mathcal{A}$ prepares a set T_t of $2^{\log_2 n + t}$-survivors, each of size $n/20$, and a set T_{-1} of $2^{\log_2 n - 1}$-survivors; the size of T_{-1} is $6/20n$. (We can as before think of this set as being partitioned into six different sets.) We let η stand for $6/20$ in the sequel.

Let p_l denote the probability that process 1 holds l as its lottery value after having taken a step in round k. For any process j in S_{-1} let also q_l denote the probability that process j holds l as its lottery value at the end of the preparation phase ρ.

The same reasoning as in Theorem 3.6.2 then leads to the inequality:

$$\begin{aligned} &P_{\mathcal{A}}[\, W_1(k) \mid \pi_{k-1} = \rho,\ 1 \in \mathcal{P}(k)\,] \\ &\quad \leq e^{-32} + (1 - e^{-32})(1 - d/r)^{n/20} + \sum_{l \geq \log_2 n + 5} p_l (1 - \frac{q_l}{r})^{\eta n}. \end{aligned}$$

Write $l = \log_2 n + x - 1 = \log_2(n/2) + x$. Then, as is seen in the proof of Corollary 3.7.4, $q_l = e^{-2^{1-\zeta}} 2^{1-\zeta}$ for some $\zeta \in (x, x+1)$. For $l \geq \log_2 n + 5$, x is at least 6 and $e^{-2^{1-\zeta}} \sim 1$ so that $q_l \sim 2^{1-\zeta} \geq 2^{1-x}$. On the other hand $p_l = 2^{-l} = 2^{-x+1}/n$.

Define $\psi(x) \stackrel{\text{def}}{=} e^{-2^{1-x}\eta n/r}$ so that $\psi'(x) = e^{-2^{1-x}\eta n/r} 2^{1-x} \eta n/r$. Then:

$$\begin{aligned}
\sum_{l \geq \log_2 + 5} p_m (1 - \frac{q_m}{r})^{\eta n} &\leq 2/n \sum_{x \geq 6} 2^{-x} (1 - \frac{2^{1-x}}{r})^{\eta n} \\
&\leq 2/n \sum_{x \geq 6} 2^{-x} e^{-(\frac{2^{1-x}}{r} \eta n)} \\
&= 1/n \sum_{x \geq 6} 2^{1-x} e^{-(\frac{2^{1-x}}{r} \eta n)} \\
&= \frac{r}{\eta n^2} \sum_{x \geq 6} \psi'(x) \\
&\leq \frac{r}{\eta n^2} \int_5^{\infty} \psi'(x) dx \\
&= \frac{r}{\eta n^2} [\psi]_5^{\infty} \\
&= \frac{r}{\eta n^2} [1 - e^{-2^{-4} \eta n/r}] \\
&\leq \frac{r}{\eta n^2}.
\end{aligned}$$

□

The next result, Theorem 3.6.6, shows that the two flaws exhibited in Theorems 3.6.1 and 3.6.2 are at the core of the problem: the algorithm does have the strong no-lockout property when we condition by adversary[42] on the property $\{|\mathcal{P}(k)| = m\}$ and when we force the algorithm to draw new values for the modified internal variables.

Conditioning by adversary specifically solves the problem expressed by Theorem 3.6.1: the measure analyzed in that theorem is inadequate[43] and allows too much knowledge to *Player*(2). On the other hand, forcing the adversary to reset to new values the internal variables of the participating processes resolves the problem revealed by (the proof of) Theorem 3.6.2. We will prove these facts in Theorem 3.6.6 for a slightly modified version of the algorithm. Recall in effect that the code given in Page 74 is optimized by making a participating process i draw a new lottery

[42] See page 54 for a definition of conditioning by adversary.
[43] See page 53 for a discussion of adequate measures.

number when it is detected that $V.B < B_i$. For simplicity, we will consider the "de-optimized" version of the code in which only the test $V.R \neq R_i$? causes of a new drawing to occur. It is clear that a correctness statement for that de-optimized algorithm implies a similar result for the original algorithm.

We proved that result before we were fully aware of the notion of adequate measures. Our original result is presented in Proposition 3.6.7 and is based on the notion of restricted adversary presented in Definition 3.6.2. As is shown in Lemma 3.6.5, a restricted adversary is exactly one that decides with probability one the set of participating processes. This result easily establishes the equivalence of Theorem 3.6.6 and Proposition 3.6.7.

Definition 3.6.2 *A step taken after the time at which Crit becomes free in round k is called a* k-real *step. We say that an adversary is* k-restricted *when, in round k, the set of participating processes is composed exactly of the set of processes scheduled when Crit is closed, along with the first process taking a k-real step. (That process might have already taken a step when Crit was closed.) An adversary is said to be restricted when it is k-restricted for every k.*

Notation. We will use the notation $\mathcal{A}'$ (as opposed to $\mathcal{A}$) in the following arguments to emphasize when the adversaries considered are k-restricted.

Lemma 3.6.5 *For every process i, for every round $k \geq 1$, and for every $(k-1)$-round run ρ,*[44]

$$\Big\{\mathcal{A}';\ \mathcal{A}' \text{ is } k\text{-restricted and } P_{\mathcal{A}'}\Big[\,|\mathcal{P}(k)| = m,\ i \in \mathcal{P}(k) \mid \mathcal{N}(k),\ \pi_{k-1} = \rho\,\Big] > 0\Big\}$$

$$= \Big\{\mathcal{A};\ P_{\mathcal{A}}\Big[\,|\mathcal{P}(k)| = m,\ i \in \mathcal{P}(k) \mid \mathcal{N}(k),\ \pi_{k-1} = \rho\,\Big] = 1\Big\}.$$

Proof. Note first that, as by assumption an adversary is deterministic, randomness can affect the decisions of *Player*(2) only through the information *Player*(2) receives from *Player*(1): the strategy of *Player*(1) – i.e., the algorithm – is indeed randomized. A moment of thought based on the description of the function f given on page 65 shows furthermore that, for *Player*(2), the only visible affects of randomness are whether a process in *Try* or *Rem* enters in *Crit* when scheduled while the critical section is free. In particular, in round k *Player*(2) follows a deterministic behavior until it learns the region new_t (either *Try* or *Crit*) reached by the first

[44] Using Convention 8.1.1, page 198, we set $P[B|A] = 0$ whenever $P[A] = 0$.

process i_1 scheduled to take a k-real step. We will use this fact in both directions of the proof.

For a k-restricted adversary, the set $\mathcal{P}(k)$ is by definition defined during that period where $\mathcal{A}$ behaves deterministically. This implies that, for every k-restricted adversary $\mathcal{A}'$, we have $P_{\mathcal{A}'}[|\mathcal{P}(k)| = m,\, i \in \mathcal{P}(k) \mid \mathcal{N}(k),\, \pi_{k-1} = \rho] > 0$ only if $P_{\mathcal{A}'}[|\mathcal{P}(k)| = m,\, i \in \mathcal{P}(k) \mid \mathcal{N}(k),\, \pi_{k-1} = \rho] = 1$. This establishes that

$$\Big\{\mathcal{A}';\, \mathcal{A}' \text{ is } k\text{-restricted and } P_{\mathcal{A}'}\Big[\, |\mathcal{P}(k)| = m,\, i \in \mathcal{P}(k) \mid \mathcal{N}(k),\, \pi_{k-1} = \rho \,\Big] > 0\Big\}$$

$$\subseteq \Big\{\mathcal{A};\, P_{\mathcal{A}}\Big[\, |\mathcal{P}(k)| = m,\, i \in \mathcal{P}(k) \mid \mathcal{N}(k),\, \pi_{k-1} = \rho \,\Big] = 1\Big\}.$$

We now show the converse inclusion. Consider some adversary $\mathcal{A}$ such that $P_{\mathcal{A}}[|\mathcal{P}(k)| = m,\, i \in \mathcal{P}(k) \mid \mathcal{N}(k),\, \pi_{k-1} = \rho] = 1$ for some value m. By the property mentioned at the beginning of the proof, the adversary follows a deterministic behavior until the first k-real step. Call i_1 the process taking that first k-real step. Let l be the number of processes scheduled up to that point (including i_1). Obviously $l \leq m$. Remark that (conditioned on $\mathcal{N}(k)$ and $\pi_{k-1} = \rho$) i_1 enters *Crit* with some non-zero probability when taking its first k-real step. This means that $P_{\mathcal{A}}[|\mathcal{P}(k)| = l,\, i \in \mathcal{P}(k) \mid \mathcal{N}(k),\, \pi_{k-1} = \rho] > 0$. The assumption $P_{\mathcal{A}}[|\mathcal{P}(k)| = m,\, i \in \mathcal{P}(k) \mid \mathcal{N}(k),\, \pi_{k-1} = \rho] = 1$ then implies that $l = m$. This precisely means that $\mathcal{A}$ is k-restricted. □

Theorem 3.6.6 *The algorithm satisfies strong, run and renew-knowing, i and m-imposing probabilistic no-lockout. Equivalently, for every process i, for every round $k \geq 1$, for every $m \leq n$ and for every $(k-1)$-round run ρ we have:*

$$\inf_{\substack{\mathcal{A};\, P_{\mathcal{A}}\left[\, |\mathcal{P}(k)|=m,\, i\in\mathcal{P}(k) \,\middle|\, \mathcal{N}(k),\, \pi_{k-1}=\rho \,\right]=1 \\ P_{\mathcal{A}}[\,\mathcal{N}(k),\, \pi_{k-1}=\rho\,]>0}} P_{\mathcal{A}}\left[\, W_i(k) \mid \mathcal{N}(k),\, \pi_{k-1} = \rho \,\right] \geq \frac{2}{3m}.$$

Proof. This result is a simple consequence of Proposition 3.6.7 whose proof is presented next. In that proposition one considers the set of k-restricted adversaries $\mathcal{A}$ such that $P_{\mathcal{A}}[\,\mathcal{N}(k),\, \pi_{k-1} = \rho,\, i \in \mathcal{P}(k),\, |\mathcal{P}(k)| = m\,] > 0$. This condition is equivalent to the conjunction of the two inequalities $P_{\mathcal{A}}[\,\mathcal{N}(k),\, \pi_{k-1} = \rho] > 0$ and $P_{\mathcal{A}}[\, i \in \mathcal{P}(k),\, |\mathcal{P}(k)| = m \mid \mathcal{N}(k),\, \pi_{k-1} = \rho\,] > 0$. By Lemma 3.6.5 the set of conditions

$$\begin{cases} \mathcal{A} \text{ restricted adversary} \\ P_{\mathcal{A}}\Big[\, i \in \mathcal{P}(k),\, |\mathcal{P}(k)| = m \;\Big|\; \mathcal{N}(k),\, \pi_{k-1} = \rho \,\Big] > 0 \\ P_{\mathcal{A}}[\,\mathcal{N}(k),\, \pi_{k-1} = \rho\,] > 0 \end{cases}$$

is equivalent to

$$\begin{cases} P_{\mathcal{A}}\Big[\, i \in \mathcal{P}(k), |\mathcal{P}(k)| = m \;\Big|\; \mathcal{N}(k), \pi_{k-1} = \rho \,\Big] = 1 \\ P_{\mathcal{A}}\big[\, \mathcal{N}(k), \pi_{k-1} = \rho \,\big] > 0\,, \end{cases}$$

where no a priori restriction applies here to $\mathcal{A}$. Theorem 3.6.6 is therefore a direct consequence of Proposition 3.6.7. □

Note that we proved along that if the adversary is m-imposing and *renew*-knowing then the distinction between an i-knowing and an i-imposing adversary disappears. More formally

$$\{\, \mathcal{A};\, P_{\mathcal{A}}[\, B_3 \mid B_1, B_2, B_4\,] = 1\,\} = \{\, \mathcal{A};\, P_{\mathcal{A}}[\, B_3, B_1 \mid B_2, B_4\,] = 1\,\}\,.^{45}$$

This shows that the way the precondition B_1 is formalized is inconsequential for m-imposing and *renew*-knowing adversaries. On the other hand, as we saw already several times, *the* adequate[46] formalization of the precondition B_2 is obtained by probabilistic conditioning. This establishes that the measure used in Theorem 3.6.6 is as adequate[46] as it can be, provided that the adversary is m-imposing and *renew*-knowing. These two restrictions are brought to solve the two problems revealed in Theorems 3.6.1 and 3.6.2, respectively.

Proposition 3.6.7 *Let i be a process, k a round number, and ρ be a $(k-1)$-round run. For concision of notation we let All denote the event schema $\{\mathcal{N}(k), \pi_{k-1} = \rho, i \in \mathcal{P}(k), |\mathcal{P}(k)| = m\}$. We have:*

$$\inf_{P_{\mathcal{A}'}[All]>0} P_{\mathcal{A}'}\Big[\, W_i(k) \;\Big|\; \mathcal{N}(k), \pi_{k-1} = \rho, i \in \mathcal{P}(k), |\mathcal{P}(k)| = m \,\Big] \geq \frac{2}{3m}\,.$$

Proof. We will make constant use of the notation $[n] \stackrel{\text{def}}{=} \{1, 2, \ldots, n\}$. Also, for any sequence $(a_j)_{j\in\mathbb{N}}$ we will write $a_i = \underset{j \in J}{\text{Umax}} a_j$ to mean that i is the *only* index in J for which $a_i = \underset{j \in J}{\text{Max}} a_j$.

We first define the events $\mathcal{U}(k)$ and $\mathcal{U}'_J(k)$, where J is any subset of $\{1, \ldots, n\}$:

$$\begin{aligned} \mathcal{U}(k) &\stackrel{\text{def}}{=} \{\exists! i \in \mathcal{P}(k) \text{ s.t. } B_i(k) = \underset{j \in \mathcal{P}(k)}{\text{Max}} B_j(k)\}, \\ \mathcal{U}'_J(k) &\stackrel{\text{def}}{=} \{\exists! i \in J \text{ s.t. } \beta_i(k) = \underset{j \in J}{\text{Max}} \beta_j(k)\}. \end{aligned}$$

[45] We never used the fact that we were conditioning on B_2 so that the same equality holds without the mention of B_2. As is discussed in page 69, this correspond to analyzing the system at the point s_k equal to the beginning of the execution.

[46] The term adequate is used in the sense defined on page 53.

The main result established in [49] can formally be restated as:

$$\forall m \leq n,\ P\Big[\mathcal{U}'_{[m]}\ (k)\Big] \geq 2/3. \tag{3.3}$$

Following the general proof technique described in the introduction we will prove that :

$$P_{\mathcal{A}'}\Big[\mathcal{U}(k) \mid\ \mathcal{N}(k),\ \pi_{k-1} = \rho,\ i \in \mathcal{P}(k), |\mathcal{P}(k)| = m\Big] = P\Big[\mathcal{U}'_m(k)\Big],$$

and that:

$$\begin{aligned} & P_{\mathcal{A}'}\Big[W_i(k) \mid \mathcal{N}(k), \pi_{k-1} = \rho, i \in \mathcal{P}(k), |\mathcal{P}(k)| = m, \mathcal{U}(k)\Big] \\ = \quad & P\Big[\beta_i(k) = \operatorname*{Max}_{j \in [m]} \beta_j(k) \mid\ \mathcal{U}'_m(k)\Big]\ . \end{aligned}$$

The events involved in the LHS of the two inequalities (e.g., $W_i(k), \mathcal{U}(k)$, $\{|\mathcal{P}(k)| = m\}$, $\{\pi_{k-1} = \rho\}$, $\{i \in \mathcal{P}(k)\}$) depend on $\mathcal{A}'$ whereas the events involved in the RHS are pure mathematical events over which $\mathcal{A}'$ has no control.

We begin with some important remarks.

(1) By definition, the set $\mathcal{P}(k) = \{i_1, i_2, \ldots\}$ is decided by the restricted adversary $\mathcal{A}'$ at the beginning of round k: for a given $\mathcal{A}'$ and conditioned on $\{\pi_{k-1} = \rho\}$, the set $\mathcal{P}(k)$ is defined *deterministically*. In particular, for any i, $P_{\mathcal{A}'}[\ i \in \mathcal{P}(k) \mid \pi_{k-1} = \rho\ ]$has value 0 or 1. Similarly, there is one value m for which $P_{\mathcal{A}'}[\ |\mathcal{P}(k)| = m \mid\ \pi_{k-1} = \rho\ ] = 1$. Hence, for a given adversary $\mathcal{A}'$, if the random event $\{\mathcal{N}(k),\ \pi_{k-1} = \rho,\ i \in \mathcal{P}(k),\ |\mathcal{P}(k)| = m\}$ has non zero probability, it is equal to the random event $\{\mathcal{N}(k),\ \pi_{k-1} = \rho\} \stackrel{\text{def}}{=} I$.

(2) Recall that, in the modified version of the algorithm that we consider here, a process i draws a new lottery value in round k exactly when $R_i(k-1) \neq R(k)$. Hence, within I, the event $\mathcal{N}(k)$ is equal to $\{R_{i_1}(k-1) \neq R(k), \ldots, R_{i_m}(k-1) \neq R(k)\}$. On the other hand, by definition, the random variables (in short r.v.s) $\beta_{i_j};\ i_j \in \mathcal{P}(k)$ are iid and independent from the r.v. $R(k)$. This proves that, (for a given $\mathcal{A}'$), conditioned on $\{\pi_{k-1} = \rho\}$, the r.v. $\mathcal{N}(k)$ is independent from all the r.v.s β_{i_j}. Note that $\mathcal{U}'_{\mathcal{P}(k)}(k)$ is defined in terms of (i.e., measurable with respect to) the $(\beta_{i_j};\ i_j \in \mathcal{P}(k))$, so that $\mathcal{U}'_{\mathcal{P}(k)}(k)$ and $\mathcal{N}(k)$ are also independent.

(3) More generally, consider any r.v. X defined in terms of the $(\beta_{i_j};\ i_j \in \mathcal{P}(k))$: $X = f(\beta_{i_1}, \ldots, \beta_{i_m})$ for some measurable function f. Recall once more that the number m and the *indices* $i_1, \ldots, i_m$ are determined by $\{\pi_{k-1} = \rho\}$ and $\mathcal{A}'$. The r.v.s β_{i_j} being iid, for a fixed $\mathcal{A}'$, X then depends on $\{\pi_{k-1} = \rho\}$ only through

the value m of $|\mathcal{P}(k)|$. Formally, this means that, conditioned on $|\mathcal{P}(k)|$, the r.v.s X and $\{\pi_{k-1} = \rho\}$ are independent: $E_{\mathcal{A}'}[X \mid \pi_{k-1} = \rho\,] = E_{\mathcal{A}'}[X \mid |\mathcal{P}(k)| = m] = E[f(\beta_1, \ldots, \beta_m)]$. (More precisely, this equality is valid for the value m for which $P_{\mathcal{A}}[\,\pi_{k-1} = \rho\ ,\ |\mathcal{P}(k)| = m\,] \neq 0$.) A special consequence of this fact is that $P_{\mathcal{A}'}[\mathcal{U}'_{\mathcal{P}(k)}(k) \mid \pi_{k-1} = \rho] = P[\mathcal{U}'_{[m]}(k)]$.

Remark that, in $\mathcal{U}(k)$, the event $W_i(k)$ is the same as the event $\{B_i(k) = \underset{j \in \mathcal{P}(k)}{\mathrm{U}\max} B_j(k)\}$. This justifies the first following equality. The subsequent ones are commented afterwards. Also, the set I that we consider here is the one having a non zero probability described in Remark (1) above.

$$\begin{aligned}
& P_{\mathcal{A}'}[\,W_i(k) \mid \mathcal{U}(k),\, I\,] \\
&= P_{\mathcal{A}'}[\,B_i(k) = \underset{j \in \mathcal{P}(k)}{\mathrm{U}\max} B_j(k) \mid \mathcal{U}(k),\, I\,] \\
&= P_{\mathcal{A}'}[\,\beta_i(k) = \underset{j \in \mathcal{P}(k)}{\mathrm{U}\max} \beta_j(k) \mid \mathcal{U}'_{\mathcal{P}(k)}(k),\ I,\,] && (3.4) \\
&= P_{\mathcal{A}'}[\,\beta_i(k) = \underset{j \in \mathcal{P}(k)}{\mathrm{U}\max} \beta_j(k) \mid \mathcal{U}'_{\mathcal{P}(k)}(k), \pi_{k-1} = \rho\,] && (3.5)
\end{aligned}$$

Equation 3.4 is true because we condition on $\mathcal{N}(k)$ and because $\mathcal{U}(k) \cap \mathcal{N}(k) = \mathcal{U}'_{\mathcal{P}(k)}(k)$. Equation 3.5 is true because $\mathcal{N}(k)$ is independent from the r.v.s β_{i_j} as is shown in Remark (2) above.

We then notice that the events $\{\beta_i(k) = \underset{j \in \mathcal{P}(k)}{\mathrm{U}\max} \beta_j(k)\}$ and $\mathcal{U}'_{\mathcal{P}(k)}(k)$ (and hence their intersection) are defined in terms of the r.v.s β_{i_j}. From remark (3) above, the value of Eq. 3.5 depends only on m and is therefore independent of i. Hence, for all i and j in $\mathcal{P}(k)$, $P_{\mathcal{A}'}[W_i(k) \mid \mathcal{U}(k), I\,] = P_{\mathcal{A}'}[W_j(k) \mid \mathcal{U}(k),\ I\,]$.

On the other hand, $\sum_{i \in \mathcal{P}(k)} P_{\mathcal{A}'}[\beta_i(k) = \underset{j \in \mathcal{P}(k)}{\mathrm{U}\max} \beta_j(k) \mid \mathcal{U}'_{\mathcal{P}(k)}(k),\ \pi_{k-1} = \rho\,] = 1$: indeed, one of the β_{i_j} has to attain the maximum.

These last two facts imply that, $\forall i \in \mathcal{P}(k)$,

$$P_{\mathcal{A}'}[W_i(k) \mid \mathcal{U}(k),\ I\,] = 1/m.$$

We now turn to the evaluation of $P_{\mathcal{A}'}[,\mathcal{U}(k) \mid I\,]$.

$$\begin{aligned}
P_{\mathcal{A}'}[\mathcal{U}(k) \mid I\,] &= P_{\mathcal{A}'}[\mathcal{U}'_{\mathcal{P}(k)}(k) \mid I\,] && (3.6) \\
&= P_{\mathcal{A}'}[\mathcal{U}'_{\mathcal{P}(k)}(k) \mid \pi_{k-1} = \rho\,] && (3.7) \\
&= P[\mathcal{U}'_{[m]}(k)\,] \\
&\geq 2/3\,. && (3.8)
\end{aligned}$$

Equation 3.6 is true because we condition on $\mathcal{N}(k)$. Eq. 3.7 is true because $\mathcal{U}'_{\mathcal{P}(k)}(k)$ and $\mathcal{N}(k)$ are independent (See Remark (2) above). The equality of Eq. 3.8 stems from Remark (3) above and the inequality from Eq. 3.3.

We can now finish the proof of Proposition 3.6.7.

$$\begin{aligned} & P_{\mathcal{A}'}[\, W_i(k) \mid I \,] \\ & \quad \geq P_{\mathcal{A}'}[\, W_i(k), \mathcal{U}(k) \mid I \,] \\ & \quad = P_{\mathcal{A}'}[\, W_i(k) \mid \mathcal{U}(k), I \,]\, P_{\mathcal{A}'}[\mathcal{U}(k) \mid I \,] \\ & \quad \geq 2/3\, m\,. \end{aligned}$$

□

We discuss here the lessons brought by our results. (1) Conditioning on $\mathcal{N}(k)$ is equivalent to force the algorithm to refresh all the variables at each round. By doing this, we took care of the undesirable lingering effects of the past, exemplified in Theorems 3.6.2 and 3.6.4. (2) It is *not* true that:

$$\begin{aligned} & P_{\mathcal{A}}\Big[\, \beta_i(k) = \max_{j \,\in\, \mathcal{P}(k)} \beta_j(k) \mid \mathcal{U}'_{\mathcal{P}(k)}(k),\ |\mathcal{P}(k)| = m \,\Big] \\ & \quad = \quad P\Big[\, \beta_i(k) = \max_{j \,\in\, [m]} \beta_j(k) \mid \mathcal{U}'_{[m]}(k) \,\Big]\,, \end{aligned}$$

i.e., that the adversary has no control over the event $\{\beta_i(k) = \max_{j \,\in\, \mathcal{P}(k)} \beta_j(k)\}$. (This was Rabin's statement in [49].) Indeed, the latter probability is equal to $1/m$ whereas we proved in Theorem 3.6.1 that there is an adversary for which the former is 0 when $2 \leq m \leq n$.

The crucial remark explaining this apparent paradox is that, implicit in the expression $P_{\mathcal{A}}[\beta_i(k) = \max_{j \,\in\, \mathcal{P}(k)} \beta_j(k) \mid \ldots]$, is the fact that the random variables $\beta_j(k)$ (for $j \in \mathcal{P}(k)$) are compared to each other in a specific way decided by $\mathcal{A}$, before one of them reveals itself to be the maximum. For instance, in the example constructed in the proof of Theorem 3.6.1, when j takes a step, $\beta_j(k)$ is compared *only* to the $\beta_l(k)$; $l \leq j$, and the situation is not symmetric among the processes in $\mathcal{P}(k)$.

But, if the adversary is restricted as in our Definition 3.6.2, or if equivalently, probabilistic conditioning is done on $|\mathcal{P}(k)| = m$, the symmetry is restored and the strong no-lockout property holds.

Rabin and Kushilevitz used these ideas from our analysis to produce their algorithm [36].

Our Theorems 3.6.1, 3.6.2 and 3.6.4 explored how the adversary can gain and use knowledge of the lottery values held by the processes. The next theorem states that the adversary is similarly able to derive some knowledge about the round numbers, contradicting the claim in [49] that "because the variable R is randomized just

before the start of the round, we have with probability 0.99 that $R_i \neq R$." Note that, expressed in our terms, the previous claim translates into $R(k) \neq R_i(k-1)$. Note also that the next measure is adequate, in the sense defined in page 53.

Theorem 3.6.8 *There exists an adversary $\mathcal{A}$, a round k, a step number t, a run ρ_t, compatible with $\mathcal{A}$, having t steps and in which round k is under way such that*

$$P_{\mathcal{A}}[\, R(k) \neq R_1(k-1) \mid \pi_t = \rho_t \,] < .99 \ .$$

PROOF. We will write $\rho_t = \rho'\rho$ where ρ' is a $k-1$-round run and ρ is the run fragment corresponding to the kth round under way. Assume that ρ' indicates that, before round k, processes $1, 2, 3, 4$ participated *only* in round $k-1$, and that process 5 never participated before round k. Furthermore, assume that during round $k-1$ the following pattern happened: $\mathcal{A}$ waited for the critical region to become free, then allocated one step in turn to processes $2, 1, 1, 3, 3, 4, 4$; at this point 4 entered the critical region. (All this is indicated in ρ'.) Assume also that the partial run ρ into round k indicates that the critical region became free before any competing process was given a step, and that the adversary then allocated one step in turn to processes $5, 3, 3$, and that, after 3 took its last step, the critical section was still free. We will establish that, at this point,

$$P_{\mathcal{A}}[\, R(k) \neq R_1(k-1) \mid \pi_t = \rho'\rho \,] < .99 \ .$$

By assumption $k-1$ is the last (and only) round before round k where processes $1, 2, 3$ and 4 participated. Hence $R_1(k-1) = R_2(k-1) = R_3(k-1) = R(k-1)$. To simplify the notations we will let R' denote this common value. Similarly we will write $\beta'_1,\ \beta'_2, \ldots$ in place of $\beta_1(k-1),\ \beta_2(k-1), \ldots$ We will furthermore write $\beta_1,\ \beta_2, \ldots$ in place of $\beta_1(k),\ \beta_2(k), \ldots$ and $B,\ R$ in place of $B(k),\ R(k)$.

Using Bayes rule gives us:

$$P_{\mathcal{A}}[\, R \neq R' \mid \rho',\ \rho \,] = \frac{P_{\mathcal{A}}[\, R \neq R' \mid \rho' \,]\ P_{\mathcal{A}}[\, \rho \mid \rho', R \neq R' \,]}{P_{\mathcal{A}}[\, \rho \mid \rho' \,]}. \tag{3.9}$$

In the numerator, the first term $P_{\mathcal{A}}[\, R \neq R' \mid \rho' \,]$ is equal to 0.99 because R is uniformly distributed and independent from R' and ρ'. We will use this fact another time while expressing the value of $P_{\mathcal{A}}[\, \rho \mid \rho' \,]$:

$$\begin{aligned} & P_{\mathcal{A}}[\, \rho \mid \rho' \,] \\ & \quad = \ P_{\mathcal{A}}[\, \rho \mid \rho', R \neq R' \,]\ P_{\mathcal{A}}[\, R \neq R' \mid \rho' \,] \end{aligned}$$

$$\begin{aligned} & + P_{\mathcal{A}}[\,\rho \mid \rho', R = R'\,]\, P_{\mathcal{A}}[\,R = R' \mid \rho'\,] \\ = \;& 0.99\; P_{\mathcal{A}}[\,\rho \mid \rho', R \neq R'\,] \\ & + 0.01\; P_{\mathcal{A}}[\,\rho \mid \rho', R = R'\,]. \end{aligned} \tag{3.10}$$

• Consider first the case where $R \neq R'$. Then process 3 gets a YES answer when going through the test "$(V.R \neq R_3)$ **or** $(V.B < B_3)$", and consequently chooses a new value $B_3(k) = \beta_3$. Hence

$$P_{\mathcal{A}}[\,\rho \mid \rho', R \neq R'\,] = P[\,\beta_3 < \beta_5\,]. \tag{3.11}$$

• Consider now the case $R = R'$. By hypothesis, process 5 never participated in the computation before round k and hence draws a new number $B_5(k) = \beta_5$. Hence:

$$P_{\mathcal{A}}[\,\rho \mid \rho', R = R'\,] = P_{\mathcal{A}}[\,B_3(k) < \beta_5 \mid \rho', R = R'\,]. \tag{3.12}$$

As processes $1, \ldots, 4$ participated *only* in round $k-1$ up to round k, the knowledge provided by ρ' about process 3 is exactly that, in round $k-1$, process 3 lost to process 2 along with process 1, and that process 2 lost in turn to process 4, i.e., that $\beta'_3 < \beta'_2$, $\beta'_1 < \beta'_2$ and $\beta'_2 < \beta'_4$. For the sake of notational simplicity, for the rest of this paragraph we let X denote a random variable whose law is the law of β'_2 conditioned on $\{\beta'_2 > \mathrm{Max}\{\beta'_1, \beta'_3\}, \beta'_2 < \beta'_4\}$. This means for instance that, $\forall x \in \mathbb{R}$,

$$P[\,X \geq x\,] = P\Big[\,\beta'_2 \geq x \mid \beta'_2 > \mathrm{Max}\{\beta'_1, \beta'_3\},\ \beta'_2 < \beta'_4\Big]\,.$$

When 3 takes its first step within round k, the program variable $V.B$ holds the value β_5. As a consequence, 3 chooses a new value when and exactly when $B_3(k-1)(= \beta'_3)$ is strictly bigger then β_5. (The case $\beta'_3 = \beta_5$ would lead 3 to take possession of the critical section at its first step in round k, in contradiction with the definition of ρ; and the case $\beta'_3 < \beta_5$ leads 3 to keep its "old" lottery value $B_3(k-1)$.) From this we deduce that:

$$\begin{aligned} P_{\mathcal{A}}[\,B_3(k) < \beta_5 \mid \rho', R = R'\,] = {} & P[\,\beta'_3 < \beta_5 \mid \beta'_3 < X\,] \\ & + P[\,\beta'_3 > \beta_5,\ \beta_3 < \beta_5 \mid \beta'_3 < X\,]. \end{aligned} \tag{3.13}$$

Using Lemma 8.1.5 we derive that:

$$P[\,\beta'_3 < \beta_5 \mid \beta'_3 < X\,] \geq P[\,\beta'_3 < \beta_5\,].$$

On the other hand $P[\,\beta'_3 < \beta_5\,] = P[\,\beta_3 < \beta_5\,]$ because all the random variables $\beta_i(j), i = 1, \ldots, n,\ j \geq 1$ are iid. Taking into account the fact that the last term of equation 3.13 is non zero, we have then established that:

$$P_{\mathcal{A}}[\,B_3(k) < \beta_5 \mid \rho', R = R'\,] > P[\,\beta_3 < \beta_5\,]. \tag{3.14}$$

Combining Equations 3.11, 3.12 and 3.14 yields:

$$P_{\mathcal{A}}[\,\rho \mid \rho', R = R'\,] > P_{\mathcal{A}}[\,\rho \mid \rho',\ R \neq R'\,].$$

Equation 3.10 then shows that $P_{\mathcal{A}}[\rho \mid \rho'] > P_{\mathcal{A}}[\rho \mid \rho',\ R \neq R']$. Plugging this result into Equation 3.9 finishes the proof. □

We finish with a result showing that all the problems that we encountered in Rabin's algorithm carry over for Ben-Or's algorithm. Ben-Or's algorithm is cited at the end of [49]. The code of this algorithm is the same as the one of Rabin with the following modifications. All variables $B, R, B_i, R_i; 1 \leq i \leq n$ are boolean variables, initially 0. The distribution of the lottery numbers is also different but this is irrelevant for our discussion.

We show that Ben-Or's algorithm does not satisfy the weak no-lockout property of Definition 3.5.2. The situation is much simpler then in the case of Rabin's algorithm: here all the variables are boolean so that a simple reasoning can be worked out.

Theorem 3.6.9 (Ben-Or's Alg.) *There is an adversary $\mathcal{A}$, a step number t and a run ρ_t compatible with $\mathcal{A}$ such that*

$$P_{\mathcal{A}}\Big[\,W_2(k) \mid \pi_t = \rho_t,\ 2 \in \mathcal{P}(k)\,\Big] = 0\ .$$

Proof. Assume that we are in the middle of round 3, and that the run ρ_t indicates that (at time 0 the critical section was free and then that) the schedule 1 2 2 3 3 was followed, that at this point 3 entered in $Crit$, that it left $Crit$, that at this point the schedule 4 1 1 5 5 was followed, that 5 entered and then left $Crit$, that 6 4 4 then took a step and that at this point $Crit$ is still free.

Without loss of generality assume that the round number $R(1)$ is 0. Then $R_2(1) = 0$, $B_1(1) = 1$ and $B_2(1) = 0$: if not 2 would have entered in $Crit$. In round 2 it then must be the case that $R(2) = 1$. Indeed if this was not the case then 1 would have entered the critical section. It must then be the case that $B_1(2) = 0$ and $B_4(2) = 1$. And then that $B_6(3) = 1$ and $R(3) = 0$: if this was not the case then 4 would have entered in $Crit$ in the 3rd round.

But at this point, 2 has *no chance* to win if scheduled to take a step! □

3.7 Appendix

This section presents some useful mathematical properties of the truncated exponential distribution used in [49]. Theorem 3.7.1 and its corollaries are used in the

construction of the adversary in Theorem 3.6.2 and Theorem 3.6.4.

Definition 3.7.1 *For any sequence* $(a_i)_{i\in\mathbb{N}}$ *we denote* $Max_s a_i \stackrel{\text{def}}{=} Max\{a_1, a_2, \ldots, a_s\}$.

In this section the sequence (β_i) is a sequence of iid geometric random variables:

$$P[\beta_i = l] = \frac{1}{2^l};\ l = 1, 2, \ldots$$

The following results are about the distribution of the extremal function $\mathrm{Max}_s \beta_i$. The same probabilistic results hold for iid random variables (β'_i), having the truncated distribution used in [49]: we just need to truncate at $\log_2 n + 4$ the random variables β_i and the values that they take. This does not affect the probabilities because, by definition, $P[\,\beta'_i(k) = \log_2 n + 4\,] = \sum_{l\geq \log_2 n+4} P[\,\beta_i = l]$. We will need the following function:

$$\phi(x) \stackrel{\text{def}}{=} 1 - e^{-2^{1-x}}. \tag{3.15}$$

Theorem 3.7.1 *For all* $s \in \mathbb{N}$ *and* $x \in \mathbb{R}$ *such that* $log_2 s + x \in \mathbb{N}$ *and such that* $(\frac{2}{s})^{1-x} \leq 1/2$, *we have the following approximation:*

$$A \stackrel{\text{def}}{=} P[\,Max_s \beta_i \geq log_2 s + x\,] \sim 1 - e^{-2^{1-x}}\ .$$

A bound on the error is given by $\left|A - (1 - e^{-2^{1-x}})\right| \leq e^{-2^{1-x}} \frac{4^{1-x}}{s}$.

Proof. We easily see that, $\forall j \in \mathbb{N}$, $P[\,\mathrm{Max}_s \beta_i < j\,] = (1 - 2^{1-j})^s$. Setting $j = \log_2 s + x$ gives:

$$P[\,\mathrm{Max}_s \beta_i < \log_2 s + x\,] = \left(1 - \frac{2}{s}^{1-x}\right)^s \sim e^{-2^{1-x}}.$$

The upper bound on the error term is obtained by writing a precise asymptotic expansion of $\left(1 - \frac{2}{s}^{1-x}\right)^s$. □

The upper bound on the error shows that this approximation is *very* tight when s is big. In the construction of Theorem 3.6.2 we consider the case where where $s \sim n/2$ and $x = -1/2 \log n$. The error term is then less then $e^{-n^{1/2}}$. As an illustration of the theorem we deduce the two following results.

Corollary 3.7.2 *Consider* $s \geq 1$. *Then* $P[\,Max_s \beta_i \geq \lfloor log_2 s \rfloor - 4\,] \geq 1 - e^{-32}$.

PROOF. Write $\lfloor \log_2 s \rfloor = \log_2 s - t$ with $0 \leq t < 1$. Then

$$\begin{aligned} P[\,\mathrm{Max}_s \beta_i = \lfloor \log_2 s \rfloor - 4\,] &\geq P[\,\mathrm{Max}_s \beta_i \geq \log_2 s - (4+t)\,] \\ &\sim 1 - e^{-2^{1+(4+t)}} \geq 1 - e^{-32}. \end{aligned}$$

□

Corollary 3.7.3 *Consider $s \geq 1$. Then $P[\,Max_s \beta_i \geq \lceil log_2 s \rceil + 8\,] \leq 0.01$.*

PROOF. $1 - e^{-2^{1-8}} \sim 2^{-7} < 0.01$. □

These results express that the maximum of s random variables β_i is concentrated tightly around $\log_2 s$: Corollary 3.7.2 shows that the maximum is with overwhelming probability at least as big as $\lfloor \log_2 s \rfloor - 4$, whereas Corollary 3.7.3 shows that with probability 99% this maximum is at most $\lceil \log_2 s \rceil + 7$.

Corollary 3.7.4 *Let $s \geq 1$. Then $P\Big[\,Max_s \beta_i = \lceil log_2 s \rceil\,\Big] \geq 0.17$. For $a \geq 1$ $P\Big[\,Max_s \beta_i = \lceil log_2 s \rceil + a\,\Big] \geq -\phi'(a+2)$. Hence $P\Big[\,Max_s \beta_i = \lceil log_2 s \rceil + a\,\Big] \geq 0.005$, for $s \geq 1$ and $a = 1, \ldots, 5$.*

PROOF. Let $x \in (0,1)$ such that $\log_2 s + x = \lceil \log_2 s \rceil$. (Recall that the random variables β_i are integer valued.) Then $P[\,\mathrm{Max}_s \beta_i = \log_2 s + x\,] \sim \phi(x) - \phi(x+1)$. This is equal to $-\phi'(\zeta) = \log_2 2\; e^{-2^{1-\zeta}} 2^{1-\zeta}$ for some $\zeta \in (x, x+1) \subseteq (0,2)$. We check immediately that ϕ'' is negative on $(-\infty, 1)$ and positive on $(1, \infty)$. This allows us to write that

$$P\Big[\,\mathrm{Max}_s \beta_i = \lceil \log_2 s \rceil\,\Big] \geq \mathrm{Min}(-\phi'(0), -\phi'(2)) \geq 0.17\ .$$

The same argument gives also that $\forall a \geq 1\ P\Big[\,\mathrm{Max}_s \beta_i = \lceil \log_2 s \rceil + a\,\Big] = -\phi'(\zeta)$ for some $\zeta \in (a, a+2)$. In this interval $-\phi'(\zeta) \geq -\phi'(a+2)$. Hence $\forall a = 1, \ldots, 5$,

$$P\Big[\,\mathrm{Max}_s \beta_i = \lceil \log_2 s \rceil + a\,\Big] \geq -\phi'(7) \sim \log_2 2\, e^{-2^{-7}} 2^{-7} \geq 0.005.$$

□

Chapter 4

Proving Time Bounds for Randomized Distributed Algorithms

4.1 Introduction

This chapter is devoted to the analysis of the timed version of Lehmann-Rabin's Dining Philosophers algorithm [37]. We consider the case where, by assumption, a participating processes cannot wait more then time 1 before taking a step. The scheduling of the processes, i.e., the order under which the various processes take steps, is not in the control of the algorithm. According to our general paradigm (see Chapter 1) we therefore let *Player*(2) decide the schedules. We will prove that Lehmann-Rabin's algorithm verifies a strong correctness property, i.e., a property holding against *Player*(2) *knowing the whole past execution.*

As discussed in page 26 in the section about randomized adversaries, randomization does not make the adversary more powerful and is not needed to establish the correctness of a given algorithm. (We argued nevertheless that considering randomized adversaries was very useful for establishing lower bounds.) We will therefore restrict ourselves in this chapter to the case of *deterministic adversaries.*

Furthermore, following the discussion of page 31, we consider the model where *Player*(2) controls the passage of time. (We showed in page 31 that we could equivalently allocate the time control to *Player*(1).)

We can summarize the previous discussion by saying that the admissible adversaries

are deterministic, know the past execution and do not let a participating process wait more then time 1 for a step. (This does not mean that these properties characterize completely the set of admissible adversaries. We will for instance also require that admissible adversaries let processes exit from their critical section.)

We showed in page 33 that the model for randomized computing presented in [54] was equivalent to the model presented in Definition 2.3.1 in the case where 1) *Player*(1) knows the complete state of the system and remembers it, and 2), deterministic adversaries are considered. We therefore can and will equivalently develop the analysis of [37] in the model of [54].

The original correctness property claimed by Lehmann and Rabin in [37] was that for all admissible adversaries, the probability that the execution is deadlocked is equal to zero. The authors of [37] did not write a formal transcription of this property. In particular they never made explicit what was the event-schema[1] associated to the informal description "the execution is deadlocked". (Note that such a property involves infinitely many random tosses.) They similarly did not provide a formal proof of correctness making explicit the probability spaces $(\Omega_{\mathcal{A}}, \mathcal{G}_{\mathcal{A}}, P_{\mathcal{A}})$ described in our Section 2.4. A more formal proof is therefore needed.

The introduction of time in the proof of correctness presents three advantages. The first one is that it will allow us to work "over a finite horizon of time" instead of the whole infinite execution. This represents a major simplification of the setting within which the proof is conducted. The second advantage is that the timed results are interesting in their own right and provide more insight on the rate at which progress occurs during an execution. (The correctness statement presented in [37] states in essence that progress eventually occurs with probability one.) Last but not least, to establish our result we develop a new general method based on progress functions defined on states, for proving *upper bounds on time* for randomized algorithms. Our method consists of proving auxiliary statements of the form $U \xrightarrow[p]{t} U'$, which means that whenever the algorithm begins in a state in set U, with probability p, it will reach a state in set U' within time t. Of course, this method can only be used for randomized algorithms that include timing assumptions. A key theorem about our method is the composability of these $U \xrightarrow[p]{t} U'$ arrows, as expressed by Theorem 4.3.2. This composability result holds even in the case of (many classes of) non-oblivious adversaries.

We also present two complementary proof rules that help in reasoning about sets of distinct random choices. Independence arguments about such choices are often crucial to correctness proofs, yet there are subtle ways in which a non-oblivious ad-

[1] see page 43 for a definition of an event-schema

versary can introduce dependencies. For example, a non-oblivious adversary has the power to use the outcome of one random choice to decide whether to schedule another random choice. Our proof rules help to systematize certain kinds of reasoning about independence.

As mentioned above, we present our proof in the context of the general framework [54] for describing and reasoning about randomized algorithms. This framework integrates randomness and nondeterminism into one model, and permits the modeling of timed as well as untimed systems. The model of [54] is, in turn, based on existing models for untimed and timed distributed systems [30, 41], and adopts many ideas from the probabilistic models of [58, 27].

Using this general method we are able to prove that $\mathcal{T} \xrightarrow[1/8]{13} \mathcal{C}$, where $\mathcal{T}$ is the set of states in which some process is in its trying region, while $\mathcal{C}$ is the set of states in which some process is in its critical region. That is, whenever the algorithm is in a state in which some process is in the trying region, with probability 1/8, within time 13, it will reach a state in which some process is in its critical region. This bound depends on the timing assumption that processes never wait more then time 1 between steps. A consequence of this claim is an upper bound (of 63) on the expected time for some process to reach its critical region.

For comparison, we already mentioned that [37] contains only proof sketches of the results claimed. The paper [62] contains a proof that Lehmann and Rabin's algorithm satisfies an *eventual* progress condition, in the presence of an adversary with complete knowledge of the past; this proof is carried out as an instance of Zuck and Pnueli's general method for proving liveness properties. Our results about this protocol can be regarded as a refinement of the results of Zuck and Pnueli, in that we obtain explicit constant time bounds rather than liveness properties.

The rest of the paper is organized as follows. Section 4.2 presents a simplified version of the model of [54]. Section 4.3 presents our main proof technique based on time-bound statements. Section 4.4 presents the additional proof rules for independence of distinct probabilistic choices. Section 4.5 presents the Lehmann-Rabin algorithm. Section 4.6.2 formalizes the algorithm in terms of the model of Section 4.2, and gives an overview of our time bound proof. Section 4.7 contains the details of the time bound proof.

Acknowledgments. Sections 4.2, 4.3 and 4.4 were written by Roberto Segala. The references in the subsequent proofs to execution automata, and to the event schemas *Unit-Time*, $\text{FIRST}(a, U)$ and $\text{NEXT}(a, U)$ are also his contribution. (The notation $U \xrightarrow[p]{t} U'$ and Theorem 4.3.2 is part of the work of the author.)

4.2 The Model

In this section, we present the model that is used to formulate our proof technique. It is a simplified version of the probabilistic automaton model of [54]. As mentioned in page 33, this model considers the case where 1) $Player(1)$ knows and remembers the complete state of the system, and 2), deterministic adversaries are considered. Under these conditions it is equivalent to the model presented in Chapter 2. Here we only give the parts of the model that we need to describe our proof method and its application to the Lehmann-Rabin algorithm; we refer the reader to [54] for more details.

Definition 4.2.1 A *probabilistic automaton*[2] M consists of four components:

- a set $states(M)$ of states
- a nonempty set $start(M) \subseteq states(M)$ of start states
- an action signature $sig(M) = (ext(M), int(M))$ where $ext(M)$ and $int(M)$ are disjoint sets of external and internal actions, respectively
- a transition relation $steps(M) \subseteq states(M) \times acts(M) \times Probs(states(states(M)))$, where the set $Probs(states(states(M)))$ is the set of probability spaces $(\Omega, \mathcal{F}, P)$ such that $\Omega \subseteq states(M)$ and $\mathcal{F} = 2^{\Omega}$. The last requirement is needed for technical convenience.

A probabilistic automaton is *fully probabilistic* if it has a unique start state and from each state there is at most one step enabled.

Thus, a probabilistic automaton is a state machine with a labeled transition relation such that the state reached during a step is determined by some probability distribution. For example, the process of flipping a coin is represented by a step labeled with an action `flip` where the next state contains the outcome of the coin flip and is determined by a probability distribution over the two possible outcomes. A probabilistic automaton also allows nondeterministic choices over steps. An example of nondeterminism is the choice of which process takes the next step in a multi-process system.

An *execution fragment* α of a probabilistic automaton M is a (finite or infinite) sequence of alternating states and actions starting with a state and, if the execution

[2] In [54] the probabilistic automata of this definition are called *simple probabilistic automata*. This is because that paper also includes the case of randomized adversaries.

fragment is finite, ending in a state, $\alpha = s_0 a_1 s_1 a_2 s_2 \cdots$, where for each i there exists a probability space $(\Omega, \mathcal{F}, P)$ such that $(s_i, a_{i+1}, (\Omega, \mathcal{F}, P)) \in \mathit{steps}(M)$ and $s_{i+1} \in \Omega$. Denote by $\mathit{fstate}(\alpha)$ the first state of α and, if α is finite, denote by $\mathit{lstate}(\alpha)$ the last state of α. Furthermore, denote by $\mathit{frag}^*(M)$ and $\mathit{frag}(M)$ the sets of finite and all execution fragments of M, respectively. An *execution* is an execution fragment whose first state is a start state. Denote by $\mathit{exec}^*(M)$ and $\mathit{exec}(M)$ the sets of finite and all executions of M, respectively. A state s of M is *reachable* if there exists a finite execution of M that ends in s. Denote by $\mathit{rstates}(M)$ the set of reachable states of M.

A finite execution fragment $\alpha_1 = s_0 a_1 s_1 \cdots a_n s_n$ of M and an execution fragment $\alpha_2 = s_n a_{n+1} s_{n+1} \cdots$ of M can be *concatenated*. In this case the concatenation, written $\alpha_1 \cdot \alpha_2$, is the execution fragment $s_0 a_1 s_1 \cdots a_n s_n a_{n+1} s_{n+1} \cdots$. An execution fragment α_1 of M is a *prefix* of an execution fragment α_2 of M, written $\alpha_1 \leq \alpha_2$, if either $\alpha_1 = \alpha_2$ or α_1 is finite and there exists an execution fragment α_1' of M such that $\alpha_2 = \alpha_1 \cdot \alpha_1'$.

In order to study the probabilistic behavior of a probabilistic automaton, some mechanism to remove nondeterminism is necessary. To give an idea of why the nondeterministic behavior should be removed, consider a probabilistic automaton with three states s_0, s_1, s_2 and with two steps enabled from its start state s_0; the first step moves to state s_1 with probability $1/2$ and to s_2 with probability $1/2$; the second step moves to state s_1 with probability $1/3$ and to s_2 with probability $2/3$. What is the probability of reaching state s_1? The answer depends on how the nondeterminism between the two steps is resolved. If the first step is chosen, then the probability of reaching state s_1 is $1/2$; if the second step is chosen, then the probability of reaching state s_1 is $1/3$. We call the mechanism that removes the nondeterminism an *adversary*, because it is often viewed as trying to thwart the efforts of a system to reach its goals. In distributed systems the adversary is often called the *scheduler*, because its main job may be to decide which process should take the next step.

Definition 4.2.2 An *adversary* for a probabilistic automaton M is a function $\mathcal{A}$ taking a finite execution fragment of M and giving back either nothing (represented as δ) or one of the enabled steps of M if there are any. Denote the set of adversaries for M by $Advs_M$[3].

Once an adversary is chosen, a probabilistic automaton can run under the control of the chosen adversary. The result of the interaction is called an execution automa-

[3] In [54] the adversaries of this definition are denoted by $DAdvs_M$, where D stands for *Deterministic*. The adversaries of [54] are allowed to use randomness.

ton. The definition of an execution automaton, given below, is rather complicated because an execution automaton must contain all the information about the different choices of the adversary, and thus the states of an execution automaton must contain the complete history of a probabilistic automaton. Note that there are no nondeterministic choices left in an execution automaton.

Definition 4.2.3 An *execution automaton* H of a probabilistic automaton M is a fully probabilistic automaton such that

1. $states(H) \subseteq frag^*(M)$.
2. for each step $(\alpha, a, (\Omega, \mathcal{F}, P))$ of H there is a step $(lstate(\alpha), a, (\Omega', \mathcal{F}', P'))$ of M, called the corresponding step, such that $\Omega = \{\alpha a s | s \in \Omega'\}$ and $P'[\alpha a s] = P[s]$ for each $s \in \Omega'$.
3. each state of H is reachable, i.e., for each $\alpha \in states(H)$ there exists an execution of H leading to state α.

Definition 4.2.4 Given a probabilistic automaton M, an adversary $\mathcal{A} \in Advs_M$, and an execution fragment $\alpha \in frag^*(M)$, the execution $H(M, \mathcal{A}, \alpha)$ of M under adversary $\mathcal{A}$ with starting fragment α is the execution automaton of M whose start state is α and such that for each step $(\alpha', a, (\Omega, \mathcal{F}, P)) \in steps(H(M, \mathcal{A}, \alpha))$, its corresponding step is the step $\mathcal{A}(\alpha')$.

Given an execution automaton H, an event is expressed by means of a set of maximal executions of H, where a maximal execution of H is either infinite, or it is finite and its last state does not enable any step in H. For example, the event "eventually action a occurs" is the set of maximal executions of H where action a does occur. A more formal definition follows. The sample space Ω_H is the set of maximal executions of H. The σ-algebra $\mathcal{F}_H$ is the smallest σ-algebra that contains the set of *rectangles* R_α, consisting of the executions of Ω_H having α as a prefix[4]. The probability measure P_H is the unique extension of the probability measure defined on rectangles as follows: $P_H[R_\alpha]$ is the product of the probabilities of each step of H generating α. In [54] it is shown that there is a unique probability measure having the property above, and thus $(\Omega_H, \mathcal{F}_H, P_H)$ is a well defined probability space. For the rest of this abstract we do not need to refer to this formal definition any more.

Events of $\mathcal{F}_H$ are not sufficient for the analysis of a probabilistic automaton. Events are defined over execution automata, but a probabilistic automaton may generate

[4]Note that a rectangle R_α can be used to express the fact that the finite execution α occurs.

several execution automata depending on the adversary it interacts with. Thus a more general notion of event is needed that can deal with all execution automata. Specific examples are given in Section 4.3.

Definition 4.2.5 An event schema e for a probabilistic automaton M is a function associating an event of $\mathcal{F}_H$ with each execution automaton H of M.

We now discuss briefly a simple way to handle time within probabilistic automata. The idea is to add a time component to the states of a probabilistic automaton, to assume that the time at a start state is 0, to add a special non-visible action ν modeling the passage of time, and to add arbitrary time passage steps to each state. A time passage step should be non-probabilistic and should change only the time component of a state. This construction is called the *patient* construction in [44, 57, 22]. The reader interested in a more general extension to timed models is referred to [54].

We close this section with one final definition. Our time bound property for the Lehmann-Rabin algorithm states that if some process is in its trying region, then no matter how the steps of the system are scheduled, some process enters its critical region within time t with probability at least p. However, this claim can only be valid if each process has sufficiently frequent chances to perform a step of its local program. Thus, we need a way to restrict the set of adversaries for a probabilistic automaton. The following definition provides a general way of doing this.

Notation. We let $Advs$ denote a subset of $Advs_M$.

4.3 The Proof Method

In this section, we introduce our key statement $U \xrightarrow[p]{t}_{Advs} U'$ and the *composability theorem*, which is our main theorem about the proof method.

The meaning of the statement $U \xrightarrow[p]{t}_{Advs} U'$ is that, starting from any state of U and under any adversary $\mathcal{A}$ of $Advs$, the probability of reaching a state of U' within time t is at least p. The suffix $Advs$ is omitted whenever we think it is clear from the context.

Definition 4.3.1 Let $e_{U',t}$ be the event schema that, applied to an execution automaton H, returns the set of maximal executions α of H where a state from U' is reached in some state of α within time t. Then $U \xrightarrow[p]{t}_{Advs} U'$ iff for each $s \in U$ and each $\mathcal{A} \in Advs$, $P_{H(M,\mathcal{A},s)}[e_{U',t}(H(M,\mathcal{A},s))] \geq p$.

Proposition 4.3.1 *Let U, U', U'' be sets of states of a probabilistic automaton M. If $U \xrightarrow[p]{t} U'$, then $U \cup U'' \xrightarrow[p]{t} U' \cup U''$.*

In order to compose time bound statements, we need a restriction for adversary schemas stating that the power of the adversary schema is not reduced if a prefix of the past history of the execution is not known. Most adversary schemas that appear in the literature satisfy this restriction.

Definition 4.3.2 An adversary schema $Advs$ for a probabilistic automaton M is *execution closed* if, for each $\mathcal{A} \in Advs$ and each finite execution fragment $\alpha \in frag^*(M)$, there exists an adversary $\mathcal{A}' \in Advs$ such that for each execution fragment $\alpha' \in frag^*(M)$ with $lstate(\alpha) = fstate(\alpha')$, $\mathcal{A}'(\alpha') = \mathcal{A}(\alpha \cdot \alpha')$.

Theorem 4.3.2 *Let $Advs$ be an execution closed adversary schema for a probabilistic timed automaton M, and let U, U', U'' be sets of states of M. If $U \xrightarrow[p_1]{t_1}_{Advs} U'$ and $U' \xrightarrow[p_2]{t_2}_{Advs} U''$, then $U \xrightarrow[p_1 p_2]{t_1+t_2}_{Advs} U''$.*

Sketch of proof: Consider an adversary $\mathcal{A} \in Advs$ that acts on M starting from a state s of U. The execution automaton $H(M, \mathcal{A}, s)$ contains executions where a state from U' is reached within time t_1. Consider one of those executions α and consider the part H of $H(M, \mathcal{A}, s)$ after the first occurrence of a state from U' in α. The key idea of the proof is to use execution closure of $Advs$ to show that there is an adversary that generates H, to use $U' \xrightarrow[p_2]{t_2}_{Advs} U''$ to show that in H a state from U'' is reached within time t_2 with probability at least p_2, and to integrate this last result in the computation of the probability of reaching a state from U'' in $H(M, \mathcal{A}, s)$ within time $t_1 + t_2$. ■

4.4 Independence

Example 4.4.1 Consider any distributed algorithm where each process is allowed to flip fair coins. It is common to say "If the next coin flip of process P yields *head* and the next coin flip of process Q yields *tail*, then some good property ϕ holds." Can we conclude that the probability for ϕ to hold is 1/4? That is, can we assume that the coin flips of processes P and Q are independent? The two coin flips are indeed independent of each other, but the presence of non-oblivious adversaries may introduce some dependence. An adversary can schedule process P to flip its coin and then schedule process Q only if the coin flip of process P yielded *head*. As a

result, if both P and Q flip a coin, the probability that P yields *head* and Q yields *tail* is $1/2$.

Thus, it is necessary to be extremely careful about independence assumptions. It is also important to pay attention to potential ambiguities of informal arguments. For example, does ϕ hold if process P flips a coin yielding *head* and process Q does not flip any coin? Certainly such an ambiguity can be avoided by expressing each event in a formal model.

In this section we present two event schemas that play a key role in the detailed time bound proof for the Lehmann-Rabin algorithm, and we show some partial independence properties for them. The first event schema is a generalization of the informal statement of Example 4.4.1, where a coin flip is replaced by a generic action a, and where it is assumed that an event contains all the executions where a is not scheduled; the second event schema is used to analyze the outcome of the first random draw that occurs among a fixed set of random draws. A consequence of the partial independence results that we show below is that under any adversary the property ϕ of Example 4.4.1 holds with probability at least $1/4$.

Let (a, U) be a pair consisting of an action of M and a set of states of M. The event schema $\text{FIRST}(a, U)$ is the function that, given an execution automaton H, returns the set of maximal executions of H where either action a does not occur, or action a occurs and the state reached after the first occurrence of a is a state of U. This event schema is used to express properties like "the i^{th} coin yields `left`". For example a can be `flip` and U can be the set of states of M where the result of the coin flip is `left`.

Let $(a_1, U_1), \ldots, (a_n, U_n)$ be a sequence of pairs consisting of an action of M and a set of states of M such that for each i, j, $1 \leq i < j \leq n$, $a_i \neq a_j$. Define the event schema $\text{NEXT}((a_1, U_1), \ldots, (a_n, U_n))$ to be the function that applied to an execution automaton H gives the set of maximal executions of H where either no action from $\{a_1, \ldots, a_n\}$ occurs, or at least one action from $\{a_1, \ldots, a_n\}$ occurs and, if a_i is the first action that occurs, the state reached after the first occurrence of a_i is in U_i. This kind of event schema is used to express properties like "the first coin that is flipped yields `left`."

Proposition 4.4.2 *Let H be an execution automaton of a probabilistic automaton M. Furthermore, let $(a_1, U_1), \ldots, (a_n, U_n)$ be pairs consisting of an action of M and a set of states of M such that for each i, j, $1 \leq i < j \leq n$, $a_i \neq a_j$. Finally, let $p_1, \ldots, p_n$ be real numbers between 0 and 1 such that for each i, $1 \leq i \leq n$, and each step $(s, a, (\Omega, \mathcal{F}, P)) \in steps(M)$ with $a = a_i$, the probability $P[U_i \cap \Omega]$ is greater than or equal to p_i, i.e., $P[U_i \cap \Omega] \geq p_i$. Then*

1. $P_H[(\text{FIRST}(a_1, U_1) \cap \cdots \cap \text{FIRST}(a_n, U_n))(H)] \geq p_1 \cdots p_n$,
2. $P_H[\text{NEXT}((a_1, U_1), \ldots, (a_n, U_n))(H)] \geq min(p_1, \ldots, p_n)$.

4.5 The Lehmann-Rabin Algorithm

The Lehmann-Rabin algorithm is a randomized algorithm for the Dining Philosophers problem. This problem involves the allocation of n resources among n competing processes arranged in a ring. The resources are considered to be interspersed between the processes, and each process requires both its adjacent resources in order to reach its critical section. All processes are identical; the algorithm breaks symmetry by using randomization. The algorithm ensures the required exclusive possession of resources, and also ensures that, with probability 1, some process is always permitted to make progress into its critical region.

Figure 4.1 shows the code for a generic process i. The n resources are represented by n shared variables $\text{Res}_1, \ldots, \text{Res}_n$, each of which can assume values in $\{\texttt{free}, \texttt{taken}\}$. Each process i ignores its own name, i, and the names, Res_{i-1} and Res_i, of its adjacent resources. However, each process i is able to refer to its adjacent resources by relative names: $\text{Res}_{(i,\texttt{left})}$ is the resource located to the left (clockwise), and $\text{Res}_{(i,\texttt{right})}$ is the resource to the right (counterclockwise) of i. Each process has a private variable u_i, which can assume a value in $\{\texttt{left}, \texttt{right}\}$, and is used to keep track of the first resource to be handled. For notational convenience we define an operator *opp* that complements the value of its argument, i.e., $opp(\texttt{right}) = \texttt{left}$ and $opp(\texttt{left}) = \texttt{right}$.

The atomic actions of the code are individual resource accesses, and they are represented in the form <*atomic-action*> in Figure 4.1. We assume that at most one process has access to the shared resource at each time.

An informal description of the procedure is "choose a side randomly in each iteration. Wait for the resource on the chosen side, and, after getting it, just check *once* for the second resource. If this check succeeds, then proceed to the critical region. Otherwise, put down the first resource and try again with a new random choice."

Each process exchanges messages with an external user. In its idle state, a process is in its remainder region R. When triggered by a `try` message from the user, it enters the competition to get its resources: we say that it enters its trying region T. When the resources are obtained, it sends a `crit` message informing the user of the possession of these resources: we then say that the process is in its critical region C. When triggered by an `exit` message from the user, it begins relinquishing its

Shared variables: $\mathrm{Res}_j \in \{\texttt{free}, \texttt{taken}\}$, $j = 1, \ldots, n$, initially `free`.

Local variables: $u_i \in \{\texttt{left}, \texttt{right}\}$, $i = 1, \ldots, n$

Code for process i:

```
0.   try                                          ** beginning of Trying Section **
1.   < u_i ← random>  ** choose left or right with equal probability **
2.   < if Res_(i,u_i) = free then
           Res_(i,u_i) := taken                   ** pick up first resource **
       else goto 2. >
3.    < if Res_(i,opp(u_i)) = free then
           Res_(i,opp(u_i)) := taken;             ** pick up second resource **
           goto 5. >
4.   < Res_(i,u_i) := free; goto 1.>              ** put down first resource **
5.   crit                                         ** end of Trying Section **
     ** Critical Section **
6.   exit                                         ** beginning of Exit Section **
7.   < u_i ← left or right                        ** nondeterministic choice **
       Res_(i,opp(u_i)) := free >                 ** put down first resources **
8.   < Res_(i,u_i) := free >                      ** put down second resources **
9.   rem                                          ** end of Exit Section **
     ** Remainder Section **
```

Figure 4.1: The Lehmann-Rabin algorithm

resources: we then say that the process is in its exit region E. When the resources are relinquished its sends a `rem` message to the user and enters its remainder region.

4.6 Overview of the Proof

In this section, we give our high-level overview of the proof. We first introduce some notation, then sketch the proof strategy at a high level. The detailed proof is presented in Section 4.7.

4.6.1 Notation

In this section we define a probabilistic automaton M which describes the system of Section 4.5. We assume that process $i+1$ is on the right of process i and that resource Res_i is between processes i and $i+1$. We also identify labels modulo n so that, for instance, process $n+1$ coincides with process 1.

A state s of M is a tuple $(X_1, \ldots, X_n, \mathrm{Res}_1, \ldots, \mathrm{Res}_n, t)$ containing the local state X_i of each process i, the value of each resource Res_i, and the current time t. Each local state X_i is a pair (pc_i, u_i) consisting of a program counter pc_i and the local variable u_i. The program counter of each process keeps track of the current instruction in the code of Figure 4.1. Rather then representing the value of the program counter with a number, we use a more suggestive notation which is explained in the table below. Also, the execution of each instruction is represented by an action. Only actions $\texttt{try}_i$, $\texttt{crit}_i$, $\texttt{rem}_i$, $\texttt{exit}_i$ below are external actions.

Number	pc_i	Action name	Informal meaning
0	R	$\texttt{try}_i$	**R**emainder region
1	F	$\texttt{flip}_i$	Ready to **F**lip
2	W	$\texttt{wait}_i$	**W**aiting for first resource
3	S	$\texttt{second}_i$	Checking for **S**econd resource
4	D	$\texttt{drop}_i$	**D**ropping first resource
5	P	$\texttt{crit}_i$	**P**re-critical region
6	C	$\texttt{exit}_i$	**C**ritical region
7	E_F	$\texttt{dropf}_i$	**E**xit: drop **F**irst resource
8	E_S	$\texttt{drops}_i$	**E**xit: drop **S**econd resource
9	E_R	$\texttt{rem}_i$	**E**xit: move to **R**emainder region

The start state of M assigns the value `free` to all the shared variables Res_i, the value R to each program counter pc_i, and an arbitrary value to each variable u_i. The transition relation of M is derived directly from Figure 4.1. For example, for each state where $pc_i = F$ there is an internal step $\texttt{flip}_i$ that changes pc_i into W and assigns `left` to u_i with probability $1/2$ and `right` to u_i with probability $1/2$; from each state where $X_i = (W, \texttt{left})$ there is a step $\texttt{wait}_i$ that does not change the state if $\mathrm{Res}_{(i,\texttt{left})} = \texttt{taken}$, and changes pc_i into S and $\mathrm{Res}_{(i,\texttt{left})}$ into `taken` if $\mathrm{Res}_{(i,\texttt{left})} = \texttt{free}$; for each state where $pc_i = E_F$ there are two steps with action $\texttt{dropf}_i$: one step sets u_i to `right` and makes $\mathrm{Res}_{(i,\texttt{left})}$ free, and the other step sets u_i to `left` makes $\mathrm{Res}_{(i,\texttt{right})}$ free. The two separate steps correspond to a nondeterministic choice that is left to the adversary. For time passage steps we assume that at any point an arbitrary amount of time can pass; thus, from each

state of M and each positive δ there is a time passage step that increases the time component of δ and does not affect the rest of the state.

The value of each pair X_i can be represented concisely by the value of pc_i and an arrow (to the left or to the right) which describes the value of u_i. Thus, informally, a process i is in state $\underrightarrow{S}$ or $\underrightarrow{D}$ (resp. $\underleftarrow{S}$ or $\underleftarrow{D}$) when i is in state S or D while holding its right (resp. left) resource; process i is in state $\underrightarrow{W}$ (resp. $\underleftarrow{W}$) when i is waiting for its right (resp. left) resource to become free; process i is in state $\underrightarrow{E_S}$ (resp. $\underleftarrow{E_S}$) when i is in its exit region and it is still holding its right (resp. left) resource. Sometimes we are interested in sets of pairs; for example, whenever $pc_i = F$ the value of u_i is irrelevant. With the simple value of pc_i we denote the set of the two pairs $\{(pc_i, \texttt{left}), (pc_i, \texttt{right})\}$. Finally, with the symbol $\#$ we denote any pair where $pc_i \in \{W, S, D\}$. The arrow notation is used as before.

For each state $s = (X_0, \ldots, X_{n-1}, \mathrm{Res}_1, \ldots, \mathrm{Res}_{n-1}, t)$ of M we denote by $X_i(s)$ the pair X_i and by $\mathrm{Res}_i(s)$ the value of the shared variable Res_i in state s. Also, for any set S of states of a process i, we denote by $X_i \in S$, or alternatively $X_i = S$ the set of states s of M such that $X_i(s) \in S$. Sometimes we abuse notation in the sense that we write expressions like $X_i \in \{F, D\}$ with the meaning $X_i \in F \cup D$. Finally, we write $X_i = E$ for $X_i = \{E_F, E_S, E_R\}$, and we write $X_i = T$ for $X_i \in \{F, W, S, D, P\}$.

A first basic lemma states that a reachable state of M is uniquely determined by the local states its processes and the current time. Based on this lemma, our further specifications of state sets will not refer to the shared variables; however, we consider only reachable states for the analysis. The proof of the lemma is a standard proof of invariants.

Lemma 4.6.1 *For each reachable state s of M and each i, $1 \leq i \leq n$, $Res_i = \texttt{taken}$ iff $X_i(s) \in \{\underrightarrow{S}, \underrightarrow{D}, P, C, E_F, \underrightarrow{E_S}\}$ or $X_{i+1}(s) \in \{\underleftarrow{S}, \underleftarrow{D}, P, C, E_F, \underleftarrow{E_S}\}$. Moreover, for each reachable state s of M and each i, $1 \leq i \leq n$, it is not the case that $X_i(s) \in \{\underrightarrow{S}, \underrightarrow{D}, P, C, E_F, \underrightarrow{E_S}\}$ and $X_{i+1}(s) \in \{\underleftarrow{S}, \underleftarrow{D}, P, C, E_F, \underleftarrow{E_S}\}$, i.e., only one process at a time can hold one resource.* □

4.6.2 Proof Sketch

In this section we show that the RL-algorithm guarantees time bounded progress, i.e., that from every state where some process is in its trying region, some process subsequently enters its critical region within an expected constant time bound. We assume that each process that is ready to perform a step does so within time 1: process i is ready to perform a step whenever it enables an action different from

$\texttt{try}_i$ or $\texttt{exit}_i$. Actions $\texttt{try}_i$ and $\texttt{exit}_i$ are supposed to be under the control of the user, and hence, by assumption, under the control of the adversary.

Formally, consider the probabilistic timed automaton M of Section 4.6.1. Define *Unit-Time* to be the set of adversaries $\mathcal{A}$ for M having the properties that, for every finite execution fragment α of M and every execution α' of $H(M, \mathcal{A}, \alpha)$, 1) the time in α' is not bounded and 2) for every process i and every state of α' enabling an action of process i different from $\texttt{try}_i$ and $\texttt{exit}_i$, there exists a step in α' involving process i within time 1. Then *Unit-Time* is execution-closed according to Definition 4.3.2. An informal justification of this fact is that the constraint that each ready process is scheduled within time 1 knowing that $\alpha \cdot \alpha'$ has occurred only reinforces the constraint that each ready process is scheduled within time 1 knowing that α' has occurred. Let

$$\mathcal{T} \triangleq \{s \in \mathit{rstates}(M) \mid \exists_i X_i(s) \in \{T\}\}$$

denote the sets of reachable states of M where some process is in its trying region, and let

$$\mathcal{C} \triangleq \{s \in \mathit{rstates}(M) \mid \exists_i X_i(s) = C\}$$

denote the sets of reachable states of M where some process is in its critical region. We show that

$$\mathcal{T} \xrightarrow[1/8]{13} \mathit{Unit\text{-}Time}\, \mathcal{C},$$

i.e., that, starting from any reachable state where some process is in its trying region, for all the adversaries of *Unit-Time*, with probability at least $1/8$, some process enters its critical region within time 13. Note that this property is trivially satisfied if some process is initially in its critical region.

Our proof is divided into several phases, each one concerned with the property of making a partial time bounded progress toward a "success state", i.e., a state of $\mathcal{C}$. The sets of states associated with the different phases are expressed in terms of $\mathcal{T}, \mathcal{RT}, \mathcal{F}, \mathcal{G}, \mathcal{P}$, and $\mathcal{C}$. Here,

$$\mathcal{RT} \triangleq \{s \in \mathcal{T} \mid \forall_i X_i(s) \in \{E_R, R, T\}\}$$

is the set of states where at least one process is in its trying region and where no process is in its critical region or holds resources while being in its exit region.

$$\mathcal{F} \triangleq \{s \in \mathcal{RT} \mid \exists_i X_i(s) = F\}$$

is the set of states of $\mathcal{RT}$ where some process is ready to flip a coin.

$$\mathcal{P} \triangleq \{s \in \mathit{rstates}(M) \mid \exists_i X_i(s) = P\}$$

is the sets of reachable states of M where some process is in its pre-critical region. The set $\mathcal{G}$ is the most important for the analysis. It parallels the set of "Good Pairs"in [62] or the set described in Lemma 4 of [37]. To motivate the definition, we define the following notions. We say that a process i is *committed* if $X_i \in \{W, S\}$, and that a process i *potentially controls* Res_i (resp. Res_{i-1}) if $X_i \in \{\underline{W}_{\rightarrow}, \underline{S}_{\rightarrow}, \underline{D}_{\rightarrow}\}$ (resp. $X_i \in \{\underline{W}_{\leftarrow}, \underline{S}_{\leftarrow}, \underline{D}_{\leftarrow}\}$). Informally said, a state in $\mathcal{RT}$ is in $\mathcal{G}$ if and only if there is a committed process whose second resource is not potentially controlled by another process. Such a process is called a *good* process. Formally,

$$\begin{aligned}\mathcal{G} \triangleq \{s \in \mathcal{RT} \mid \exists_i \; & X_i(s) \in \{\underline{W}_{\leftarrow}, \underline{S}_{\leftarrow}\} \text{ and } X_{i+1}(s) \in \{E_R, R, F, \underline{\#}_{\rightarrow}\}, \text{ or} \\ & X_i(s) \in \{\underline{W}_{\rightarrow}, \underline{S}_{\rightarrow}\} \text{ and } X_{i-1}(s) \in \{E_R, R, F, \underline{\#}_{\leftarrow}\}\}\end{aligned}$$

Reaching a state of $\mathcal{G}$ is a substantial progress toward reaching a state of $\mathcal{C}$. Actually, the proof of Proposition 4.7.11 establishes that, if i a is good process, then, with probability $1/4$, one of the three processes $i-1, i$ and $i+1$ soon succeeds in getting its two resources. The hard part is to establish that, with constant probability, within a *constant* time, $\mathcal{G}$ is reached from any state in $\mathcal{T}$. A close inspection of the proof given in [62] shows that, there, the timed version of the techniques used is unable to deliver this result. The phases of our proof are formally described below.

$$\begin{aligned}
&\mathcal{T} \xrightarrow{2} \mathcal{RT} \cup \mathcal{C} && \text{(Proposition 4.7.3)},\\
&\mathcal{RT} \xrightarrow{3} \mathcal{F} \cup \mathcal{P} && \text{(Proposition 4.7.15)},\\
&\mathcal{F} \xrightarrow[1/2]{2} \mathcal{G} \cup \mathcal{P} && \text{(Proposition 4.7.14)},\\
&\mathcal{G} \xrightarrow[1/4]{5} \mathcal{P} && \text{(Proposition 4.7.11)},\\
&\mathcal{P} \xrightarrow{1} \mathcal{C} && \text{(Proposition 4.7.1)}.
\end{aligned}$$

The first statement states that, within time 2, every process in its exit region relinquishes its resources. By combining the statements above by means of Proposition 4.3.1 and Theorem 4.3.2 we obtain

$$\mathcal{T} \xrightarrow[1/8]{13} \mathcal{C},$$

which is the property that was to be proven. Using the results of the proof summary above, we can furthermore derive a constant upper bound on the expected time required to reach a state of $\mathcal{C}$ when departing from a state of $\mathcal{T}$. Note that, departing from a state in $\mathcal{RT}$, with probability at least $1/8$, $\mathcal{P}$ is reached in time (at most) 10; with probability at most $1/2$, time 5 is spent before failing to reach $\mathcal{G} \cup \mathcal{P}$ ("failure at the third arrow"); with probability at most $7/8$, time 10 is spent before failing

to reach $\mathcal{P}$ ("failure at the fourth arrow"). If failure occurs, then the state is back into $\mathcal{RT}$. Let V denote a random variable satisfying the following induction

$$V = 1/8 \cdot 10 + 1/2\,(5 + V_1) + 3/8\,(10 + V_2)\ ,$$

where V_1 and V_2 are random variables having the same distribution as V. The previous discussion shows that the expected time spent from $\mathcal{RT}$ to $\mathcal{P}$ is at most $E[V]$. By taking expectation in the previous equation, and using that $E[V] = E[V_1] = E[V_2]$, we obtain that $E[V] = 60$ is an upper bound on the expected time spent from $\mathcal{RT}$ to $\mathcal{P}$, and that, consequently, the expected time for progress starting from a state of $\mathcal{T}$ is at most 63.

4.7 The Detailed Proof

We prove in this section the five relations used in Section 4.6.2. However, for the sake of clarity, we do not prove the relations in the order they were presented. Throughout the proof we abuse notation by writing events of the kind FIRST($\texttt{flip}_i, \texttt{left}$) meaning the event schema FIRST($\texttt{flip}_i, \{s \in \mathit{states}(M) \mid X_i(s) = \underset{\leftarrow}{W}\}$).

Proposition 4.7.1 *If some process is in P, then, within time 1, it enters C, i.e.,*

$$\mathcal{P} \xrightarrow[1]{1} \mathcal{C}.$$

PROOF. This step corresponds to the action `crit`: within time 1, process i informs the user that the critical region is free. □

Lemma 4.7.2 *If some process is in its Exit region then, within time 3, it will enter R.*

PROOF. The process needs to take first two steps to relinquish its two resources, and then one step to send a `rem` message to the user. □

Proposition 4.7.3 $\mathcal{T} \xrightarrow{2} \mathcal{RT} \cup \mathcal{C}$.

PROOF. From Lemma 4.7.2 within time 2 every process that begins in E_F or E_S relinquishes its resources. If no process begins in C or enters C in the meantime, then the state reached at this point is a state of $\mathcal{RT}$; otherwise, the starting state or the state reached when the first process enters C is a state of $\mathcal{C}$. □

We now turn to the proof of $\mathcal{G} \xrightarrow[1/4]{5} \mathcal{P}$. The following lemmas form a detailed cases analysis of the different situations that can arise in states of $\mathcal{G}$. Informally, each lemma shows that some event of the form of Proposition 4.4.2 is a sub-event of the properties of reaching some other state.

Lemma 4.7.4

1. *Assume that* $X_{i-1} \in \{E_R, R, F\}$ *and* $X_i = \underleftarrow{W}$. *If* FIRST($\texttt{flip}_{i-1}$, `left`), *then, within time* 1, *either* $X_{i-1} = P$ *or* $X_i = S$.

2. *Assume that* $X_{i-1} = D$ *and* $X_i = \underleftarrow{W}$. *If* FIRST($\texttt{flip}_{i-1}$, `left`), *then, within time* 2, *either* $X_{i-1} = P$ *or* $X_i = S$.

3. *Assume that* $X_{i-1} = S$ *and* $X_i = \underleftarrow{W}$. *If* FIRST($\texttt{flip}_{i-1}$, `left`), *then, within time* 3, *either* $X_{i-1} = P$ *or* $X_i = S$.

4. *Assume that* $X_{i-1} = W$ *and* $X_i = \underleftarrow{W}$. *If* FIRST($\texttt{flip}_{i-1}$, `left`), *then, within time* 4, *either* $X_{i-1} = P$ *or* $X_i = S$.

Proof. The four proofs start in the same way. Let s be a state of M satisfying the respective properties of items *1* or *2* or *3* or *4*. Let $\mathcal{A}$ be an adversary of *Unit-Time*, and let α be the execution of M that corresponds to an execution of $H(M, \mathcal{A}, \{s\})$ where the result of the first coin flip of process $i-1$ is `left`.

1. By hypothesis, $i-1$ does not hold any resource at the beginning of α and has to obtain Res_{i-2} (its left resource) before pursuing Res_{i-1}. Within time 1, i takes a step in α. If $i-1$ does not hold Res_{i-1} when i takes this step, then i progresses into configuration S. If not, it must be the case that $i-1$ succeeded in getting it in the meanwhile. But, in this case, Res_{i-1} was the second resource needed by $i-1$ and $i-1$ therefore entered P.

2. If $X_i = S$ within time 1, then we are done. Otherwise, after one unit of time, X_i is still equal to $\underleftarrow{W}$, i.e., $X_i(s') = \underleftarrow{W}$ for all states s' reached in time 1. However, process $i-1$ takes also a step within time 1. Let $\alpha = \alpha_1 \cdot \alpha_2$ such that the last action of α_1 corresponds to the first step taken by process $i-1$. Then $X_{i-1}(\mathit{fstate}(\alpha_2)) = F$ and $X_i(\mathit{fstate}(\alpha_2)) = \underleftarrow{W}$. Since process $i-1$ did not flip any coin during α_1, from the execution closure of *Unit-Time* and item *1* we conclude.

3. If $X_i = S$ within time 1, then we are done. Otherwise, after one unit of time, X_i is still equal to $\underset{\leftarrow}{W}$, i.e., $X_i(s') = \underset{\leftarrow}{W}$ for all states s' reached in time 1. However, also process $i-1$ takes a step within time 1. Let $\alpha = \alpha_1 \cdot \alpha_2$ such that the last action of α_1 corresponds to the first step taken by process $i-1$. If $X_{i-1}(fstate(\alpha_2)) = P$ then we are also done. Otherwise it must be the case that $X_{i-1}(fstate(\alpha_2)) = D$ and $X_i(fstate(\alpha_2)) = \underset{\leftarrow}{W}$. Since process $i-1$ did not flip any coin during α_1, from the execution closure of *Unit-Time* and item *2* we conclude.

4. If $X_i = S$ within time 1, then we are done. Otherwise, after one unit of time, X_i is still equal to $\underset{\leftarrow}{W}$, i.e., $X_i(s') = \underset{\leftarrow}{W}$ for all states s' reached in time 1. However, since within time 1 process i checks its left resource and fails, process $i-1$ gets its right resource within time 1, and hence reaches at least state S. Let $\alpha = \alpha_1 \cdot \alpha_2$ where the last step of α_1 is the first step of α leading process $i-1$ to state S. Then $X_{i-1}(fstate(\alpha_2)) = S$ and $X_i(fstate(\alpha_2)) = \underset{\leftarrow}{W}$. Since process $i-1$ did not flip any coin during α_1, from the execution closure of *Unit-Time* and item *3* we conclude. □

Lemma 4.7.5 *Assume that* $X_{i-1} \in \{E_R, R, T\}$ *and* $X_i = \underset{\leftarrow}{W}$. *If* FIRST($\texttt{flip}_{i-1}$, `left`), *then, within time* 4, *either* $X_{i-1} = P$ *or* $X_i = S$.

PROOF. The lemma follows immediately from Lemma 4.7.4 after observing that $X_{i-1} \in \{E_R, R, T\}$ means $X_{i-1} \in \{E_R, R, F, W, S, D, P\}$. □

The next lemma is a useful tool for the proofs of Lemmas 4.7.7, 4.7.8, and 4.7.9.

Lemma 4.7.6 *Assume that* $X_i \in \{\underset{\leftarrow}{W}, \underset{\leftarrow}{S}\}$ *or* $X_i \in \{E_R, R, F, \underset{\leftarrow}{D}\}$ *with* FIRST($\texttt{flip}_i$, `left`), *and assume that* $X_{i+1} \in \{\underset{\rightarrow}{W}, \underset{\rightarrow}{S}\}$ *or* $X_{i+1} \in \{E_R, R, F, \underset{\rightarrow}{D}\}$ *with* FIRST($\texttt{flip}_{i+1}$, `right`). *Then the first of the two processes* i *or* $i+1$ *testing its second resource enters* P *after having performed this test (if this time ever comes).*

PROOF. By Lemma 4.6.1 Res_i is free. Moreover, Res_i is the second resource needed by both i and $i+1$. Whichever tests for it first gets it and enters P. □

Lemma 4.7.7 *If* $X_i = \underset{\leftarrow}{S}$ *and* $X_{i+1} \in \{\underset{\rightarrow}{W}, \underset{\rightarrow}{S}\}$ *then, within time* 1, *one of the two processes* i *or* $i+1$ *enters* P. *The same result holds if* $X_i \in \{\underset{\leftarrow}{W}, \underset{\leftarrow}{S}\}$ *and* $X_{i+1} = \underset{\rightarrow}{S}$.

PROOF. Being in state S, process i tests its second resource within time 1. An application of Lemma 4.7.6 finishes the proof. □

Lemma 4.7.8 *Assume that* $X_i = \underleftarrow{S}$ *and* $X_{i+1} \in \{E_R, R, F, \underrightarrow{D}\}$. *If* FIRST($\texttt{flip}_{i+1}$, `right`), *then, within time* 1, *one of the two processes* i *or* $i+1$ *enters* P. *The same result holds if* $X_i \in \{E_R, R, F, D\}$, $X_{i+1} = \underrightarrow{S}$ *and* FIRST($\texttt{flip}_i$, `left`).

PROOF. Being in state S, process i tests its second resource within time 1. An application of Lemma 4.7.6 finishes the proof. □

Lemma 4.7.9 *Assume that* $X_{i-1} \in \{E_R, R, T\}$, $X_i = \underleftarrow{W}$, *and* $X_{i+1} \in \{E_R, R, F, \underrightarrow{W}, \underrightarrow{D}\}$. *If* FIRST($\texttt{flip}_{i-1}$, `left`) *and* FIRST($\texttt{flip}_{i+1}$, `right`), *then within time* 5 *one of the three processes* $i-1$, i *or* $i+1$ *enters* P.

PROOF. Let s be a state of M such that $X_{i-1}(s) \in \{E_R, R, T\}$, $X_i(s) = \underleftarrow{W}$, and $X_{i+1}(s) \in \{E_R, R, F, \underrightarrow{W}, \underrightarrow{D}\}$. Let $\mathcal{A}$ be an adversary of *Unit-Time*, and let α be the execution of M that corresponds to an execution of $H(M, \mathcal{A}, \{s\})$ where the result of the first coin flip of process $i-1$ is `left` and the result of the first coin flip of process $i+1$ is `right`. By Lemma 4.7.5, within time 4 either process $i-1$ reaches configuration P in α or process i reaches configuration $\underleftarrow{S}$ in α. If $i-1$ reaches configuration P, then we are done. If not, then let $\alpha = \alpha_1 \cdot \alpha_2$ such that $lstate(\alpha_1)$ is the first state s' of α with $X_i(s') = \underleftarrow{S}$. If $i+1$ enters P before the end of α_1, then we are done. Otherwise, $X_{i+1}(fstate(\alpha_2))$ is either in $\{\underrightarrow{W}, \underrightarrow{S}\}$ or it is in $\{E_R, R, F, \underrightarrow{D}\}$ and process $i+1$ has not flipped any coin yet in α. From execution closure of *Unit-Time* we can then apply Lemma 4.7.6. Within one more time process i tests its second resource and enters P if process $i+1$ did not check its second resource in the meantime. On the other hand, process $i+1$ enters P if it checks its second resource before i does so. □

Lemma 4.7.10 *Assume that* $X_{i-1} \in \{E_R, R, F, \underleftarrow{W}, \underleftarrow{D}\}$, $X_i = \underrightarrow{W}$, *and* $X_{i+1} \in \{E_R, R, T\}$. *If* FIRST($\texttt{flip}_{i-1}$, `left`) *and* FIRST($\texttt{flip}_{i+1}$, `right`), *then within time* 5 *one of the three processes* $i-1$, i *or* $i+1$, *enters* P.

PROOF. Analogous to Lemma 4.7.9. □

Proposition 4.7.11 *Starting from a global configuration in* $\mathcal{G}$, *then, with probability at least* $1/4$ *and within time at most* 5, *some process enters* P. *Equivalently:*

$$\mathcal{G} \xrightarrow[1/4]{5} \mathcal{P}.$$

PROOF. Lemmas 4.7.7 and 4.7.8 jointly treat the case where $X_i = \underleftarrow{S}$ and $X_{i+1} \in \{E_R, R, F, \underrightarrow{\#}\}$ and the symmetric case where $X_{i-1} \in \{E_R, R, F, \underleftarrow{\#}\}$ and $X_i = \underrightarrow{S}$; Lemmas 4.7.9 and 4.7.10 jointly treat the case where $X_i = \underleftarrow{W}$ and $X_{i+1} \in \{E_R, R, F, \underrightarrow{W}, \underrightarrow{D}\}$ and the symmetric case where $X_{i-1} \in \{E_R, R, F, \underleftarrow{W}, \underleftarrow{D}\}$ and $X_i = \underrightarrow{W}$.

Specifically, each lemma shows that a compound event of the kind FIRST($\texttt{flip}_i, x$) and FIRST($\texttt{flip}_j, y$) leads to $\mathcal{P}$. Each of the basic events FIRST($\texttt{flip}_i, x$) has probability 1/2. From Proposition 4.4.2 each of the compound events has probability at least 1/4. Thus the probability of reaching $\mathcal{P}$ within time 5 is at least 1/4. □

We now turn to $\mathcal{F} \xrightarrow[1/2]{2} \mathcal{G} \cup \mathcal{P}$. The proof is divided in two parts and constitute the global argument of the proof of progress.

Lemma 4.7.12 *Start with a state s of $\mathcal{F}$. If there exists a process i for which $X_i(s) = F$ and $(X_{i-1}, X_{i+1}) \neq (\underrightarrow{\#}, \underleftarrow{\#})$, then, with probability at least 1/2 a state of $\mathcal{G} \cup \mathcal{P}$ is reached within time 1.*

PROOF. The conclusion holds trivially if $s \in \mathcal{G}$. Let s be a state of $\mathcal{F} - \mathcal{G}$ and let i be such that $X_i(s) = F$ and $(X_{i-1}, X_{i+1}) \neq (\underrightarrow{\#}, \underleftarrow{\#})$. Assume without loss of generality that $X_{i+1} \neq \underleftarrow{\#}$, i.e., $X_{i+1} \in \{E_R, R, F, \underrightarrow{\#}\}$. (The case for $X_{i-1} \neq \underrightarrow{\#}$ is similar.) We can furthermore assume that $X_{i+1} \in \{E_R, R, F, \underrightarrow{D}\}$ since if $X_{i+1} \in \{\underrightarrow{W}, \underrightarrow{S}\}$ then s is already in $\mathcal{G}$.

We show that the event NEXT($(\texttt{flip}_i, \texttt{left}), (\texttt{flip}_{i+1}, \texttt{right})$), which by Proposition 4.4.2 has probability at least 1/2, leads in time at most 1 to a state of $\mathcal{G} \cup \mathcal{P}$. Let $\mathcal{A}$ be an adversary of *Unit-Time*, and let α be the execution of M that corresponds to an execution of $H(M, \mathcal{A}, \{s\})$ where if process i flips before process $i+1$ then process i flips left, and if process $i+1$ flips before process i then process $i+1$ flips right.

Within time 1, i takes one step and reaches W. Let $j \in \{i, i+1\}$ be the first of i and $i+1$ that reaches W and let s_1 be the state reached after the first time process j reaches W. If some process reached P in the meantime, then we are done. Otherwise there are two cases to consider. If $j = i$, then, $\texttt{flip}_i$ gives $\texttt{left}$ and $X_i(s_1) = \underleftarrow{W}$ whereas X_{i+1} is (still) in $\{E_R, R, F, \underrightarrow{D}\}$. Therefore, $s_1 \in \mathcal{G}$. If $j = i+1$, then $\texttt{flip}_{i+1}$ gives $\texttt{right}$ and $X_{i+1}(s_1) = \underrightarrow{W}$ whereas $X_i(s_1)$ is (still) F. Therefore, $s_1 \in \mathcal{G}$. □

Lemma 4.7.13 *Start with a state s of $\mathcal{F}$. Assume that there exists a process i for which $X_i(s) = F$ and for which $(X_{i-1}(s), X_{i+1}(s)) = (\underrightarrow{\#}, \underleftarrow{\#})$. Then, with probability at least $1/2$, within time 2, a state of $\mathcal{G} \cup \mathcal{P}$ is reached.*

PROOF. The hypothesis can be summarized into the form $(X_{i-1}(s), X_i(s), X_{i+1}(s)) = (\underrightarrow{\#}, F, \underleftarrow{\#})$. Since $i-1$ and $i+1$ point in different directions, by moving to the right of $i+1$ there is a process k pointing to the left such that process $k+1$ either points to the right or is in $\{E_R, R, F\}$, i.e., $X_k(s) \in \{\underleftarrow{W}, \underleftarrow{S}, \underleftarrow{D}\}$ and $X_{k+1}(s) \in \{E_R, R, F, \underrightarrow{W}, \underrightarrow{S}, \underrightarrow{D}\}$. If $X_k(s) \in \{\underleftarrow{W}, \underleftarrow{S}\}$ then $s \in \mathcal{G}$ and we are done. Thus, we can restrict our attention to the case where $X_k(s) = \underleftarrow{D}$.

We show that the event NEXT$((\texttt{flip}_k, \texttt{left}), (\texttt{flip}_{k+1}, \texttt{right}))$, which by Proposition 4.4.2 has probability at least $1/2$, leads in time at most 2 to $\mathcal{G} \cup \mathcal{P}$. Let $\mathcal{A}$ be an adversary of *Unit-Time*, and let α be an execution of M that corresponds to an execution of $H(M, \mathcal{A}, \{s\})$ where if process k flips before process $k+1$ then process k flips left, and if process $k+1$ flips before process k then process $k+1$ flips right.

Within time 2, process k takes at least two steps and hence goes to configuration W. Let $j \in \{k, k+1\}$ be the first of k and $k+1$ that reaches W and let s_1 be the state reached after the first time process j reaches W. If some process reached P in the meantime, then we are done. Otherwise there are two cases to consider. If $j = k$, then, $\texttt{flip}_k$ gives `left` and $X_k(s_1) = \underleftarrow{W}$ whereas X_{k+1} is (still) in $\{E_R, R, F, \underrightarrow{\#}\}$. Therefore, $s_1 \in \mathcal{G}$. If $j = k+1$, then $\texttt{flip}_{k+1}$ gives `right` and $X_{k+1}(s_1) = \underrightarrow{W}$ whereas $X_k(s_1)$ is (still) in $\{\underleftarrow{D}, F\}$. Therefore, $s_1 \in \mathcal{G}$. □

Proposition 4.7.14 *Start with a state s of $\mathcal{F}$. Then, with probability at least $1/2$, within time 2, a state of $\mathcal{G} \cup \mathcal{P}$ is reached. Equivalently:*

$$\mathcal{F} \xrightarrow[1/2]{2} \mathcal{G} \cup \mathcal{P}.$$

PROOF. The two different hypotheses of Lemmas 4.7.12 and 4.7.13 form a partition of $\mathcal{F}$. □

Finally, we prove $\mathcal{RT} \xrightarrow{3} \mathcal{F} \cup \mathcal{P}$.

Proposition 4.7.15 *Starting from a state s of $\mathcal{RT}$, then, within time 3, a state of $\mathcal{F} \cup \mathcal{P}$ is reached. Equivalently:*

$$\mathcal{RT} \xrightarrow{3} \mathcal{F} \cup \mathcal{P}.$$

PROOF. Let s be a state of $\mathcal{RT}$. If $s \in \mathcal{F}$, then we are trivially done: We can therefore restrict ourselves to the case where in s each process is in $\{E_R, R, W, S, D\}$ and where there exists at least one process in $\{W, S, D\}$. Furthermore we can restrict ourselves to the case where no process reaches $\mathcal{P}$ in time 3, i.e., where the state stays in $\mathcal{RT}$. (Else we are done.) Let $\mathcal{A}$ be an adversary of *Unit-Time*, and let α be the execution of M that corresponds to an execution of $H(M, \mathcal{A}, \{s\})$.

Within time 1 a process reaches $\{S, D, F\}$. Therefore Within time 2 a process reaches $\{D, F\}$. Therefore Within time 3 a process reaches $\{F\}$. As, by assumption, the state stays in $\mathcal{RT}$ in time 3, we have therefore proven that $\mathcal{F}$ is reached in time 3. □

Chapter 5

A Deterministic Scheduling Protocol

In this chapter we present a scheduling problem, analyze it and provide optimal *deterministic* solutions for it. The proof involves re-expressing the problem in graph-theoretical terms. In particular the main tool used in the proof of optimality is Ore's Deficiency Theorem [45] giving a dual expression of the size of a maximum matching in a bipartite graph. We will consider in Chapter 7 the randomized version of this scheduling problem.

5.1 Introduction

Many control systems are subject to failures that can have dramatic effects. One simple way to deal with this problem is to build in some redundancy so that the whole system is able to function even if parts of it fail. In a general situation, the system's manager has access to some observations allowing it to control the system efficiently. Such observations bring information about the state of the system that might consist of partial fault reports. The available controls might include repairs and/or replacement of faulty processors.

To model the problem, one needs to make assumptions regarding the occurrence of faults. Typically, they are assumed to occur according to some stochastic process. To make the model more tractable, one often considers the process to be memoryless, i.e. faults occur according to some exponential distribution. However, to be more realistic, many complications and variations can be introduced in the

stochastic model, and they complicate the time analysis. Examples are: a processor might become faulty at any time or only during specific operations; the fault rate might vary according to the work load; faults might occur independently among the processors or may depend on proximity. The variations seem endless and the results are rarely general enough so as to carry some information or methodology from one model to another.

One way to derive general results, independent of the specific assumptions about the time of occurrence of faults, is to adopt a *discrete time*, that, instead of following an absolute frame, is incremented only at each occurrence of a fault. Within this framework, we measure the maximal number of faults to be observed until the occurrence of a crash instead of the maximal time of survival of a system until the occurrence of a crash.

As an introduction to this general situation, we make the following assumptions and simplifications:

Redundancy of the system: We assume the existence of a pool $\mathcal{N}$ composed of p identical processors from among which, at every time t, a set s_t of n processors is selected to configure the system. The system works satisfactorily as long as at least $n - m$ processors among the n currently in operation are not faulty. However, the system cannot tolerate more than m faults at any given time: it stops functioning if $m + 1$ processors among these n processors are faulty.

Occurrence of faults, reports and logical time: We consider the situation in which failures do not occur simultaneously and where, whenever a processor fails, a report is issued, stating that a failure has occurred, but without specifying the location of the failure. (Reporting additional information might be too expensive or time consuming.) Based on these reports, the scheduler might decide to reconfigure the system whenever such failure is reported. As a result, we restrict our attention to the discrete model, in which time t corresponds to the t-th failure in the system.

Repairs: No repair is being performed.

Deterministic Algorithms: We assume that the scheduler does not use randomness.

Since the universe consists of only p processors, and one processor fails at each time, no scheduling policy can guarantee that the system survives beyond time p. (A better a priori upper bound is $p - n + m + 1$: at this time, only $n - m - 1$ processors are still non-faulty. This does not allow for the required quorum of $n - m$

non-faulty processors.) But some scheduling policies seem to allow the system to survive longer than others. An obviously bad policy is to choose n processors once and for all and never to change them: the system would then collapse at time $m+1$. This chapter investigates the problem of determining the best survival time.

This best survival time is defined from a worst-case point-of-view: a *given* scheduler allows the system to survive (up to a certain time) only if it allows it to survive against *all* possible failure patterns in which one processor fails at each time.

Our informal description so far apparently constrains the faults to occur in on-line fashion: for each t, the t-th fault occurs before the scheduler decides the set s_{t+1} to be used subsequently. However, since we have assumed that no reports about the locations of the faults are available, there is no loss of generality in requiring the sets s_t to be determined a priori. (Of course, in practice, some more precise fault information may be available, and each set s_t would depend on the fault pattern up to time t.) Also, as we have assumed a *deterministic* scheduler, we can assume that the decisions $s_1, \ldots, s_p$ are *revealed* before the occurrence of any fault. We express this by saying that the faults occur in an off-line fashion.

5.2 The Model

Throughout this chapter, we fix a universal set $\mathcal{N}$ of processors, and let p denote its cardinality. We also fix a positive integer n ($n \leq p$) representing the number of processors that are needed at each time period, and a positive integer m representing the number of failures that can be tolerated ($m < n$).

We model the situation described in the introduction as a simple game between two entities, a *scheduler* and an *adversary*. The game consists of only one round, in which the scheduler plays first and the adversary second. The scheduler plays by selecting a sequence of p sets of processors (the *schedule*), each set of size n, and the adversary responds by choosing, from each set selected by the scheduler, a processor to kill. We consider only sequences of size p because the system must collapse by time p, since, at each time period, a new processor breaks down.

Formally, a schedule $\mathcal{S}$ is defined to be a finite sequence, $s_1, \ldots, s_p$, of subsets of $\mathcal{N}$, such that $|s_t| = n$ for all t, $1 \leq t \leq p$. An *adversary* $\mathcal{A}$ is defined to be a function associating to every schedule $\mathcal{S} = (s_1, \ldots, s_p)$ a sequence $\mathcal{A}(\mathcal{S}) = (f_1, \ldots, f_p)$ of elements of $\mathcal{N}$ such that $f_t \in s_t$ for every t.

Now let $\mathcal{S}$ be a schedule, and $\mathcal{A}$ an adversary. Define the *survival time*, $T(\mathcal{S}, \mathcal{A})$, to be the largest value of t such that, for all $u \leq t$, $|\{f_1, \ldots f_u\} \cap s_u| \leq m$, (where

$(f_1, \ldots, f_p) = \mathcal{A}(\mathcal{S})$). That is, for all time periods u up to and including time period t, there are no more than m processors in the set s_u that have failed by time u.

We are interested in the minimum survival time for a particular schedule, with respect to arbitrary adversaries. Thus, we define the *minimum survival time* for a schedule, $T(\mathcal{S})$, to be $T(\mathcal{S}) \stackrel{\text{def}}{=} \min_{\mathcal{A}} \mathcal{T}(\mathcal{S}, \mathcal{A})$. An adversary $\mathcal{A}$ for which $T(\mathcal{S}) = \min_{\mathcal{A}} \mathcal{T}(\mathcal{S}, \mathcal{A})$ is said to be *minimal* for $\mathcal{S}$. Finally, we are interested in determining the schedule that guarantees the greatest minimum survival time. Thus, we define the *optimum survival time* t_{opt}, to be $\max_{\mathcal{S}} T(\mathcal{S}) = \max_{\mathcal{S}} \min_{\mathcal{A}} T(\mathcal{S}, \mathcal{A})$. Also define a schedule $\mathcal{S}$ to be *optimum* provided that $T(\mathcal{S}) = t_{\text{opt}}$. Our objectives in this chapter are to compute t_{opt} as a function of p, n and m, to exhibit an optimum schedule, and to determine a minimal adversary for each schedule.

5.3 The Result

Recall that $1 \leq m < n \leq p$ are three fixed integers. Our main result is stated in terms of the following function defined on the set of positive real numbers (see Figure 5.1):

$$h_{n,m}(k) \stackrel{\text{def}}{=} \left\lfloor \frac{k}{n} \right\rfloor m + \left(k - \left\lfloor \frac{k}{n} \right\rfloor n + m - n \right)^+,$$

where $(x)^+ = \max(x, 0)$. In particular, $h_{n,m}(k) = \frac{k}{n} m$ when n divides k.

The main result of this chapter is:

Theorem 5.3.1

$$t_{\text{opt}} = h_{n,m}(p).$$

We will present our proof in two lemmas proving respectively that t_{opt} is no smaller and no bigger than $h_{n,m}(p)$.

Lemma 5.3.2

$$t_{\text{opt}} \geq h_{n,m}(p).$$

PROOF. Consider the schedule $\mathcal{S}_{\text{trivial}}$ in which the p processors are partitioned into $\lfloor \frac{p}{n} \rfloor$ batches of n processors each and one batch of $q = p - \lfloor \frac{p}{n} \rfloor n$. Each of the first $\lfloor \frac{p}{n} \rfloor$ batches is used m time periods and then set aside. Then, the last batch of processors along with any $n - q$ of the processors set aside is used for $(m + q - n)^+$ time periods. It is easy to see that no adversary can succeed in killing $m + 1$ processors within a batch before this schedule expires. □

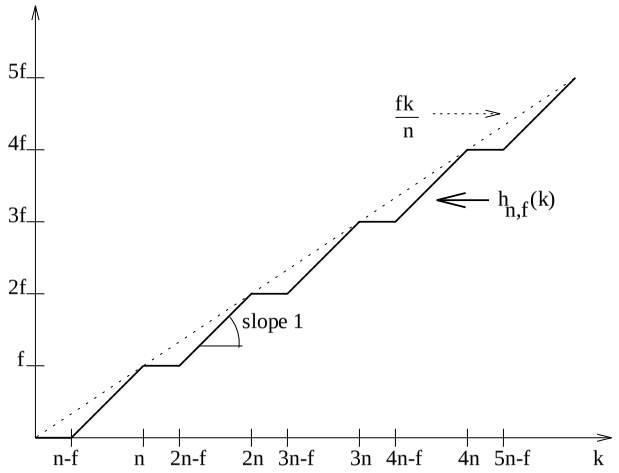

Figure 5.1: The function $h_{n,m}(k)$

In order to prove the other direction of Theorem 5.3.1, we need the following result about the rate of increase of the function $h_{n,m}(k)$.

Lemma 5.3.3 *For $0 \leq k$ and $0 \leq l \leq n$ we have $h_{n,m}(k) \leq h_{n,m}(k+l) + n - l - m$.*

PROOF. Notice first that $h_{n,m}(k) = h_{n,m}(k+n) - m$ for all $k \geq 0$. Moreover, the function h increases at a sublinear rate (see Figure 5.1) so that, for $p, q \geq 0$, we have $h_{n,m}(p+q) \leq h_{n,m}(p) + q$. Letting $p = k + l$ and $q = n - l$, we obtain

$$h_{n,m}(k) = h_{n,m}(k+n) - m \leq h_{n,m}(k+l) + n - l - m,$$

which proves the lemma. □

5.4 The Upper Bound

In this section we establish the other direction of the main theorem. We begin with some general graph theoretical definitions.

Definition 5.4.1

- For every vertex v of a graph G, we let $\gamma_G(v)$ denote the set of vertices adjacent to v. We can extend this notation to sets: for all sets C of vertices $\gamma_G(C) \stackrel{\text{def}}{=} \cup_{v \in C} \gamma_G(v)$.

- For every bipartite graph G, $\nu(G)$ denotes the size of a maximum matching of G.
- For every pair of positive integers L, R, a *left totally ordered* bipartite graph of size (L, R) is a bipartite graph with bipartition $\mathcal{L}, \mathcal{R}$, where $\mathcal{L}$ is a totally ordered set of size L and $\mathcal{R}$ is a set of size R. We label $\mathcal{L} = \{a_1, \ldots, a_L\}$ so that, $a_i < a_j$ for every $1 \leq i < j \leq L$. For every $\mathcal{L}' \subseteq \mathcal{L}$ and $\mathcal{R}' \subseteq \mathcal{R}$, the subgraph induced by $\mathcal{L}'$ and $\mathcal{R}'$ is a left totally ordered bipartite graph with the total order on $\mathcal{L}$ inducing the total order on $\mathcal{L}'$.
- Let G be a left totally ordered bipartite graph of size (L, R). For $t = 1, \ldots, L$, we let $I_t(G)$ denote the left totally ordered subgraph of G induced by the subsets $\{a_1, a_2, \ldots, a_{t-1}\} \subseteq \mathcal{L}$ and $\gamma_G(a_t) \subseteq \mathcal{R}$.

Let us justify quickly the notion of left total order. In this definition, we have in mind that $\mathcal{L}$ represents the labels attached to the different times, and that $\mathcal{R}$ represents the labels attached to the available processors. The times are naturally ordered. The main argument used in the proof is to reduce an existing schedule to a shorter one. In doing so, we in particular select a subsequence of times. Although these times are not necessarily consecutive, they are still naturally ordered. The total order on $\mathcal{L}$ is the precise notion formalizing the ordering structure characterizing time.

Consider a finite schedule $\mathcal{S} = s_1, \ldots, s_t$. In graph theoretic terms, it can be represented as a left totally ordered bipartite graph G with bipartition $\mathcal{T} = \{1, 2, \ldots, T\}$ and $\mathcal{N} = \{1, 2, \ldots, p\}$. There is an edge between vertex $t \in \mathcal{T}$ and vertex $i \in \mathcal{N}$ if the processor i is selected at time t. The fact that, for all t, $|s_t| = n$ translates into the fact that vertex $t \in \mathcal{T}$ has degree n. For such a bipartite graph, the game of the adversary consists in selecting one edge incident to each vertex $t \in \mathcal{T}$.

Observe that the adversary can kill the schedule at time t if it has already killed, before time t, m of the n processors used at time t. It then kills another one at time t and the system collapses. In terms of the graph G, there exists an adversary that kills the schedule at time t if and only if the subgraph $I_t(G)$ has a matching of size m, i.e. $\nu(I_t(G)) \geq m$. Therefore, the set $\mathcal{P}$ that we now define represents the set of integers L and R for which there exists a schedule that survives at time L, when R processors are available.

Definition 5.4.2 Let L and R be two positive integers. $(L, R) \in \mathcal{B}$ iff there exists a left totally ordered bipartite graph G of size (L, R) with bipartition $\mathcal{L}$ and $\mathcal{R}$ satisfying the two following properties:

1. All vertices in $\mathcal{L}$ have degree exactly equal to n,
2. For every $t = 1, \ldots, |\mathcal{L}|$, all matchings in $I_t(G)$ have size at most equal to $m-1$, i.e. $\nu(I_t(G)) \leq f - 1$.

The main tool used in the proof of Theorem 5.3.1 is the following duality result for the maximum bipartite matching problem, known as Ore's Deficiency Theorem [45]. A simple proof of this theorem and related results can be found in [39].

Theorem 5.4.1 *Let G be a bipartite graph with bipartition A and B. Then the size $\nu(G)$ of a maximum matching is given by the formula:*

$$\nu(G) = \min_{C \subseteq B} \left[|B - C| + |\gamma_G(C)| \right]. \tag{5.1}$$

The following lemma is crucial for our proof.

Lemma 5.4.2 *There are no positive integers L and R such that $(L, R) \in \mathcal{B}$ and such that $L > h_{n,m}(R)$.*

PROOF. Working by contradiction, consider two positive integers L and R such that $(L, R) \in \mathcal{B}$ and $L > h_{n,m}(R)$. We first show the existence of two integers L' and R' such that $L' < L$, $(L', R') \in \mathcal{B}$ and $L' > h_{n,m}(R')$.

Let $\mathcal{L} = \{a_1, a_2, \ldots, a_L\}$ and $\mathcal{R} = \{b_1, b_2, \ldots, b_R\}$ be the bipartition of the graph G whose existence is ensured by the hypothesis $(L, R) \in \mathcal{B}$.

We apply Theorem 5.4.1 to the graph $I_L(G)$ where we set $A = \{a_1, a_2, \ldots, a_{L-1}\}$ and $B = \gamma_G(a_L)$. Let C denote a subset of B for which the minimum in (5.1) is attained. (C is possibly empty.) Define $\mathcal{L}' \stackrel{\text{def}}{=} \mathcal{L} - (\{a_L\} \cup \gamma_{I_L(G)}(C))$ and $\mathcal{R}' \stackrel{\text{def}}{=} \mathcal{R} - C$ and let L' and R' denote the cardinalities of $\mathcal{L}'$ and $\mathcal{R}'$. Hence, $L' = L - 1 - |\gamma_{I_L(G)}(C)|$ so that $L' < L$. Consider the bipartite subgraph G' of G induced by the set of vertices $\mathcal{L}' \cup \mathcal{R}'$. In other words, in order to construct G' from G, we remove the set $C \cup \{a_L\}$ of vertices and *all* vertices adjacent to some vertex in C. We have illustrated this construction in Figure 5.2. In that specific example, $n = 4$, $m = 3$, $L = 6$ and $R = 7$, while $h_{4,3}(7) = 5$. One can show that $C = \{b_5, b_6, b_7\}$ and as a result G' is the graph induced by the vertices $\{a_1, a_2, a_3, a_4, b_1, b_2, b_3, b_4\}$. The graph G' has size $(L', R') = (4, 4)$.

We first show that $(L', R') \in \mathcal{B}$. Since the vertices in $\mathcal{L}'$ correspond to the vertices of $\mathcal{L} - \{a_L\}$ not connected to C, their degree in G' is also n. Furthermore, G', being

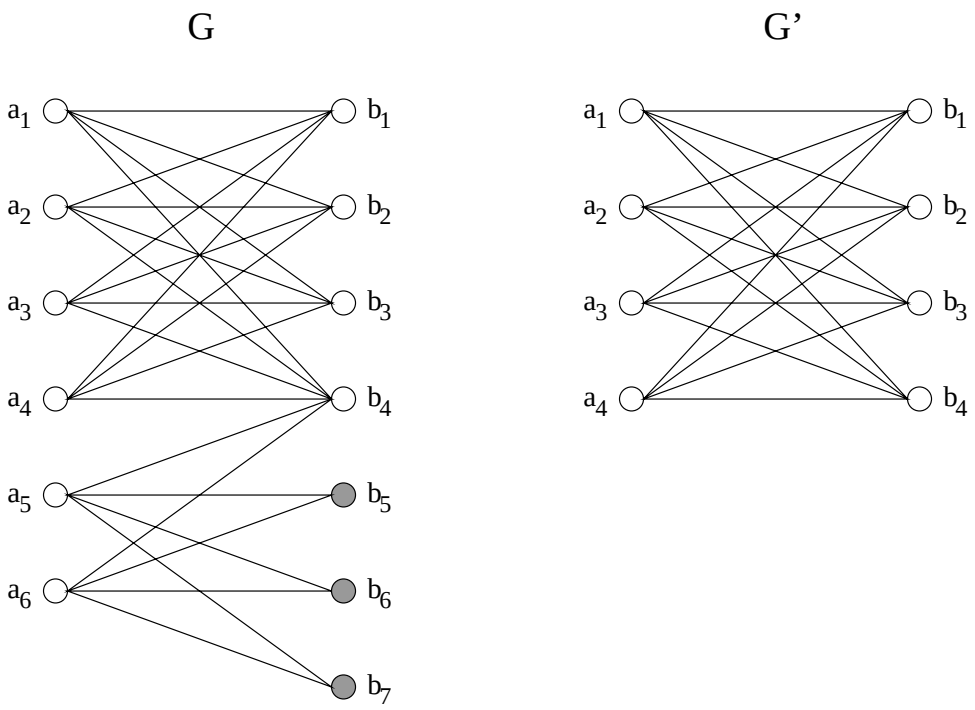

Figure 5.2: An example of the construction of G' from G. The vertices in C are darkened.

a subgraph of G, inherits property 2 of Definition 5.4.2. Indeed, assume that there is a vertex $a_{t'}$ in G' such that $I_{t'}(G')$ has a matching of size m. Let t be the label of the corresponding vertex in graph G. Since the total order on $\mathcal{L}'$ is induced by the total order on $\mathcal{L}$, $I_{t'}(G')$ is a subgraph of $I_t(G)$. Therefore, $I_t(G)$ would also have a matching of size m, a contradiction.

Let us show that $L' > h_{n,m}(R')$. The assumption $(L,R) \in \mathcal{B}$ implies that $m - 1 \geq \nu(I_L(G))$. Using Theorem 5.4.1 and the fact that $B = \gamma_G(L)$ has cardinality n, this can be rewritten as

$$\begin{aligned} f - 1 &\geq \nu(I_L(G)) = |B - C| + |\gamma_{I_L(G)}(C)| \\ &= n - |C| + |\gamma_{I_L(G)}(C)|. \end{aligned} \tag{5.2}$$

Since $C \subseteq B \subseteq \mathcal{R}$, we have that $0 \leq |C| \leq n \leq R$ and, thus, the hypotheses of Lemma 5.3.3 are satisfied for $k = R - |C|$ and $l = |C|$. Therefore, we derive from the lemma that

$$h_{n,m}(R') = h_{n,m}(R - |C|) \leq h_{n,m}(R) + n - |C| - f.$$

Using (5.2), this implies that

$$h_{n,m}(R') \leq h_{n,m}(R) - |\gamma_{I_L(G)}(C)| - 1.$$

By assumption, L is strictly greater than $h_{n,m}(R)$, implying

$$h_{n,m}(R') < L - 1 - |\gamma_{I_L(G)}(C)|.$$

But the right-hand-side of this inequality is precisely L', implying that $L' > h_{n,m}(R')$.

We have therefore established that for all integers L and R such that $(L,R) \in \mathcal{B}$ and $L > h_{n,m}(R)$, there exists two integers L' and R' such that $L' < L$, $(L',R') \in \mathcal{B}$ and $L' > h_{n,m}(R')$. Among all such pairs (L,R), we select the pair for which L is minimum. By the result that we just established, we obtain a pair (L',R') such that $(L',R') \in \mathcal{B}$ and $L' < L$. This contradicts the minimality of L.

□

Lemma 5.4.3

$$t_{\text{opt}} \leq h_{n,m}(p).$$

PROOF. By assumption, $(t_{\text{opt}}, N) \in \mathcal{B}$. Hence this result is a direct consequence of Lemma 5.4.2 .

□

This Lemma along with Lemma 5.3.2 proves Theorem 5.3.1.

In the process of proving Lemma 5.3.2 we proved that $\mathcal{S}_{\text{trivial}}$ is an optimum schedule. On the other hand, the interpretation of the problem as a graph problem also demonstrates that the adversary has a polynomial time algorithm for finding an optimum killing sequence for each schedule $\mathcal{S}$. When provided with $\mathcal{S}$, the adversary needs only to compute a polynomial number (actually fewer than p) of maximum bipartite matchings, for which well known polynomial algorithms exist (for the fastest known, see [31]).

5.5 Extensions

The problem solved in this chapter is a first step towards modeling complex resilient systems and there are many interesting extensions. We mention only a few.

An interesting extension is to consider the case of a system built up of processors of different types. For instance consider the case of a system built up of a total of n processors, that is reconfigured at each time period and that needs at least g_1 non-faulty processors of type 1 and at least g_2 non-faulty processors of type 2 in order to function satisfactorily. Assume also that these processors are drawn from a pool $\mathcal{N}_1$ of p_1 processors of type 1 and a pool $\mathcal{N}_2$ of p_2 processors of type 2, that $\mathcal{N}_1 \cap \mathcal{N}_2 = \emptyset$, that that there are no repairs. It is easy to see that the optimum survival time t_{opt} is at least the survival time of every strategy for which the number of processors of type 1 and type 2 is kept constant throughout. Hence:

$$t_{\text{opt}} \geq \max_{\{(n_1,n_2);n_1+n_2=n\}} \min\left(h_{n_1,n_1-g_1}(p_1), h_{n_2,n_2-g_2}(p_2)\right).$$

It would be an interesting question whether t_{opt} is exactly equal to this value or very close to it.

Extend the definition of a *scheduler* to represent a randomized scheduling protocol. (Phrased in this context, the result presented in this chapter is only about deterministic scheduling protocols.) A scheduler is called *adversary-oblivious* if it decides the schedule independently of the choices $f_1, f_2, \ldots$ made by the adversary. An *off-line* adversary is an adversary that has access to the knowledge of the full schedule $s_1, s_2, \ldots$ before deciding the full sequence $s_1, s_2, \ldots$ Note that, by definition, off-line adversaries make sense only with adversary-oblivious schedulers. By comparison, an *on-line* adversary decides for each time t which processor f_t to kill, without knowing the future schedule: at each time t the adversary decides f_t based on the sole knowledge of $s_1, \ldots, s_t$ and of $f_1, \ldots, f_{t-1}$. In this more general framework, the quantity

we want to determine is

$$t_{\text{opt}} \stackrel{\text{def}}{=} \max_{\mathcal{S}} \min_{\mathcal{A}} E\left[T(\mathcal{S}, \mathcal{A})\right]. \tag{5.3}$$

For an adversary-oblivious, randomized scheduler, one can consider two cases based on whether the adversary is on-line or off-line. As is easily seen, if the adversary is off-line, randomness does not help in the design of optimal schedulers: introducing randomness in the schedules cannot increase the survival time if the adversary gets full knowledge of the schedule before committing to any of its choices. As a result, the off-line version corresponds to the situation investigated in this chapter.

It is of interest to study the online version of Problem 5.3. On-line adversaries model somewhat more accurately practical situations: faults naturally occur in an on-line fashion and the role of the program designer is then to design a scheduler whose expected performance is optimum. We study this question for $m = 1$ in Chapter 7 and provide in this case a characterization of the set of optimal randomized scheduling policies.

Chapter 6

Establishing the Optimality of a Randomized Algorithm

Proving the precise optimality of a randomized algorithm solving a given problem $\mathcal{P}$ is always a very difficult and technical enterprise and only very few such proofs exist (see [25, 61]).

A first difficulty is to define an adequate probabilistic model for the analysis of the randomized algorithms solving $\mathcal{P}$. This model must take into account that, in general, some choices are not in the control of the algorithm considered but, instead, controlled by the adversary. It must also reckon with the fact that each randomized algorithm uses different random coins and hence carries a different probabilistic structure; nevertheless a common probabilistic structure has to be defined allowing for the comparison of all the algorithms solving $\mathcal{P}$. The few papers published so far and dealing with lower bounds [25, 35, 33, 61] rarely address this issue. ([25] introduces an ad-hoc model for the proof presented there.) The model presented in Chapter 2 is, to our knowledge, the first to allow formal proofs of lower-bounds for general randomized algorithms.

A second difficulty is that, for a given problem $\mathcal{P}$, the set of randomized algorithms is infinite in general and hence looking for an optimal randomized algorithm involves doing a maximization over an infinite set.

We let $f(\pi, \mathcal{A})$ denote the performance of a given randomized algorithm π when used in conjunction with an adversary $\mathcal{A}$. Examples of typical performances $f(\pi, \mathcal{A})$ are the expected running time or the probability of "good termination" when the algorithm π is used in conjunction with the adversary $\mathcal{A}$. By changing changing if necessary $f(\pi, \mathcal{A})$ into $-f(\pi, \mathcal{A})$ we can always assume that the algorithms π are

chosen so as to maximize the value $f(\pi, \mathcal{A})$. The worst case performance of an algorithm π is given by $\inf_{\mathcal{A}} f(\pi, \mathcal{A})$, and therefore the optimal worst case performance is given by $\sup_{\pi} \inf_{\mathcal{A}} f(\pi, \mathcal{A})$.

As discussed in Chapter 2, the problem of analyzing an algorithm – and proving its optimality – is best described in the language of game theory. (See also Section 8.3 for a presentation of the main notions of Game Theory.) We let *Player*(1) be the entity selecting the algorithm (in short, the *algorithm designer*) and *Player*(2) be the entity selecting the adversary (in short, the *adversary designer*). If *Player*(1) selects the algorithm π and if *Player*(2) selects the adversary $\mathcal{A}$, the game played consists of the alternative actions of the algorithm and the adversary: *Player*(1) takes all the actions as described by π until the first point where some choice has to be resolved by the adversary; *Player*(2) then takes actions to resolve this choice as described by $\mathcal{A}$ and *Player*(1) resumes action once the choice has been resolved...

Note that, by definition, an algorithm π is defined independently of a given adversary $\mathcal{A}$. On the other hand, an adversary might seem to be defined only in terms of a given algorithm: the adversary is by definition the entity that resolves all choices not in the control of the algorithm considered. If the model allowed for such an asymmetry between the notions of algorithm and adversary we could not speak of an adversary independently of the algorithm it would be associated to. For reasons that will soon be explained, it is critical for our method to model adversaries independently of any specific algorithm. In this case the algorithm designer and the adversary designer are two well defined players playing a zero sum non-cooperative game. The set of strategies Π and A are respectively the algorithms π and the adversaries $\mathcal{A}$. The rules governing the interaction of the two players during an execution of the game are set by the description of the problem $\mathcal{P}$.

A very delicate matter is the nature of the information about the system held by either player when taking a step, and the formal way this information is taken into account in the model. Generally, some information about the moves of either player is conveyed onto the other player during the execution (i.e., during the game). A player having a more precise knowledge of the state of the system is more capable to act optimally toward its goal (maximizing or minimizing the performance function $f(\pi, \mathcal{A})$). The proof of optimality of a player is therefore tantamount to proving that, at each round, the player uses optimally the information available in order to take its next move. This is in general a very difficult task for which no clear general approach seems to exist. Nevertheless, using the concept of saddle point in game theory allows us to derive a general proof strategy for proving the optimality of an algorithm. We now present and discuss this methodology.

If adversaries are each defined independently of any specific algorithm π, for every adversary $\mathcal{A}$ we can consider the family $(f(\pi, \mathcal{A}))_\pi$ obtained by letting π range over the whole set of algorithms. Therefore, in this case, for every adversary $\mathcal{A}$, the quantity $\sup_\pi f(\pi, \mathcal{A})$ is well-defined.

By Lemma 8.2.2, for every algorithm π_0 and every adversary $\mathcal{A}_0$ we have $\inf_\mathcal{A} f(\pi_0, \mathcal{A}) \leq \sup_\pi f(\pi, \mathcal{A}_0)$. Furthermore, this inequality is an equality only if π_0 is an optimal algorithm. This simple fact provides us with the following general proof methodology to attempt to prove that a given algorithm π_0 is optimal.

1. Construct a Π/A-structure modeling the interaction between *Player*(1) and *Player*(2). (This means in particular that an adversary is defined independently of the choice of any specific algorithm.)
2. Provide an adversary $\mathcal{A}_0$ such that $\inf_\mathcal{A} f(\pi_0, \mathcal{A}) = \sup_\pi f(\pi, \mathcal{A}_0)$.

By Proposition 8.2.3, the existence of a pair $(\pi_0, \mathcal{A}_0)$ realizing the equality $\inf_\mathcal{A} f(\pi_0, \mathcal{A}) = \sup_\pi f(\pi, \mathcal{A}_0)$ occurs if and only if $\max_\pi \inf_\mathcal{A} f(\pi, \mathcal{A}) = \min_\mathcal{A} \sup_\pi f(\pi, \mathcal{A})$.

(This equality succinctly expresses the three following facts. 1) $\sup_\pi \inf_\mathcal{A} f(\pi, \mathcal{A}) = \inf_\mathcal{A} \sup_\pi f(\pi, \mathcal{A})$. 2) A protocol π achieves the sup in $\sup_\pi \inf_\mathcal{A} f(\pi, \mathcal{A})$ i.e., $\sup_\pi \inf_\mathcal{A} f(\pi, \mathcal{A}) = \max_\pi \inf_\mathcal{A} f(\pi, \mathcal{A})$. And 3) an adversary $\mathcal{A}$ achieves similarly the inf in $\min_\mathcal{A} \sup_\pi f(\pi, \mathcal{A})$ i.e., $\inf_\mathcal{A} \sup_\pi f(\pi, \mathcal{A}) = \min_\mathcal{A} \sup_\pi f(\pi, \mathcal{A})$.)

To find an algorithm and prove its optimality using the previous methodology, we are therefore led to model algorithms and adversaries in such a way that the equality $\sup_\pi \inf_\mathcal{A} f(\pi, \mathcal{A}) = \inf_\mathcal{A} \sup_\pi f(\pi, \mathcal{A})$ holds. There exists two cases where this happens:

Von Neumann: We assume that the set Π of strategies of *Player*(1) is the set of probability distributions on a given *finite* set I and that, similarly, the set A of strategies of *Player*(2) is the set of probability distributions on a given *finite* set J. When saying this, we actually abuse language and identify a probability distribution with the procedure consisting in drawing an element at random according to this probability distribution. Hence, by convention,

for every $\pi \in \Pi$, the strategy π consists in drawing an element i of I at random and according to the distribution π. Similarly, for every $\mathcal{A} \in A$, the strategy $\mathcal{A}$ consists in drawing an element j of J at random and according to the distribution $\mathcal{A}$. For every $(i,j) \in I \times J$ and every strategies π and $\mathcal{A}$ resulting in the selection of i and j, a predetermined cost $T(i,j)$ is incurred. The performance $f(\pi, \mathcal{A})$ is by assumption the expected cost $E_{\pi,\mathcal{A}}[T]$ obtained under the strategies π and $\mathcal{A}$.

The game just described can be encoded as a matrix game: I and J are the sets of pure strategies whereas Π and A are the sets of mixed strategies of the game. By Von Neumann's theorem, (see Theorem 8.3.2), $\max_\pi \min_\mathcal{A} f(\pi, \mathcal{A}) = \min_\mathcal{A} \max_\pi f(\pi, \mathcal{A})$. Recall once more that the finiteness of both I and J is critical for this result.

Strong Byzantine: Assume that the rules of the game played between *Player*(1) and *Player*(2) specify that, in every execution, *Player*(2) first learns *explicitly* the strategy π chosen by *Player*(1) before having to commit itself to any action. (We could picture this by saying that, by convention, an execution begins with *Player*(1) "sending a message" to *Player*(2) disclosing the strategy π under use.) Hence, in this situation, a strategy $\mathcal{A}$ for *Player*(2) is actually a family $\mathcal{A} = (\mathcal{A}_\pi)_\pi$ of strategies $\mathcal{A}_\pi$, one for every strategy π of *Player*(1). We say that $\mathcal{A}_\pi$ is an *adversary specially designed for* π. Assume furthermore that the performance function is such that, for every adversaries $\mathcal{A}$ and $\mathcal{A}'$, for every algorithm π, if $\mathcal{A}_\pi = \mathcal{A}'_\pi$ then $f(\pi, \mathcal{A}) = f(\pi, \mathcal{A}')$. This last property allows us to extend the definition of f: for every algorithm π and every strategy a specially designed for π, we set $f(\pi, a) = f(\pi, \mathcal{A})$ where $\mathcal{A}$ is any adversary such that $\mathcal{A}_\pi = a$. Assume also that A is *stable under reshuffling* in the following sense. Let $(a(\pi))_\pi$ be a family of specially designed adversaries, one for every $\pi \in \Pi$. (Hence, by definition, for every π and π' in Π, there exists an adversary $\mathcal{A}$ and an adversary $\mathcal{A}'$ such that $\mathcal{A}_\pi = a(\pi)$ and $\mathcal{A}'_{\pi'} = a(\pi')$. The adversaries $\mathcal{A}$ and $\mathcal{A}'$ are a priori different.) Then $(a(\pi))_\pi$ is itself an admissible adversary, i.e., an element of A.

The definition $\mathcal{A} = (\mathcal{A}_\pi)_\pi$ immediately shows that an adversary $\mathcal{A}$ does not depend on the choice of a specific algorithm. Hence the Strong Byzantine setting verifies point 1 of our general methodology.

We now show that, in this setting, $\sup_\pi \inf_\mathcal{A} f(\pi, \mathcal{A}) = \inf_\mathcal{A} \sup_\pi f(\pi, \mathcal{A})$ and that, therefore, an algorithm π_0 is optimal if and only if there exists an adversary $\mathcal{A}_0$ such that $\inf_\mathcal{A} f(\pi_0, \mathcal{A}) = \sup_\pi f(\pi, \mathcal{A}_0)$. This will show that the Strong Byzantine setting is well suited for an implementation of our general methodology.

For every $\varepsilon > 0$ and every π in Π – the set of strategies of *Player*(1) – let $\mathcal{A}_\pi(\varepsilon)$ be an adversary specially designed for π and such that

$$f(\pi, \mathcal{A}_\pi(\varepsilon)) \leq \inf_{\mathcal{A}} f(\pi, \mathcal{A}) + \varepsilon \, .$$

The set A of adversaries being stable under reshuffling we can define an adversary $\mathcal{A}(\varepsilon)$ by $\mathcal{A}(\varepsilon) = (\mathcal{A}_\pi(\varepsilon))_\pi$. We have:

$$\begin{aligned} \inf_{\mathcal{A}} \sup_{\pi} f(\pi, \mathcal{A}) &\leq \sup_{\pi} f(\pi, \mathcal{A}(\varepsilon)) \\ &= \sup_{\pi} f(\pi, \mathcal{A}_\pi(\varepsilon)) \\ &\leq \sup_{\pi} \inf_{\mathcal{A}} f(\pi, \mathcal{A}) + \varepsilon \, . \end{aligned}$$

The parameter ε being arbitrary, this shows that $\inf_{\mathcal{A}} \sup_\pi f(\pi, \mathcal{A}) \leq \sup_\pi \inf_{\mathcal{A}} f(\pi, \mathcal{A})$. By Lemma 8.2.1 the converse inequality $\sup_\pi \inf_{\mathcal{A}} f(\pi, \mathcal{A}) \leq \inf_{\mathcal{A}} \sup_\pi f(\pi, \mathcal{A})$ holds trivially. Hence $\sup_\pi \inf_{\mathcal{A}} f(\pi, \mathcal{A}) = \inf_{\mathcal{A}} \sup_\pi f(\pi, \mathcal{A})$ which concludes the proof.

We present here an intuitive interpretation of this result.

Recall first that, as discussed in Page 207, in the expression $\sup_\pi \inf_{\mathcal{A}} f(\pi, \mathcal{A})$, *Player*(2) can be assumed to learn *implicitly* the strategy π chosen by *Player*(1). Symmetrically, in the expression $\inf_{\mathcal{A}} \sup_\pi f(\pi, \mathcal{A})$, *Player*(1) learns implicitly the strategy $\mathcal{A}$ chosen by *Player*(2). Furthermore, as discussed after Equation 8.4, Page 206, the strict inequality $\sup_\pi \inf_{\mathcal{A}} f(\pi, \mathcal{A}) < \inf_{\mathcal{A}} \sup_\pi f(\pi, \mathcal{A})$ means precisely that the outcome of the game is different according to which of the two players can thus learn its opponent's strategy.

If, by construction, *Player*(2) is informed *explicitly* of the strategy used by *Player*(1), its knowledge is evidently unaffected by whether it furthermore learns this fact implicitly (as in the expression $\sup_\pi \inf_{\mathcal{A}} f(\pi, \mathcal{A})$) or not (as in the expression $\inf_{\mathcal{A}} \sup_\pi f(\pi, \mathcal{A})$). Let $\mathcal{A}_0$ be the strategy for *Player*(2) informally described by "Wait for the disclosure of the strategy π selected by *Player*(1). Then select an optimal strategy to be adopted for the rest of the game." It is clear that $\mathcal{A}_0$ is an optimal strategy for *Player*(2). Assume that *Player*(2) plays optimally and adopts this strategy and consider the case where *Player*(1) learns implicitly that *Player*(2) uses strategy $\mathcal{A}_0$. We easily see that this knowledge does not confer any advantage to *Player*(1): *Player*(1) can only derive from it that, for every strategy π it elects, *Player*(2) chooses a corresponding optimal strategy.

> This establishes that, when *Player*(2) is informed explicitly of the strategy used by *Player*(1), *Player*(2) gains no additional advantage in learning implicitly the strategy π used by *Player*(1); and that, in this case, *Player*(1) gains similarly no advantage in being guaranteed that *Player*(2) uses its (optimal) strategy $\mathcal{A}_0$. This shows that, when *Player*(2) is informed explicitly of the strategy used by *Player*(1), the outcome of the game is not affected when one or the other player learns implicitly its opponent's strategy. As argued at the beginning of this discussion, this means that $\sup_\pi \inf_\mathcal{A} f(\pi, \mathcal{A}) = \inf_\mathcal{A} \sup_\pi f(\pi, \mathcal{A})$ for every Strong Byzantine game.

As a short aside and to illustrate the generality of our proof methodology we show that the complicated proof of given in [25] falls in the framework of the Strong Byzantine case of the methodology. (In [25], Graham and Yao consider the Byzantine Generals problem with 3 processes, one of which is faulty.)

By assumption *Player*(2) knows the algorithm π selected by *Player*(1). (This point is never stated explicitly in [25]: the authors of [25] just mention that they "have incorporated the capability for faulty processes to collude, to spy on all communication lines and to wait for messages transmitted by non-faulty processes in the current round to arrive before making decisions on their own messages." Nevertheless the strategies σ_A, σ_B and σ_C of *Player*(2) are described in terms of π.) Hence, as discussed in page 128, a strategy $\mathcal{A}$ of *Player*(2) is actually a family $(\mathcal{A}_\pi)_\pi$ and does not depend on the choice of a specific π.

The performance function is defined to be

$$f(\pi, \mathcal{A}) \stackrel{\text{def}}{=} P_{\pi, \mathcal{A}_\pi}[\textit{good termination}],$$

where $P_{\pi,\mathcal{A}_\pi}$ is the probability on the set of executions induced by the algorithm π and the adversary $\mathcal{A}_\pi$ specially designed for π and associated to $\mathcal{A}$. The event *good termination* is a special event defined in terms of the game studied in [25]. The definition of $f(\pi, \mathcal{A})$ shows immediately that $f(\pi, \mathcal{A}) = f(\pi, \mathcal{A}')$ if $\mathcal{A}_\pi = \mathcal{A}'_\pi$. Furthermore, the set A of adversaries considered in [25] is by assumption stable under reshuffling. We are therefore in the Strong Byzantine setting of the general methodology. We now summarize the proof presented in [25] and show that it follows precisely our general methodology.

The proof of [25] is organized as follows. A specific algorithm π_0 is first given for which the quantity $\textit{performance}(\pi_0) \stackrel{\text{def}}{=} \inf_\mathcal{A} f(\pi_0, \mathcal{A})$ is easily derived.[1] A specific (but very complex) strategy $\mathcal{A}_0$ for *Player*(2) is then described. In order to

[1] This algorithm is actually called $\mathbb{A}_0$ in [25]. We use π_0 to be consistent with the rest of our discussion.

implement strategy $\mathcal{A}_0$,[2] *Player*(2) uses critically its knowledge of the strategy π used by *Player*(1): at every point of the game (i.e., of the execution) *Player*(2) selects its next move by emulating π under certain conditions. Working by induction on the number of rounds of the algorithm π selected by *Player*(1), [25] then shows that, for every π, $f(\pi, \mathcal{A}_0) \leq \mathit{performance}(\pi_0)$. This implies that $\sup_\pi f(\pi, \mathcal{A}_0) \leq \mathit{performance}(\pi_0) \stackrel{\text{def}}{=} \inf_{\mathcal{A}} f(\pi_0, \mathcal{A})$. By Lemma 8.2.2, the converse inequality $\sup_\pi f(\pi, \mathcal{A}_0) \geq \inf_{\mathcal{A}} f(\pi_0, \mathcal{A})$ is trivially true. Hence

$$\sup_{\pi} f(\pi, \mathcal{A}_0) = \inf_{\mathcal{A}} f(\pi_0, \mathcal{A}),$$

which establishes the second point of our general proof methodology and therefore proves that π_0 is optimal.

A natural question is whether the two previous settings, although different in form, are truly different. In slightly more precise terms, the question is whether the existence of a proof of optimality of a given algorithm π_0 in one of the two settings implies the existence of a proof of optimality of π_0 in the other setting.

The following argument tends to suggest a similarity between the two settings. (At least when the performance function $f(\pi, \mathcal{A})$ is equal to the expected value $E_{\pi,\mathcal{A}}[T]$ of a random variable T: recall that the Von Neumann setting requires this condition.)

Let (G, Π, A) be a game[3] between *Player*(1) and *Player*(2) with a performance function f. Consider all the possible modifications of this game obtained by providing *Player*(2) during the execution of the game with some information about the strategy π followed by *Player*(1). All these different games yield the same value $\sup_\pi \inf_{\mathcal{A}} f(\pi, \mathcal{A})$, because, as discussed in Page 207, *Player*(2) can be assumed to learn *implicitly* the complete strategy π in the expression $\sup_\pi \inf_{\mathcal{A}} f(\pi, \mathcal{A})$: receiving some complementary explicit information about π does not then raise its knowledge. This shows that there is a whole spectrum of models for the adversary and that to all of them is attached the same class of optimal algorithms. The two settings presented above, the "Von Neumann setting" and the "Strong Byzantine setting", correspond to two extreme situations where *Player*(2) receives only a bounded number of bits of information about π in the course of an execution, and where it receives the complete description of π at the very beginning of the game. The argument above seems to suggest that the two settings are equally good to establish the optimality of a randomized algorithm.

[2] More precisely, in order to implement $\mathcal{A}_{0,\pi}$, the adversary associated to $\mathcal{A}_0$ and specially designed for π.

[3] See page 205 for a discussion on game theory.

We now discuss two examples, the algorithm of [25], (again), and the scheduling algorithm presented in Chapter 7. These examples reveal that the choice of the setting actually influences greatly the proof of optimality of a randomized algorithm.

Consider first the scheduling problem considered in Chapter 7. The performance function $f(\pi, \mathcal{A})$ considered is the expected value $E_{\pi,\mathcal{A}}[T]$ of a random variable T called the survival-time. In this game, *Player*(2) does not know a priori the strategy selected by *Player*(1). This is formally seen in the model presented in Section 7.2: at each time t, the view of *Player*(2) contains the schedule $s_1, \ldots, s_t$ previously selected by *Player*(1) but contains no additional information about the algorithm π that generated that schedule. We prove that all the algorithms in the set $Prog_0$ defined in page 151 are optimal. Our discussion above therefore shows that these algorithms would similarly be optimal if *Player*(2) was endowed with the spying capability and learned the strategy selected by *Player*(1) at the beginning of the execution. Nevertheless, the proof that we present uses critically that *Player*(2) does *not* have this capability: if *Player*(2) was modeled as knowing the algorithm π, our Lemma 7.3.2 would not be true and all the results of Section 7.6 would not hold anymore.

We consider now the Byzantine Generals problem of [25] and show that, in contrast to the previous example, both the Strong Byzantine setting and the Von Neumann setting can be used to formalize the proof given by Graham and Yao.

The performance function $f(\pi, \mathcal{A})$ considered in [25] is the probability of termination with agreement on a correct value when *Player*(1) selects selects the algorithm π and *Player*(2) plays according to the strategy $\mathcal{A}$. A probability being a special case of an expected value, the performance function $f(\pi, \mathcal{A})$ *is* the expected value $E_{\pi,\mathcal{A}}[T]$ of some variable schema[4] T.

We argued on page 130 that Graham and Yao use the strong byzantine setting in their proof: their *Player*(2) uses as a black box the algorithm π chosen by *Player*(1) in order to generate its own outputs in the course of the execution. Nevertheless an even more careful reading of their proof reveals that *Player*(2) does not need the full knowledge of π but just needs to have access to finitely many values produced by π. Hence we could consider a setting where, by convention, *Player*(1) would provide *Player*(2) with those values: in that case *Player*(2) would need no additional knowledge about π. As argued above on page 131, in this modified game – where *Player*(1) gives some partial information about its strategy – the algorithm π_0 of [25] is still optimal. But we are now in the Von Neumann setting (when analyzing algorithms terminating in finitely many rounds).

[4] The definition of a variable schema is given in Definition 2.4.1.

We have thus argued that, by reducing the information transmitted to *Player*(2) from the complete description of π to only finitely many bits of information we could adapt the proof of [25] given in the Byzantine setting into one given in the Von Neumann setting. We could go further and consider the case where *Player*(1) does not cooperate with *Player*(2) and provides *Player*(2) with *no* information about its strategy (except for what can be "naturally" deduced from an execution). The discussion given above on page 130 shows as before that the algorithm π_0 of [25] is still optimal. Nevertheless, in this case, the proof of [25] does not apply and it is not clear at all how a direct proof would then proceed.

These two examples show that the choice of setting is far from innocent and influences greatly the proof of optimality of a randomized algorithm. We present in the next theorem a result establishing formally that the two settings are in some cases incompatible.

Theorem 6.0.1 *Let $\mathcal{P}$ be a problem, let Π be the set of randomized algorithms solving $\mathcal{P}$ and A be the set of adversaries. Assume that Π contains more then one element. Assume also the interaction between the two players modeled to allow Player(2) to know the algorithm π under use. Then the Von Neumann setting cannot be used to model the interaction between Player(1) and Player(2).*

PROOF. Note first that the Von Neumann setting applies only if the sets Π and A of strategies contain all the convex combinations of their elements: for every strategies π_1 and π_2 in Π, for every non-negative numbers α_1 and α_2 summing to one, $\alpha_1\pi_1+\alpha_2\pi_2$ is also in Π. (Recall that, by definition, in the Von Neumann setting, the strategies π_1 and π_2 are probability distributions so that the linear combinations $\alpha_1\pi_1 + \alpha_2\pi_2$ are well defined.) Hence, in the case where the Von Neumann setting applies, the set Π is either a singleton or an *infinite* set. (The case where Π is a singleton is a degenerate case where *Player*(1) has only one strategy, which is trivially optimal.[5])

Also, in the case where the Von Neumann setting applies, a single probability space $(\Omega, \mathcal{G})$ can be used to analyze the probabilistic behavior of all the pairs $(\pi, \mathcal{A})$ of algorithm and adversary. This probability space can be chosen to be the product space $\Omega = I \times J$ endowed with its complete σ-field $\mathcal{G} = 2^{\Omega}$: Ω and $\mathcal{G}$ are both *finite*.

In the general case, we saw in Chapter 2 that the construction of an adequate probabilistic structure is more complicated and yields a possibly different space $(\Omega_{\pi,\mathcal{A}}, \mathcal{G}_{\pi,\mathcal{A}})$ for every pair $(\pi, \mathcal{A})$. Consider the case, where, as when the Von Neumann setting applies, a single space $(\Omega, \mathcal{G})$ is used for all the pairs $(\pi, \mathcal{A})$. The sample

[5] Remember that all this discussion is geared at finding an optimal algorithm!

space Ω must contain a *different* element ω for each possible execution i.e., for each sequence of actions $(act_1, act_2, act_3, \ldots)$ arising from the game played by $Player(1)$ and $Player(2)$.

Assume the interaction between $Player(1)$ and $Player(2)$ modeled to allow $Player(2)$ to *know* the algorithm π under use. Therefore, by assumption, in every execution, there must be a move (or a sequence of moves), specific to π, undertaken by $Player(1)$, and informing $Player(2)$ of the strategy π chosen.

Working by contradiction, assume that we could use the Von Neumann setting to model the game between $Player(1)$ and $Player(2)$. This means that the set Π can be represented as a set of probability distributions on a given *finite* set I, and that, similarly, the set A can be represented as a set of probability distributions on a given *finite* set J (and that the performance $f(\pi, \mathcal{A})$ is the expected value $E_{\pi,\mathcal{A}}[T]$ of a random variable T). In that case the sample space Ω is equal to $I \times J$ and is therefore *finite*.

On the other hand, as discussed above, in that case, the set Π must be infinite. As in each execution $Player(1)$ informs $Player(2)$ of its strategy π, the set of different executions must therefore also be infinite. This implies that Ω is *infinite*, a contradiction. $\square$

Chapter 7

An Optimal Randomized Algorithm

In this chapter we consider the scheduling problem studied in Chapter 5 but allow algorithms to use randomness. The terms *protocol* and *algorithm* are synonymous but for notational emphasis we favor here the use of protocol: the notations $\Pi, \pi, P_\pi, Pgenerating$ will refer to protocols whereas the notations $A, \mathcal{A}, P_{\mathcal{A}}$, $Agenerating$ will refer to adversaries.

Using the general model presented in Chapter 2 we construct a Π/A-structure associated to the scheduling problem. This allows us to characterize very precisely the optimization problem. We provide a specific randomized protocol and give a *completely formal proof* of its optimality.

This proof is to our knowledge the first completely formal proof of optimality of a randomized algorithm.[1] This chapter should therefore illuminate the power and relevance of the model presented in Chapter 2.

7.1 The Scheduling Problem

7.1.1 Description of the problem

We recall quickly here the setting of the problem. m, n and p are three non-negative integers such that $1 \leq m < n \leq p$.

[1] The proof given by Graham and Yao in [25] still needs some fine tuning...

- p is the total number of processors available.
- n is the number of processors that are necessary to configure the system: at each time n processors are in operation.
- We assume that a processor can become faulty only when in operation and that faults occur one at a time. We also assume that no repairs are available. m is the resiliency parameter of the system: the system functions as long as the set of n processors selected does not include more then m faulty processors. The system crashes as soon as the set of n processors currently in use includes at least $m + 1$ faulty processors.

We define the *discrete time* t of an execution to be the execution point at which the t-th fault occurs. In the sequel, we write time instead of discrete time.

We consider the *blind* situation where, during an execution, the scheduler is informed of the occurrence of a fault whenever one occurs, but does not get any additional information about the location of this fault. Upon notification of a fault the scheduler reconfigures the system. We let s_1 denote the set of n elements selected for the first time and, for $t \geq 2$, we let s_t denote the set of n elements selected after report of fault $t-1$ (i.e., after time $t-1$). We also let f_1, ($f_1 \in s_1$), denote the location of the first fault and generally we let f_t, ($f_t \in s_t$), denote the location of the t-th fault. For the sake of modeling we say that the sequence $f_1, f_2, \ldots$, is decided by an entity called the *adversary*. The purpose of this work is to find a scheduling protocol guarantying the best expected survival time against worst case, *on-line* adversaries. This means that, when selecting the t-th location of fault f_t, the adversary "knows" the whole past $s_1, f_1, s_2, f_2, \ldots, s_t$. We can equivalently say that, for each t, the adversary "receives the information" of what the choice s_t of the scheduling protocol is before deciding what the next fault is. Note that, by contrast, expressed in this language of on-line information, the assumption that the protocol is blind means that, for each t, the scheduling protocol "receives no information" about the choices previously made by the adversary before deciding itself what the set s_t is. We will provide in Section 7.2 a formal setting allowing to interpret these notions of knowledge.

7.1.2 Interpretation using game theory

The purpose of this section is to present some intuition for the formal model presented in Section 7.2. Some notions as the notion of actions, internal and external, that we introduce in the course of the discussion are not presented formally but should be clear from the context.

Following the methodology outlined in Chapters 2 and 6 we describe the scheduling problem presented in Section 7.1.1 as a game played between two players *Player*(1) and *Player*(2). We will refer to this game as the "scheduling game". In this setting, a protocol is a strategy of *Player*(1) and an adversary is a strategy of *Player*(2): *Player*(1) is called the protocol-designer and *Player*(2) is called the adversary-designer. The game played by the two players follows the rules of the scheduling problem described in Section 7.1.1: *Player*(1) plays first, chooses s_1 and informs *Player*(2) of its decision. *Player*(2) then plays and selects f_1 in s_1. No information is conveyed from *Player*(2) to *Player*(1). More generally, in the t-th round, *Player*(1) selects a set s_t and informs *Player*(2) of its choice; *Player*(2) then plays and selects an element f_t in s_t. We adopt the model where *Player*(2) does not know explicitly the strategy π followed by *Player*(1). [2] (In the model where *Player*(2) knows explicitly the strategy π followed by *Player*(1), *Player*(1) "sends a message" informing *Player*(2) of the strategy selected by *Player*(1).)

As discussed in Chapter 6, Page 134, for every protocol π and for every adversary $\mathcal{A}$ the sample space Ω must contain a different ω for each possible execution i.e., for each sequence of actions $(act_1, act_2, act_3, \ldots)$ undertaken in the game played by *Player*(1) and *Player*(2) when following the rules of the scheduling game. Some care has to be devoted to characterize the actions that we here consider. A specific protocol (or adversary) can be implemented in various ways, each of them having specific internal actions. Nevertheless, internal actions are irrelevant for the performance analysis of a protocol: the performance analysis of a protocol is solely measured in terms of its external actions, i.e., the specific actions it undertakes as prescribed by the rules of the game. In a figurative sense, we treat a protocol (resp. an adversary) as a black box and only analyze its external actions.

In our scheduling game and in the model where *Player*(2) does not know the strategy π followed by *Player*(1), the external actions undertaken by *Player*(1) are the successive choices of a set s_t and communications to *Player*(2) of the choice last made. To simplify the discussion we will omit explicit reference of the communication between *Player*(1) and *Player*(2) and implicitly assume that this communication is systematically (and instantaneously) performed at each selection of a set s_t. Similarly, the external actions undertaken by *Player*(2) are the successive choices of an element f_t in the set s_t last selected by *Player*(1). To simplify further the discussion we will abuse language and speak of the actions s_t and f_t in place of "choice of s_t" and "choice of f_t". The assumption $m < p$ clearly implies that the system cannot survive more then p faults. We can therefore restrict our analysis to the

[2] See Section 8.3 and Chapter 6 for a presentation of the notions of explicit/implicit knowledge.

times $t = 1, \ldots, p$. From this discussion we deduce that the sample space

$$\Omega \stackrel{\text{def}}{=} \{(s_1, f_1, \ldots, s_p, f_p);\, s_t \in \mathcal{P}_n(p),\, f_t \in s_t,\, 1 \leq t \leq p\}$$

is big enough for the probabilistic analysis.

This definition is in conformity with the general construction given on page 41 of of Chapter 2. For every protocol π and every adversary $\mathcal{A}$ we defined there

$$\Omega_{\pi,\mathcal{A}} = \{\omega\,;\, \omega \text{ is a } (\pi, \mathcal{A})\text{-execution}\},$$

where

$$\omega = \underbrace{a_1\,(s_1, x_1, y_1)}_{Player(2)}\ \underbrace{a_2\,(s_2, x_2, y_2)}_{Player(1)}\ \underbrace{a_3\,(s_3, x_3, y_3)}_{Player(2)} \ldots$$

The discussion given on page 42 shows that, for every t, (s_t, x_t, y_t) is a deterministic function of $a_1, \ldots, a_t$ so that, from a probabilistic point of view, an execution can equivalently be defined to be the sequence $a_1, a_2, \ldots$ This is the definition adopted in this chapter.

We have so far informally defined protocols and adversaries to be the strategies of *Player*(1) and *Player*(2), respectively. We now discuss how these notions can be formalized, beginning with the notion of adversary. Our construction is a direct application of the general construction given in Chapter 2.

We define an adversary to be a family of probability distributions $(Q_v)_{v \in V}$, one for each v in V: V is the set of all the possible views that *Player*(2) can have of the system at any time of the game i.e., at any time of the execution. (We will make this more explicit in Section 7.2.) For every element f, the quantity $Q_v(f)$ represents the probability that *Player*(2) chooses f if its view of the system is v. Note that, according to the general presentation made in Chapter 2, we should define an adversary to be a family of probability spaces $(\Omega_v, \mathcal{G}_v, P_v)_{v \in V}$. Nevertheless we can take all the measurable spaces $(\Omega_v, \mathcal{G}_v)$ to be equal to $(\{1, \ldots, N\}, 2^{\{1, \ldots, N\}})$. This allows us to omit mentioning $(\Omega_v, \mathcal{G}_v)$ in the definition of the adversary.

Note that a family $(f_v)_{v \in V}$, i.e., the choice of an element f_v for each view v in V, corresponds to a *decision tree* of *Player*(2).[3] As the number of rounds of a game is bounded and as, at each round, the number of different actions of both players is also bounded, the number of decision trees of *Player*(2) is similarly bounded. In this case it is easy to see that the set of strategies of *Player*(2), i.e., the set of families

[3] Actually a decision tree corresponds to a "weeded out" family $(f_v)_{v \in V'}$, where V' is a maximal subset of V having the property that each view v in V' is compatible with the choices f_w made previously by the player. Nevertheless, the extension to the set of all views is inconsequential and we adopt for simplification this characterization of a decision tree.

$(Q_v)_{v \in V}$, is in one-to-one correspondence with the set of probability distributions on decision trees.

We could therefore equivalently define an adversary to be a probability distribution on the set of decision trees (of $Player(2)$). We say that the definition in terms of a family $(Q_v)_{v \in V}$ adopts the *local point of view* whereas the definition in terms of a decision tree adopts the *global point of view*. Let us emphasize that the equivalence of these two points of view depends on the finiteness of the number of decision trees.[4]

Following the same model for $Player(1)$ we could define a protocol to be a family $(P_u)_{u \in U}$ of probability distributions, one for each possible view u that $Player(1)$ can have of the system at any time of the game.[5] Nevertheless, as by assumption the protocol receives no information from the adversary, we find it easier to adapt the global point of view: in this case a decision tree (of $Player(1)$) is simply a sequence $(s_1, \ldots, s_p)$ in $\mathcal{P}_n(p)$. We therefore define a protocol to be a probability distribution on $\mathcal{P}_n(p)$.

Note that the distinction between protocol and protocol-designer (resp. between adversary and adversary-designer) is often not kept and we refer to properties of the protocol (resp. the adversary) that should be more properly attributed to the protocol-designer (resp. the adversary-designer). A case where both points of views are equally valid is when we refer to the decisions done by the protocol or by the protocol-designer: the protocol is the strategy used by the protocol-designer for its decision making. By contrast, when speaking of "the knowledge held by the adversary" or of "the information received by the adversary" we should more correctly speak of the knowledge held by the adversary-designer: by definition, an adversary $\mathcal{A}$ is a family $(Q_v)_v$ of probability distributions which receives no information. On the other hand, $Player(2)$, the adversary-designer, does receive some information during an execution and uses this information as prescribed by its strategy $\mathcal{A}$.

---

[4] This duality is well known, but not everyone realizes the caveat about finiteness. For instance Hart et al. say in [29] and we quote: "*There are two main ways of introducing randomizations ... The first consists of a random choice of the next process at each node of the execution tree ... The second way consists of taking a probability distribution over the set of deterministic [executions] (i.e., make all the random decisions a priori.) ... It is easy to see that the first case (independent randomization at each decision node) is a special case of the second one (by doing all randomizations at the start!)*" Note though that, in the case of *infinite* executions, it is not trivial to convert "the first way" into "the second way". This is actually the heart of the problem in the construction of the probability distribution $P_{\mathcal{A}}$ given on page 41.

[5] As previously for the adversary, note that, according to the general presentation made in Chapter 2, we should actually define a protocol to be a family of probability spaces $(\Omega_u, \mathcal{G}_u, P_u)_{u \in U}$. Nevertheless we can take all the measurable spaces $(\Omega_u, \mathcal{G}_u)$ to be equal to $(\mathcal{P}_n(p), 2^{\mathcal{P}_n(p)\}})$. This allows us to omit mentioning $(\Omega_u, \mathcal{G}_u)$ in the definition of the adversary.

As is established in Chapter 2, a given strategy π of $Player(1)$ and a given strategy $\mathcal{A}$ of $Player(2)$ define a unique probability distribution $P_{\pi,\mathcal{A}}$ on the set Ω of executions. We will give a precise characterization of $P_{\pi,\mathcal{A}}$ in Section 7.2. We will also define there formally the random variable T representing the survival time. With these notions, the optimal expected survival time achievable against every adversary is

$$\sup_{\pi} \inf_{\mathcal{A}} E_{\pi,\mathcal{A}}[T] .$$

Note that we adopt here the Von Neumann setting described in Chapter 6: for each of the two players the set of pure strategies is the *finite* set of decision trees of that player. Hence, by Von Neumann's theorem, (see Theorem 8.3.2),

$$\sup_{\pi} \inf_{\mathcal{A}} E_{\pi,\mathcal{A}}[T] = \inf_{\mathcal{A}} \sup_{\pi} E_{\pi,\mathcal{A}}[T] ,$$

and the general proof methodology described in Chapter 6, Page 127, applies. Our proof will follow this methodology.

The rest of the chapter is organized as follows. In Section 7.2 we formalize the previous discussion and construct the probabilistic model used in the subsequent sections. In Section 7.3 we define for every π and $\mathcal{A}$ two pseudo-probability distributions P_π and $P_\mathcal{A}$ which play a crucial role in the proof. (The denomination "pseudo-probability distribution" refers to the fact that P_π and $P_\mathcal{A}$ are *not* probability distributions but that, as is asserted in Lemma 7.3.4, some "conditional" variants of them are well defined probability distributions.) Section 7.4 describes a class $Prot(Prog_0)$ of protocols π_0 and an adversary $\mathcal{A}_0$. The main result of this chapter which is presented in Theorem 7.4.6 asserts that π_0 and $\mathcal{A}_0$ verify point 2 of the methodology given in Page 127, and hence that every protocol $\pi_0 \in Prot(Prog_0)$ is optimal:

$$\sup_{\pi} E_{\pi,\mathcal{A}_0}[T] = E_{\pi_0,\mathcal{A}_0}[T] = \inf_{\mathcal{A}} E_{\pi_0,\mathcal{A}}[T] .$$

The proof of this theorem is the object of the rest of the chapter. Section 7.5 presents some random variables that are fundamental for the proof. Section 7.6 establishes that $\sup_\pi E_{\pi,\mathcal{A}_0}[T] = E_{\pi_0,\mathcal{A}_0}[T]$. Similarly, Section 7.7 establishes in essence that $\inf_\mathcal{A} E_{\pi_0,\mathcal{A}}[T] = E_{\pi_0,\mathcal{A}_0}[T]$.

7.2 The Probabilistic Model

We formalize here 1) the notions of *protocol* and *adversary* and 2) the probability spaces that will be used in the subsequent analyses. Recall that a protocol is a strategy of *Player*(1) and that, similarly, an adversary is a strategy of *Player*(2).

For every $t, 1 \leq t \leq p$, a sequence $\sigma = (s_1, s_2, \ldots, s_t)$ in $\mathcal{P}_n(p)^t$ is called a *t-schedule* and a sequence $\phi = (f_1, \ldots, f_p)$ of elements of $[p]$ is called a *t-fault-sequence*. A *t-fault-sequence* ϕ is *adapted* to a schedule σ if $f_j \in s_j$ for all $j, 1 \leq j \leq t$, and if, for all j, the condition $s_j \not\subseteq \{f_1, \ldots, f_{j-1}\}$ implies that $f_j \in s_j - \{f_1, \ldots, f_{j-1}\}$.

A *t-execution* is an alternating sequence $\omega = (s_1, f_1, s_2, f_2, \ldots, s_t, f_t)$ obtained from a t-schedule $(s_1, \ldots, s_t)$ and a t-fault-sequence $(f_1, \ldots, f_t)$ adapted to $(s_1, \ldots, s_t)$. A *t-odd execution* is a sequence $\omega = (s_1, f_1, s_2, f_2, \ldots, s_t)$ obtained from a t-schedule $(s_1, \ldots, s_t)$ and a $t-1$-fault-sequence $(f_1, \ldots, f_{t-1})$ adapted to $(s_1, \ldots, s_{t-1})$. For simplicity, we use the term execution in place of p-execution.

We define the sample space Ω to be the set of all executions: $\Omega \stackrel{\text{def}}{=} \{\omega;\ \omega \text{ execution}\}$. We endow Ω with its discrete σ-field $\mathcal{G} \stackrel{\text{def}}{=} 2^{\Omega}$.

We now define various random variable on $(\Omega, \mathcal{G})$. Throughout, random variables are denoted by upper-case and their realizations by lower-case. For all $t, 1 \leq t \leq p$, S_t and F_t are defined by $S_t(\omega) = s_t$, and $F_t(\omega) = f_t$. This allows us to define the derived random variables $\mathcal{S}_t = (S_1, \ldots, S_t)$, $\mathcal{F}_t = (F_1, \ldots, F_t)$ and $\mathcal{E}_t = (S_1, F_1, \ldots, F_{t-1}, S_t)$. $\mathcal{S}_t$, $\mathcal{F}_t$, and $\mathcal{E}_t$ are respectively the random t-schedule, the random t-fault sequence and the random t-odd-execution produced up to time t. We also let $\mathcal{G}_t \stackrel{\text{def}}{=} \sigma\left(S_1, F_1, \ldots, S_t, F_t\right)$ and $\mathcal{G}'_t \stackrel{\text{def}}{=} \sigma\left(S_1, F_1, \ldots, S_{t+1}\right)$ be the σ-fields of events "happening no later" then the selection of f_t and s_{t+1}, respectively.

We now define protocols and adversaries. As recalled in the previous section, the protocols and the adversaries are the strategies of *Player*(1) and *Player*(2) taking random steps in turn and sending information to the other player at the end of each step. The probability distribution used for this step depends on the *view* of the system held by the player. In our specific problem, *Player*(2) is informed of all the moves of *Player*(1), (but nothing else about the protocol π selected by *Player*(1)), whereas *Player*(1) learns nothing from and about *Player*(2). We adopt the local point of view to describe an adversary and the global point of view to describe a protocol.[6]

Hence, an *adversary* $\mathcal{A}$ is a family of probability distributions $(Q_v)_{v \in V}$ on $[p]$, one for each $t, 1 \leq t \leq p$ and each t-odd-execution $v = (s_1, f_1, \ldots, f_{t-1}, s_t)$ and such

[6] See page 139 for a definition of the local and of the global point of views.

that, with Q_v-probability one, $(f_1, \ldots, f_t)$ is adapted to $(s_1, \ldots, s_t)$.

A scheduling *protocol* π is a probability distribution on $\mathcal{P}_n(p)^p$.

As is established in Section 2.4, there is a unique and well defined probability distribution $P_{\pi,\mathcal{A}}$ induced on the set of executions by a given protocol π and a given adversary $\mathcal{A}$. For the sake of illustration, we give an explicit characterization of $P_{\pi,\mathcal{A}}$ in the next proposition.

Proposition 7.2.1 *Let π and $\mathcal{A} = (Q_v)_{v \in V}$ be a protocol and an adversary as defined above. Then there is a unique and well defined probability distribution $P_{\pi,\mathcal{A}}$ on $(\Omega, \mathcal{G})$ satisfying the following two properties:*

i $P_{\pi,\mathcal{A}}\Big[\mathcal{S}_p = \cdot\Big] = \pi$.

ii *For every $t, 1 \leq t \leq p$, every t-execution $v = (s_1, f_1, \ldots, f_{t-1}, s_t)$, and every $(p-t)$-schedule $(s_{t+1}, \ldots, s_p)$ such that $\pi(s_1, \ldots, s_p) > 0$ and $Q_{s_1}(f_1) Q_{s_1 f_1 s_2}(f_2)$ $Q_{s_1 f_1 s_2 \ldots s_{t-1}}(f_{t-1}) > 0$, we have:*

$$P_{\pi,\mathcal{A}}\Big[F_t = \cdot \mid \mathcal{E}_t = v,\ (S_{t+1}, \ldots, S_p) = (s_{t+1}, \ldots, s_p)\Big] = Q_v \ .$$

Property **i** formalizes the fact that the protocol receives no on-line information and makes its decisions in isolation. Property **ii** formalizes the fact that the adversary is on-line and selects F_t based on the sole knowledge of the past odd-execution $\mathcal{E}_t$, independently of the schedule $(S_{t+1}, \ldots, S_p)$ selected for subsequent times. Using Convention 8.1.1 we extend the definition of the conditional probability in **ii** and set $P_{\pi,\mathcal{A}}\Big[F_t = \cdot \;\Big|\; \mathcal{E}_t = v,\ (S_{t+1}, \ldots, S_p) = (s_{t+1}, \ldots, s_p)\Big] = 0$ whenever $Q_{s_1}(f_1) Q_{s_1 f_1 s_2}(f_2) Q_{s_1 f_1 s_2 \ldots s_{t-1}}(f_{t-1}) = 0$.

Proof. Let $\omega = (s_1, f_1, s_2, f_2, \ldots, s_p, f_p)$ be a generic execution in Ω and, for every t, let $\omega_t = (s_1, f_1, \ldots, s_t)$ be the associated t-odd-execution. By successive conditioning we can write

$$\begin{aligned} P_{\pi,\mathcal{A}}(\omega) \;=\; & P_{\pi,\mathcal{A}}\Big[\mathcal{S}_p = (s_1, \ldots, s_p)\Big]\ P_{\pi,\mathcal{A}}\Big[F_1 = f_1 \mid \mathcal{E}_1 = \omega_1, \mathcal{S}_{2,p} = (s_2, \ldots, s_p)\Big] \ldots \\ & \ldots P_{\pi,\mathcal{A}}\Big[F_{p-1} = f_{p-1} \mid \mathcal{E}_{p-1} = \omega_{p-1}, S_p = s_p\Big]\ \ P_{\pi,\mathcal{A}}\Big[F_p = f_p \;\Big|\; \mathcal{E}_p = \omega_p\Big] \ , \end{aligned}$$

so that we see that the conditions **i** and **ii** imply that, if it exists, $P_{\pi,\mathcal{A}}(\omega)$ must be equal to

$$P_{\pi,\mathcal{A}}(\omega) = \pi[(s_1, \ldots, s_p)] \prod_{t=1}^{p} Q_{\omega_t}(f_t) \ .$$

We easily check that $P_{\pi,\mathcal{A}}$ thus defined is additive and that $P_{\pi,\mathcal{A}}(\Omega) = 1$. As $\mathcal{G}$ is finite, the σ-additivity of $P_{\pi,\mathcal{A}}$ holds trivially. Hence $P_{\pi,\mathcal{A}}$ is a well defined probability measure. □

We have therefore defined the family of probability distributions $(P_{\pi,\mathcal{A}})_{\pi,\mathcal{A}}$ on the *same* space $(\Omega, \mathcal{G})$. In situations different from our scheduling problem, such a modeling is in general not possible and a different probability space $(\Omega_{\pi,\mathcal{A}}, \mathcal{G}_{\pi,\mathcal{A}})$ must be associated to each couple $(\pi, \mathcal{A})$. Also, in the case of infinite executions, the construction of a measure $P_{\pi,\mathcal{A}}$ is a non-trivial probability problem requiring the use of an extension theorem (e.g., Kolmogorov's extension theorem). We presented the general construction in Section 2.4 of Chapter 2 when the the coins of both players have at most countably many outcomes.

We can now set up formally the optimization problem presented in Section 7.1. The survival time is defined to be the random variable $T = \max\{t;\ \forall u \leq t, |S_u \cap \{F_1, \ldots, F_u\}| \leq m\}$. For every protocol π, let $t(\pi) = \inf_{\mathcal{A}} E_{\pi,\mathcal{A}}[T]$ be the worst case performance of π. We let

$$t_{\text{opt}} = \sup_{\pi} t(\pi) = \sup_{\pi} \inf_{\mathcal{A}} E_{\pi,\mathcal{A}}[T]\,.$$

7.3 Some Specific Probability Results

7.3.1 A formal interpretation of the on-line knowledge of the adversary

We begin this section by presenting a lemma expressing that, for every protocol and every adversary, conditioned on the past, the selection F_t made by the adversary at time t is independent of the choices $S_{t+1}, \ldots, S_p$ made for ensuing times by the protocol. This shows that the definitions of a protocol and of an adversary given in the previous section formalize accurately the on-line nature of the information received by the adversary: eventhough a protocol decides the whole schedule $s_1, \ldots, s_p$ at the beginning of the execution, the adversary does not get to see each decision s_t before time t. (As mentioned in Section 7.1.2, Page 139, a more correct statement is "the adversary-designer does not get to see each decision s_t before time t".)

Lemma 7.3.1 *Let π be a protocol, $\mathcal{A}$ be an adversary, $t, t \leq p-1$, be a time and v be a t-odd-execution such that $P_{\pi,\mathcal{A}}[\mathcal{E}_t = v] > 0$. Then the random variables F_t and $\mathcal{S}_{t+1,p}$ are independent with respect to the measure $P_{\pi,\mathcal{A}}$ when conditioned on $\mathcal{E}_t = v$.*

Proof. This is a direct consequence of Proposition 7.2.1: by Condition bf ii, for every $p-(t+1)$-schedule σ, for all v and σ,

$$P_{\pi,\mathcal{A}}\Big[\, F_t = \cdot \mid \mathcal{E}_t = v,\, \mathcal{S}_{t+1,p} = \sigma \,\Big] = P_{\pi,\mathcal{A}}\Big[\, F_t = \cdot \mid \mathcal{E}_t = v \,\Big] .$$

This expresses exactly the independence of F_t and $\mathcal{S}_{t+1,p}$ conditioned on $\mathcal{E}_t = v$. □

7.3.2 The Notations P_π and $P_\mathcal{A}$

The next two definitions Definition 7.3.1 and Definition 7.3.2 introduce the family of protocols that select with non-zero probability a given t-schedule, and the family of adversaries that select with non-zero probability a given t-fault sequence. The associated lemmas, Lemma 7.3.2 and Lemma 7.3.3 introduce and justify the notations P_π and $P_\mathcal{A}$. These two lemmas will be fundamental in our proofs for the following reasons. We will introduce one (family of) algorithm π_0 and an adversary $\mathcal{A}_0$. In one part of the proof we will establish that π_0 is optimal against $\mathcal{A}_0$. Throughout this part we will consider only the adversary $\mathcal{A}_0$. Lemma 7.3.2 will allow us to consider the unique expression $P_{\mathcal{A}_0}$ instead of the family $(P_{\pi,\mathcal{A}_0})_{\pi\in\Pi}$. This will make the analysis much simpler as the optimization over π will not involve the probability

measure. Symmetrically, in the second part of the proof we will in essence establish that $\mathcal{A}_0$ is optimal against π_0. In this part we will need to consider only the expression P_{π_0} instead of the family $(P_{\pi_0,\mathcal{A}})_{\mathcal{A}\in A}$.

Definition 7.3.1 *Let $(s_1,\ldots,s_t)$ be some t-schedule. Then*

$$Pgenerating(s_1,\ldots,s_t) \stackrel{\text{def}}{=} \Big\{\pi;\ \exists(s_{t+1},\ldots,s_p),\ \pi[(s_1,\ldots,s_p)]>0\Big\}\ .$$

As just mentioned, the next Lemma provides the fundamental technical tool that will allow us to handle conveniently all the probabilistic expressions required in the proof of Lemma 7.4.7 – the proof that π_0 is optimal against $\mathcal{A}_0$.[7] Similarly, Lemma 7.3.3 will provide the probabilistic tool required in the proof of Lemma 7.4.8 – the proof that $\mathcal{A}_0$ is optimal against π_0.

Lemma 7.3.2 *Let t, $1\le t\le N$ and let $\sigma=(s_1,\ldots,s_t)$ be a t-schedule. Let $\Phi\in\mathcal{G}_t$. Then, for every adversary $\mathcal{A}$,*

$$P_{\pi,\mathcal{A}}\left[\Phi \mid \mathcal{S}_t=\sigma\right]$$

is independent of the protocol $\pi\in Pgenerating(\sigma)$. We let

$$P_{\mathcal{A}}\left[\Phi \mid \mathcal{S}_t=\sigma\right]$$

denote this common value.

Following Convention 8.1.1, $P_{\pi,\mathcal{A}}\left[\Phi \mid \mathcal{S}_t=\sigma\right]$ is set to zero if $\pi\notin Pgenerating(\sigma)$.

PROOF. Let π be any protocol in $Pgenerating(\sigma)$ and let $\mathcal{A}=(Q_v)_{v\in V}$ be a given adversary. Recall that, by definition, (see page 141), $\mathcal{G}_t=\sigma(S_1,F_1,\ldots,S_t,F_t)$. Thus, an event Φ is in $\mathcal{G}_t$ if and only if there exists a boolean random variable ϕ depending on ω only through the random variables $(S_1,F_1,\ldots,S_t,F_t)$, (i.e., $\phi(\omega)=\psi(S_1(\omega),\ldots,F_t(\omega))$ for some real random variable ψ), such that $\Phi=\{\omega\,;\,\phi(\omega)=1\}$. We have:

$$\begin{aligned}&P_{\pi,\mathcal{A}}\Big[\Phi \mid \mathcal{S}_t=\sigma\Big]\\ &\quad= E_{\pi,\mathcal{A}}\Big[\phi \mid \mathcal{S}_t=\sigma\Big]\end{aligned}$$

[7] Both π_0 and $\mathcal{A}_0$ are defined in Section 7.4.

$$
\begin{aligned}
&= \Big(\sum_{f_1} P_{\pi,\mathcal{A}}\Big[F_1 = f_1 \mid \mathcal{S}_t = \sigma\Big] \sum_{f_2} P_{\pi,\mathcal{A}}\Big[F_2 = f_2 \mid \mathcal{S}_t = \sigma, F_1 = f_1\Big] \ldots \\
&\qquad \ldots \sum_{f_t} P_{\pi,\mathcal{A}}\Big[F_t = f_t \mid \mathcal{S}_t = \sigma,\ F_1 = f_1, \ldots, F_{t-1} = f_{t-1}\Big]\Big)\ \psi(s_1, f_1, \ldots, s_t, f_t) \\
&= \sum_{f_1} Q_{s_1}[f_1] \sum_{f_2} Q_{s_1,f_1,s_2}[f_2] \ldots \sum_{f_t} Q_{s_1,f_1,\ldots,s_t}[f_t]\ \psi(s_1, f_1, \ldots, s_t, f_t)\ .
\end{aligned}
$$

This last expression is independent of the protocol π, as needed. □

The following definition describes the set of adversaries that generate with non zero probability a given fault sequence. Note that, by Lemma 7.3.2, $P_{\mathcal{A}}[\mathcal{F}_t = \phi \,|\, \mathcal{S}_t = \sigma]$ is itself well-defined.

Definition 7.3.2 *Let σ be a t-schedule, and ϕ a t-fault sequence adapted to σ. Then*

$$
Agenerating_{\sigma}(\phi) \stackrel{\text{def}}{=} \Big\{\mathcal{A};\ P_{\mathcal{A}}[\mathcal{F}_t = \phi \ \mid\ \mathcal{S}_t = \sigma] > 0\Big\}\ .
$$

Lemma 7.3.3 *Let t, $1 \leq t \leq p$, let σ be a t-schedule, ϕ be a t-fault sequence adapted to σ and Let Φ be an event in $\mathcal{G}'_t$. Then, for every protocol $\pi \in Pgenerating(\sigma)$, the expression*

$$
P_{\pi,\mathcal{A}}\,[\Phi \mid \mathcal{S}_t = \sigma, \mathcal{F}_t = \phi]
$$

is independent of the adversary $\mathcal{A} \in Agenerating_{\sigma}(\phi)$. We let $P_{\pi}[\Phi \mid \mathcal{S}_t = \sigma, \mathcal{F}_t = \phi]$ denote this common value.

Proof. The proof closely follows that of Lemma 7.3.2. We write $\sigma = (s_1, \ldots, s_t)$ and $\phi = (f_1, \ldots, f_t)$. By assumption, there exists a boolean random variable ϕ depending on ω only through the random variables $(S_1, F_1, \ldots, S_t, F_t, S_{t+1})$, (i.e., $\phi(\omega) = \psi(S_1, F_1, \ldots, S_t, F_t, S_{t+1})(\omega)$) for some real random variable ψ), such that $\Phi = \{\omega\ ;\ \phi(\omega) = 1\}$. Let π be a protocol in $Pgenerating(\sigma)$ and $\mathcal{A} = (Q_v)_{v \in V}$ be an adversary in $Agenerating_{\sigma}(\phi)$. We let $P_{\pi}[S_{t+1} = s_{t+1} \mid \mathcal{S}_t = \sigma]$ denote the conditional probability $(\sum_{\alpha} \pi[(\sigma_t, s_{t+1}, \alpha)])/(\sum_{\alpha'} \pi[(\sigma_t, \alpha')])$ where the first summation is over all $p-(t+1)$-schedules α and the second over all $p-t$-schedules α'. Then:

$$
\begin{aligned}
&P_{\pi,\mathcal{A}}[\,\Phi \mid \mathcal{S}_t = \sigma, \mathcal{F}_t = \phi\,] \\
&\quad = E_{\pi,\mathcal{A}}[\,\phi \mid \mathcal{S}_t = \sigma, \mathcal{F}_t = \phi\,] \\
&\quad = \sum_{s_{t+1}} P_{\pi}[\,S_{t+1} = s_{t+1} \mid \mathcal{S}_t = \sigma\,]\ \psi(s_1, f_1, \ldots, s_t, f_t, s_{t+1})\ .
\end{aligned}
$$

This last expression is independent of the adversary $\mathcal{A}$ used and this concludes the proof. □

Lemma 7.3.4 *Let t be a time, $1 \leq t \leq p$. Let σ be a t-schedule and $\mathcal{A}$ be an adversary. Then $P_{\mathcal{A}}[\ \cdot \mid \mathcal{S}_t = \sigma]$ is a probability distribution on $(\Omega, \mathcal{G}_t)$. Similarly, if σ is a t-schedule, ϕ a t-fault sequence adapted to σ and π a protocol in $Pgenerating(\sigma)$, then $P_\pi[\ \cdot \mid \mathcal{S}_t = \sigma,\ \mathcal{F}_t = \phi]$ is a probability distribution on $(\Omega, \mathcal{G}'_t)$.*

PROOF. $P_{\mathcal{A}}[\ \cdot \mid \mathcal{S}_t = \sigma]$ is defined on $(\Omega, \mathcal{G}_t)$ to be equal to $P_{\pi,\mathcal{A}}[\ \cdot \mid \mathcal{S}_t = \sigma]$ for any $\pi \in Pgenerating(\sigma)$ and is therefore a well defined probability distribution (on $(\Omega, \mathcal{G}_t)$). Similarly, $P_\pi[\ \cdot \mid \mathcal{S}_t = \sigma,\ \mathcal{F}_t = \phi]$ is defined on $(\Omega, \mathcal{G}'_t)$ to be equal to $P_{\pi,\mathcal{A}}[\ \cdot \mid \mathcal{S}_t = \sigma,\ \mathcal{F}_t = \phi]$ for any $\mathcal{A} \in Agenerating_\sigma(\phi)$ and hence is a well defined probability distribution on $(\Omega, \mathcal{G}'_t)$. □

Note: In spite of its apparent simplicity, Lemma 7.3.2 answers a subtle point illustrating the difference between *implicit* and *explicit* knowledge that we quickly recall.[8]

In order to compute the optimal survival time $t_{\text{opt}} = \sup_\pi \inf_{\mathcal{A}} E_{\pi,\mathcal{A}}[T]$ we are led to consider the performance values $t(\pi) = \inf_{\mathcal{A}} E_{\pi,\mathcal{A}}[T]$ associated to all protocols π. In the previous formula the infimum is taken over all adversaries for a given π. A common interpretation of this fact is that the optimal adversary "knows" the protocol: this consideration entitled us to assume an off-line adversary in the deterministic case. Hence, such an adversary is provided with:

- the on-line information of the past schedule. At every time t, we can picture that an *explicit* message is relayed to the adversary to inform it of the set s_t last selected by the protocol.
- the off-line information of the protocol π that $\mathcal{A}$ is associated with: this information is *implicitly* provided to an optimal adversary.

These two notions of knowledge are very different, and Lemma 7.3.2 would *not* hold if we assumed that the adversary was provided with the explicit knowledge of π and was able to use this information in the selection of the elements $F_1, \ldots, F_p$. For instance, consider the case where $p = 3, n = 2$ and $m = 1$. Consider a greedy adversary $\mathcal{A}$, selecting at each time t a processor F_t so as to maximize the probability that F_t is in S_{t+1}. Assume that $s_1 = \{1, 2\}$. Consider two different protocols π_1 and π_2. Assume that protocol π_1 always selects $s_2 = \{1, 3\}$ whereas π_2 always selects $s_2 = \{2, 3\}$. If $\mathcal{A}$ knows the protocol it is associated with, $\mathcal{A}$ selects $F_1 = 1$ with probability one when associated with π_1, and selects $F_1 = 2$ with probability one

[8] These notions are presented in Page 207.

when associated with π_2. Hence, in this case, the probability

$$P_{\pi,\mathcal{A}}\left[F_1 = 1 \mid S_1 = \{1,2\}\right]$$

is not independent of the protocol π considered, even though the event $\{F_1 = 1\}$ is clearly in the σ-field $\mathcal{G}_1$.

Hence Lemma 7.3.2 would not be true, had we assumed, as in [25], that the adversary "knew" the protocol. Recall that, as is argued in Page 131, this change of model only affects the way the adversary is defined and interacts with the protocol. In particular, it does not affect the class of optimal protocols. Nevertheless, as Lemma 7.3.2 is crucial for the proofs given in Section 7.6, our proof of optimality would not carry over in the Strong Byzantine setting.

7.3.3 Applications of the definition of $P_{\mathcal{A}}$ and of P_π

Lemma 7.3.5 *For all j and t, $j \leq t$, all t-schedules σ and all adversaries $\mathcal{A}$, $P_{\mathcal{A}}[F_j \in \cdot \mid \mathcal{S}_t = \sigma]$ and $P_{\mathcal{A}}[T \geq t \mid \mathcal{S}_t = \sigma]$ are well defined probabilities. Similarly, if σ is a t-schedule in $\mathcal{N}_t$, ϕ a t-fault-sequence adapted to σ and π a protocol in Pgenerating(σ), then $P_\pi[\, T \geq t+1 \mid \mathcal{S}_t = \sigma, \mathcal{F}_t = \phi\,]$ is a well defined probability.*

Proof. The random variable F_j is clearly $\mathcal{G}_t$-measurable for all $j, 1 \leq j \leq t$ (by definition of $\mathcal{G}_t$!). Hence the first result is a simple application of Lemma 7.3.2. On the other hand we can write

$$\begin{aligned}\{T \geq t\} &= \bigcap_{j=1}^{t} \Big\{ |S_j \cap \{F_1, \ldots, F_j\}| \leq m \Big\} \\ &= \bigcap_{j=1}^{t} \Big\{ |S_j \cap \{F_1, \ldots, F_{j-1}\}| \leq m-1 \Big\}.\end{aligned}$$

For all j, the event $\{|S_j \cap \{F_1, \ldots, F_{j-1}\}| \leq m-1\}$ is clearly in $\mathcal{G}'_{j-1} \subseteq \mathcal{G}'_{t-1} \subseteq \mathcal{G}_t$. Hence $\{T \geq t\}$ is also in $\mathcal{G}_t$ and Lemma 7.3.2 again shows that $P_{\mathcal{A}}[T \geq t \mid \mathcal{S}_t = \sigma]$ is a well defined probability. Similarly $\{T \geq t+1\}$ is in $\mathcal{G}'_t$ so that, by Lemma 7.3.3, $P_\pi[\, T \geq t+1 \mid \mathcal{S}_t = \sigma, \mathcal{F}_t = \phi\,]$ is a well defined probability. □

The following lemma expresses that, conditioned on $\mathcal{S}_{t-1} = \sigma$, the events $S_t = s$ and $T \geq t-1$ are independent with respect to the measure $P_{\mathcal{A}}$ (i.e. with respect to any measure $P_{\pi,\mathcal{A}}$).

Lemma 7.3.6 *For every $t, 2 \leq t \leq N$, every $t-1$-schedule σ and every $s \in \mathcal{P}_n(p)$,*

$$P_{\mathcal{A}}[T \geq t-1 \mid \mathcal{S}_t = (\sigma, s)] = P_{\mathcal{A}}[T \geq t-1 \mid \mathcal{S}_{t-1} = \sigma] \ .$$

PROOF. Note first that, by Lemma 7.3.5, the quantities involved in the previous equality are well defined.

$$\begin{aligned} & P_{\mathcal{A}}\Big[T \geq t-1 \ \Big| \ \mathcal{S}_{t-1} = \sigma, S_t = s\Big] \\ &= P_{\mathcal{A}}\Big[\bigcap_{u=1}^{t-1} \Big\{S_u \cap \{F_1, \ldots, F_{u-1}\} = \emptyset\Big\} \ \Big| \ \mathcal{S}_{t-1} = \sigma, S_t = s \ \Big] \\ &= P_{\mathcal{A}}\Big[\bigcap_{u=1}^{t-1} \Big\{s_u \cap \{F_1, \ldots, F_{u-1}\} = \emptyset\Big\} \ \Big| \ \mathcal{S}_{t-1} = \sigma, S_t = s \ \Big]. \end{aligned}$$

By Lemma 7.3.1, conditioned on $\mathcal{S}_{t-1} = \sigma$, the random variables $(F_1, \ldots, F_{t-1})$ and S_t are independent. Hence, conditioned on $\mathcal{S}_{t-1} = \sigma$, the events $\cap_{u=1}^{t-1}\{s_u \cap \{F_1, \ldots, F_{u-1}\} = \emptyset\}$ and $\{S_t = s\}$ are similarly independent so that

$$\begin{aligned} & P_{\mathcal{A}}\Big[\bigcap_{u=1}^{t-1} \Big\{s_u \cap \{F_1, \ldots, F_{u-1}\} = \emptyset\Big\} \ \Big| \ \mathcal{S}_{t-1} = \sigma, S_t = s \ \Big] \\ &= P_{\mathcal{A}}\Big[\bigcap_{u=1}^{t-1} \Big\{s_u \cap \{F_1, \ldots, F_{u-1}\} = \emptyset\Big\} \ \Big| \ \mathcal{S}_{t-1} = \sigma \ \Big] \ . \end{aligned}$$

This establishes our claim. □

The following definition characterizes the schedules σ which allow the system to survive with non-zero probability.

Definition 7.3.3 *Let σ be a t-schedule such that $\sup_\pi P_{\pi,\mathcal{A}}\Big[\mathcal{S}_t = \sigma, T \geq t\Big] > 0$. We then say that σ is an $\mathcal{A}$-*feasible *t-schedule and we denote this by:*

$$\sigma \in \mathit{Feas}_{\mathcal{A}}.$$

Remarks:

1. Using Convention 8.1.1 we see that $\sigma \in \mathrm{Feas}_{\mathcal{A}}$ if and only if $\sup_\pi P_{\pi,\mathcal{A}}[T \geq t \mid \mathcal{S}_t = \sigma] > 0$ i.e., if $P_{\mathcal{A}}[T \geq t \mid \mathcal{S}_t = \sigma] > 0$.
2. We will provide in Corollary (7.6.2) a pure combinatorial characterization of $\mathrm{Feas}_{\mathcal{A}_0}$ for the adversary $\mathcal{A}_0$ defined in Definition 7.4.1, page 163.

7.4 Description of a Randomized Scheduling Algorithm

For the rest of this chapter we restrict ourselves to the case $m = 1$, i.e., when the system can sustain one fault but crashes as soon as at least two of the n processes are faulty. We provide a family of protocols and prove their optimality.

7.4.1 Description of the program

We formally defined a protocol to be a probability distribution on $\mathcal{P}_n(p)^p$. In place of a single protocol π_0 we present here a family $Prog_0$ of programs not only outputting random schedules $(S_1, \ldots, S_p)$ in $\mathcal{P}_n(p)^p$ but making also other, internal, random draws. For the sake of clarity we distinguish between a program and the protocol associated to it, i.e., between a program and the probability distribution of the random schedule $(S_1, \ldots, S_p)$ that it generates. For every program $prog$ in $Prog_0$ we let π_{prog} denote the associated protocol. We also let $Prot(Prog_0)$ denote the family of protocols derived from $Prog_0$:

$$Prot(Prog_0) = \{\pi_{\text{prog}};\ prog \in Prog_0\}.$$

In the code describing $Prog_0$ we use statements of the kind $X :=$ `uniform`$(a; A)$ and $X :=$ `arbitrary`$(a; A)$. We call these statements *randomized invocations*. For every set A and integer $a, a \leq |A|$, the statement $X :=$ `uniform`$(a; A)$ means that the set X is chosen uniformly at random from $\mathcal{P}_a(A)$ i.e., from among all subsets of A of size a. Similarly, for every set A and integer $a, a \leq |A|$, the statement $X :=$ `arbitrary`$(a; A)$ means that the set X is chosen at random – but not necessarily uniformly – from $\mathcal{P}_a(A)$. The probability distribution used for this random selection is *arbitrary* and depends on the values returned on the past previous randomized invocations done by the program. This means that, for every t, the probability distribution used for the $t+1$-st invocation can be written $P_{r_1,\ldots,r_t}$, where $r_1, \ldots, r_t$ are the t values returned by the first t randomized invocations.

$Prog_0$ represents a family of programs, one for each choice of the probability distributions $P_{r_1,\ldots,r_t}$ used at all the randomized invocations. (Recall though, by definition, if the $t+1$-st randomized invocation is $X :=$ `uniform`$(a; A)$ then $P_{r_1,\ldots,r_t} \equiv \mathcal{U}_{\mathcal{P}_a(A)}$ for all $r_1, \ldots, r_t$.) We will not make these choices explicit and will show that all programs $prog$ in $Prog_0$ are optimal.

We present the code describing $Prog_0$ in the next figure and provide explanations after it.

$Prog_0$

Variables:

$C_0 \subseteq [p]$; initially $[p]$
$C_j \subseteq [p],\ j = 1, \ldots, p$; initially arbitrary
$S_j \in \mathcal{P}_n(p),\ j = 1, \ldots, p$; initially arbitrary
$S \in \mathcal{P}_n(p)$; initially arbitrary
$I, I', J \subseteq [p]$; initially arbitrary
$K \in [p]$; initially arbitrary
$\alpha \in \mathbb{N}$; initially n

Code:

01. **for** $t = 1, \ldots, \lfloor p/n \rfloor$ **do** :
02. $S_t := \texttt{arbitrary}(n; C_0)$
03. $C_0 := C_0 - S_t$
04. $C_t := S_t$

05. **for** $t = \lfloor p/n \rfloor + 1, \ldots, p - n + 1$ **do** :
06. **if** $t = \lfloor p/n \rfloor + 1$ **then** :
07. $S_t := C_0$
08. $C_0 := \emptyset$
09. **else** $S_t := \emptyset$

10. **if** $\lfloor \frac{p-n}{t-1} \rfloor < \alpha$ **then** :
11. $\alpha := \lfloor \frac{p-n}{t-1} \rfloor$
12. $I := \texttt{arbitrary}(p - n - \lfloor \frac{p-n}{t-1} \rfloor (t-1); [t-1])$
13. $J := I$; $I' := [t-1] - I$
14. **while** $I \neq \emptyset$ **do** :
15. $K := \texttt{arbitrary}(1; I)$
16. $S := \texttt{uniform}(\alpha + 1; C_K)$
17. $S_t := S_t \cup (C_K - S)$
18. $C_K := S$
19. $I := I - \{K\}$
20. **while** $I' \neq \emptyset$ **do** :
21. $K := \texttt{arbitrary}(1; I')$
22. $S := \texttt{uniform}(\alpha; C_K)$
23. $S_t := S_t \cup (C_K - S)$

24. $C_K := S$
25. $I' := I' - \{K\}$
26. $C_t := S_t$

27. **if** $\lfloor \frac{p-n}{t-1} \rfloor = \alpha$ **then** :
28. $S :=$ `uniform`$(\alpha + 1; C_{t-1})$
29. $S_t := C_{t-1} - S$
30. $C_{t-1} := S$
31. $I' :=$ `arbitrary`$(\alpha + 1; J \cup \{t-1\})$
32. $J := (J \cup \{t-1\}) - I'$
33. **while** $I' \neq \emptyset$ **do** :
34. $K :=$ `arbitrary`$(1; I')$
35. $S :=$ `uniform`$(\alpha; C_K)$
36. $S_t := S_t \cup (C_K - S)$
37. $C_K := S$
38. $I' := I' - \{K\}$
39. $C_t := S_t$

7.4.2 A presentation of the ideas underlying $Prog_0$

We begin by presenting the purpose of the program variables used in $Prog_0$. At the end of each round t:

1. S_t is the set of n elements selected by the protocol for round t.
2. C_0 represents the set of elements of $[p]$ that have not been used in the first t rounds.
3. For every $j, 1 \leq j \leq t$, C_j is the set of elements selected at time j –i.e., elements in S_j – and which have never been used in later rounds $j+1, j+2, \ldots, t$ – i.e., which are in the complement $(S_{j+1} \cup \ldots \cup S_t)^c$ of $S_{j+1} \cup \ldots \cup S_t$.

For reasons presented below, at the end of each round $t, t \geq \lfloor \frac{p}{n} \rfloor + 1$, and for every $j, 1 \leq j \leq t-1$, C_j is either of size $\lfloor \frac{p-n}{t-1} \rfloor$ or of size $\lfloor \frac{p-n}{t-1} \rfloor + 1$. To achieve this, at each round $t, t \geq \lfloor \frac{p}{n} \rfloor + 1$, the program variable α is re-initialized to this value $\lfloor \frac{p-n}{t-1} \rfloor$. The program variables I, I' and J are used to distinguish and manipulate the two sets $\{j \in [1, t-1]; |C_j| = \alpha\}$ and $\{j \in [1, t-1]; |C_j| = \alpha + 1\}$. The program variable K is used to make some non-deterministic choices in the course of the execution.

The following explanations will explain the code and bring some intuition behind the choice of these numbers.

The idea of the code is to select for each round t the set S_t in a greedy fashion so as to optimize the probability of surviving one more round, if faulty processors are selected uniformly at random. For each $t \leq \lfloor \frac{p}{n} \rfloor$, the set C_0 of fresh elements has size at least n at the beginning of round t and it is therefore possible to select S_t as a subset of C_0. This is accomplished in lines 01 through 04 of the code. As only so far unselected elements are selected at each round t, we have $C_j = S_j$ for every $1 \leq j \leq t \leq \lfloor \frac{p}{n} \rfloor$.

In round $\lfloor \frac{p}{n} \rfloor + 1$ the set C_0 is possibly non-empty at the beginning of the round, but holds less then n elements. We select all its elements and allocate them to $S_{\lfloor \frac{p}{n} \rfloor + 1}$. This is done in lines 06 through 08 of the code. From that point on, i.e., for the completion of the selection of the set $S_{\lfloor \frac{p}{n} \rfloor + 1}$ as well as for the selection of later sets S_t, we have to select elements that have been selected previously at least once. We adopt the following simple strategy: we select elements from the sets $C_j, 1 \leq j \leq t$, that have the biggest size until n elements have been selected. For every $j, 1 \leq j \leq t$, by definition of C_j – as the set of elements selected in round j but never selected afterwards – every element of C_j selected in round t must be removed from C_j during this same round. Hence the strategy consists in *transferring* into S_t n elements from the sets C_j that have the biggest size. Once this transfer is accomplished S_t is of size n and we initialize C_t to be equal to S_t.

At the point in round $\lfloor \frac{p}{n} \rfloor + 1$ when C_0 becomes finally empty, (in line 08), and when the transfer strategy begins to be implemented, the sets $C_1, \ldots, C_{\lfloor \frac{p}{n} \rfloor}$ are all of size n. We can picture transferring elements away from sets C_j of biggest size as the selective flow of resources away from big reservoirs: by doing so we maintain to parity the level of all the reservoirs. In our case, as we are transferring discrete elements in place of a fluid, the transfer strategy keeps the size of the sets C_j different by at most one. Another consequence of the fact that elements are transferred from the sets C_j into S_t is that, at the end of each round $t, t \geq \lfloor \frac{p}{n} \rfloor + 1$, the sets $C_1, \ldots, C_{t-1}, S_t$ are a partition of the set $[p]$ of all elements. Hence we are in a situation where p elements are partitioned into t sets: on the one hand, a set S_t of n elements and, on the other hand, $t-1$ sets whose size differ by at most one. A computation (see Lemma 7.4.3) shows that these sets must be either of size $\lfloor \frac{p-n}{t-1} \rfloor$ or of size $\lfloor \frac{p-n}{t-1} \rfloor + 1$ and that the number of sets of size $\lfloor \frac{p-n}{t-1} \rfloor + 1$ is $p - n - \lfloor \frac{p-n}{t-1} \rfloor (t-1)$. The idea in the code is to use these numbers and modify for every round t the sets $C_1, \ldots, C_{t-2}, S_{t-1}$ determined at the end of each round $t-1$ to produce the partition $C_1, \ldots, C_{t-1}, S_t$ that must result in round t.

By our previous argument, at the end of each round $t-1$, the sets $C_1, \ldots, C_{t-2}$ are all of size at least $\lfloor \frac{p-n}{t-2} \rfloor$ and $S_{t-1} = C_{t-1}$ is of size n. At the end of each round t, these $t-1$ sets must be modified and reduced to size $\lfloor \frac{p-n}{t-1} \rfloor$ or $\lfloor \frac{p-n}{t-1} \rfloor + 1$. We need to distinguish two cases according to whether $\lfloor \frac{p-n}{t-1} \rfloor$ is less or equal to $\lfloor \frac{p-n}{t-2} \rfloor$. In the first case the code branches according to line 10, and lines 11 through 26 are followed. In the second case the code branches according to line 27, and lines 28 through 39 are followed.

Consider the case where $\lfloor \frac{p-n}{t-1} \rfloor < \lfloor \frac{p-n}{t-2} \rfloor$. In this case, every set $C_j, 1 \leq j \leq t-1$, existing at the end of round $t-1$ is of size at least $\lfloor \frac{p-n}{t-2} \rfloor \geq \lfloor \frac{p-n}{t-1} \rfloor + 1$ and hence is big enough to be reduced to either one of the two allowable sizes ($\lfloor \frac{p-n}{t-1} \rfloor$ or $\lfloor \frac{p-n}{t-1} \rfloor + 1$) at the end of round t. Consider now the case where $\lfloor \frac{p-n}{t-1} \rfloor = \lfloor \frac{p-n}{t-2} \rfloor$. In this case the sets $C_j, 1 \leq j \leq t-1$, which are of the smaller size $\lfloor \frac{p-n}{t-2} \rfloor$ at the end of round $t-1$ cannot be reduced to the size $\lfloor \frac{p-n}{t-1} \rfloor + 1$ ($= \lfloor \frac{p-n}{t-2} \rfloor + 1$) in the next round: only sets being of the bigger size $\lfloor \frac{p-n}{t-2} \rfloor + 1$ at the end of round $t-1$ can give rise to sets of the bigger size $\lfloor \frac{p-n}{t-1} \rfloor + 1$ at the end of round t.

In the first case, we branch according to line 10 and select in line 12 an arbitrary subset I of $\{1, \ldots, t-1\}$ of size $p - n - \lfloor \frac{p-n}{t-1} \rfloor (t-1)$: this set is the set of indices describing which of the sets $C_1, \ldots, C_{t-1}$ will be the bigger sets at the end of round t. We keep in the variable J a record of this selection (see line 13). Then, in lines 15 through 19 and 21 through 25 we transfer elements from the sets C_j into S_t, leaving the sets $C_j, 1 \leq j \leq t-1$, in their pre-decided size. We finally initialize C_t to the value S_t once the selection of S_t is finished (see line 26). Let us emphasize that, at the end of the round, J records the identity of the bigger sets C_j.

In the second case we branch according to line 27. In this case, at the beginning of round t, the sets $C_1, \ldots, C_{t-2}$ are all of one of the two sizes $\lfloor \frac{p-n}{t-1} \rfloor + 1$ and $\lfloor \frac{p-n}{t-1} \rfloor$, and J records the identity of the bigger sets $C_j, 1 \leq j \leq t-2$. Also, at this point, C_{t-1} is of size n. As all the sets $C_1, \ldots, C_{t-1}$ must be reduced to size at most $\lfloor \frac{p-n}{t-1} \rfloor + 1$, we first transfer $n - (\lfloor \frac{p-n}{t-1} \rfloor + 1)$ elements from C_{t-1} to S_t (see lines 28 through 30). We finish the selection of S_t by selecting one element from each of $\lfloor \frac{p-n}{t-1} \rfloor + 1$ sets arbitrarily selected from $J \cup \{t-1\}$. (The selection of the $\lfloor \frac{p-n}{t-1} \rfloor + 1$ sets is done in line 31. The transfer of the elements is done in lines 34 through 38.) As in the previous case, we update J so as to record the identity of the bigger sets C_j at the end of the round (see line 32) and initialize C_t with the value S_t (see line 39).

We quickly discuss the choice of the probability distributions used in $Prog_0$. As mentioned at the very beginning of our explanations, the idea of the code is to select for each round t the set S_t in a greedy fashion so as to optimize the probability of surviving one more round, if faulty processors are selected uniformly at random. Let us call *random adversary* the adversary that selects the faulty processors in this fashion. The main idea of the proof will be to prove that any π in $Prog_0$ is optimal against the random adversary and then to prove that, against any such π, no adversary can do better then the random adversary. As we will see, we could replace all the `uniform` randomized invocations by `arbitrary` randomized invocations and still obtain optimal protocols against the random adversary. The reason is, (as we will see), that for every time t, the fact that the system is still alive after the occurrence of the t-th fault means *exactly* that, for every $j, j \leq t-1$, the j-th fault is in the set C_j. (This is where the condition $m = 1$ plays its role.) Furthermore, if the system is still alive after the occurrence of the t-th fault and if the adversary is the random adversary, all the elements of C_j are equally likely to be faulty. This is due to the nature of the random adversary which, by definition, makes its selections irrespectively of the identity of the elements chosen by the protocol. Hence, for every N, if in round $t+1$ the protocol is to select a given number N of elements from one of the sets C_j, all the $\binom{|C_j|}{N}$ choices are equally as good, as any N elements of C_j have the same probability of all being not faulty.

But this does not hold for a general adversary. In effect, a general adversary can differentiate between different elements of a given C_j when the protocol uses arbitrary distributions to make such selections. We show that the strategy according to which, whenever selecting elements from a given set C_j, the protocol always uses the *uniform distribution*, disallows the adversary such capabilities of differentiation. This means that the use of `uniform` randomized invocations reduces the power of any adversary to the one of the random adversary.

We use `arbitrary` randomized invocations to emphasize that for the other randomized invocations made by the protocol, the choice of the distribution is irrelevant for the effectiveness of the protocol. For the simplicity of the exposition we let the protocol make these choices. But we could easily extend our results and prove that the programs in $Prog_0$ would perform equally well if these choices were made by the adversary.

7.4.3 Invariants

For $t = 1, \ldots, \lfloor p/n \rfloor$ and for every execution of a program in $Prog_0$, we define round t to be the part of the execution corresponding to the t-th loop, between line 02 and line 04 of the program. Similarly, for $t = \lfloor p/n \rfloor, \ldots, p - n + 1$ and for every execution of a program in $Prog_0$, we define round t to be the part of the execution corresponding to the t-th loop, between line 06 and line 39 of the program.

Lemma 7.4.1 *$Prog_0$ satisfies the following invariants and properties valid at the end of every round $t, 1 \leq t \leq n - p + 1$.*

a. *All invocations in round t of the commands* `arbitrary` *or* `uniform` *are licit, i.e., $a \leq |A|$ for every invocation of* `arbitrary`$(a; A)$ *and of* `uniform`$(a; A)$.

b. *α is equal to n if $t \leq \lfloor \frac{p}{n} \rfloor$ and equal to $\lfloor \frac{p-n}{t-1} \rfloor$ if $\lfloor \frac{p}{n} \rfloor + 1 \leq t \leq p - n + 1$.*

c. *$|S_t| = n$.*

d. *For every $j, 1 \leq j \leq t$, $C_j = S_j \cap (S_{j+1} \cup \ldots \cup S_t)^c$. In particular $C_t = S_t$.*

e. *$C_0 = (\cup_{j=1}^{t} S_j)^c = (\cup_{j=1}^{t} C_j)^c$.*

f. *$C_0, C_1, \ldots, C_t$ form a partition of $[p]$.*

g. *If $1 \leq t \leq \lfloor \frac{p}{n} \rfloor$ then $|C_1| = \ldots = |C_t| = n$.*

h. *If $\lfloor \frac{p}{n} \rfloor + 1 \leq t \leq p - n + 1$ then*

1. *$C_0 = \emptyset$*

2-i. *For every j, $1 \leq j \leq t - 1$, $|C_j|$ is equal to $\lfloor \frac{p-n}{t-1} \rfloor$ or $\lfloor \frac{p-n}{t-1} \rfloor + 1$*

2-ii. *There exists j, $1 \leq j \leq t - 1$, such that $|C_j| = \lfloor \frac{p-n}{t-1} \rfloor$.*

3. *$|C_t|$ is equal to n.*

i. *If $\lfloor \frac{p}{n} \rfloor + 1 \leq t \leq p - n + 1$ then $J = \{i \in [t-1]; |C_i| = \alpha + 1\}$ and $|J| = p - n - \lfloor \frac{p-n}{t-1} \rfloor (t-1)$.*

PROOF. For every program variable X, we let $X(t)$ denote the value held by X at the end of round t. We extend this definition to $t = 0$ and let $X(0)$ denote the initial value of any program variable X.

We easily see that, for every t, the program variable S_t is changed only in round t. Hence $S_t(t) = S_t(t+1) = \ldots = S_t(n - p + 1)$ so that we can abuse language and let

S_t also denote the value $S_t(t)$ held by the program variable S_t at the end of round t. (We will make explicit when S_t refers to the program variable and not to the value $S_t(t)$.)

Invariant b. For every round $t, 1 \leq t \leq \lfloor p/n \rfloor$, the program variable α is left unchanged: $\alpha(t) = \alpha(0) = n$. For every $t, \lfloor p/n \rfloor + 1 \leq t \leq p - n + 1$, the program variable α is left unchanged if equal to $\lfloor \frac{p-n}{t-1} \rfloor$ and reset to this value otherwise. (See lines 10, 11 and 27). Hence $\alpha(t) = \lfloor \frac{p-n}{t-1} \rfloor$. This establishes invariant **b**. We prove the other invariants by induction on the round number t.

Case A. Consider first the case $0 \leq t \leq \lfloor \frac{p}{n} \rfloor$. In this case we furthermore establish that $S_1, \ldots, S_t$ are all disjoint, that $C_j(t) = S_j$ for every $j, 1 \leq j \leq t \leq \lfloor \frac{p}{n} \rfloor$ and that $C_0(t)$ is a set of size $p - tn$. This is true for $t = 0$ as $C_0(0) = [p]$ and as, in this case, the family $S_1, \ldots, S_t$ is empty. Assume that all the invariants are true for some round $t-1$, $0 \leq t-1 < \lfloor \frac{p}{n} \rfloor$. Consider round t. In line 02 of the program, C_0 has value $C_0(t-1)$, which is of size $p-(t-1)n$, by induction. As $t-1 < \lfloor \frac{p}{n} \rfloor$, $p-(t-1)n \geq n$ and hence the invocation $S_t := \texttt{arbitrary}(n; C_0)$ is licit: invariant **a** is satisfied for round t. Hence S_t is a well-defined set of size n and invariant **c** is satisfied for round t. As S_t is a subset of $C_0(t)$, invariant **e** (for $t-1$) shows that S_t is disjoint from $\cup_i^{t-1} S_i$ so that $S_1, \ldots, S_t$ are all disjoint. In lines 02–04 the program variables $C_j, 1 \leq j \leq t-1$ are not changed. Hence $C_j(t) = C_j(t-1) = S_j$, $1 \leq j \leq t-1$. On the other hand, by line 04, $C_t(t) = S_t$, and by line 03, $C_0(t-1)$ is the disjoint union of $C_0(t)$ and of S_t. Hence $|C_0(t)| = |C_0(t-1)| - |S_t| = (p-(t-1)n) - n = p - tn$. From these properties we easily check that the invariants **d**, **e**, **f** and **g** are true for t.

Case B. We now turn to the case $\lfloor p/n \rfloor + 1 \leq t \leq p - n + 1$. Assume that t is such an integer and such that all the invariants are true for $t-1$.

Case B-I. Assume $\lfloor \frac{p-n}{t-1} \rfloor < \alpha$ in line 10 is true.

We first establish that $t = \lfloor p/n \rfloor + 1$ falls in case B-I. We just proved that all the invariants **a**, ..., **g** hold for $t - 1 = \lfloor p/n \rfloor$. By Lemma 7.4.2, $\lfloor \frac{p-n}{t-1} \rfloor < n$. On the other hand, in line 10, the program variable α has value $\alpha(t-1) = \alpha(0) = n$. Hence the precondition $\lfloor \frac{p-n}{t-1} \rfloor < \alpha$ in line 12 is true for $t = \lfloor p/n \rfloor + 1$.

Invariant a. We easily check that for every t, $0 \leq p - n - \lfloor \frac{p-n}{t-1} \rfloor (t-1) < t-1$. (For every numbers x and y, $x - \lfloor x/y \rfloor y$ is the rest of the Euclidean division of x

by y.) Hence the invocation $I :=$ `arbitrary`$(p - n - \lfloor \frac{p-n}{t-1} \rfloor (t-1); [t-1])$ of line 12 is always licit (for $t = \lfloor p/n \rfloor + 1, \ldots, p - n + 1$).

In line 17 and in line 23 the program variable C_K has value $C_K(t-1)$, a set of size $\alpha(t-1)$ or $\alpha(t-1)+1$ by invariant **b** and **h** for $t-1$. Recall that, by assumption, the precondition of line 10 is true: $\alpha(t) < \alpha(t-1)$ i.e., $\alpha(t)+1 \leq \alpha(t-1)$. This implies that the invocations $S :=$ `uniform`$(\alpha+1; C_K)$ and $S :=$ `uniform`$(\alpha; C_K)$ of respectively lines 16 and 22 are licit. This shows that the invariant **a** is true for t.

Invariant h-1. The variable C_0 is set to $\emptyset$ in round $\lfloor p/n \rfloor + 1$ and never altered afterwards so that $C_0(t) = \emptyset$ for $t = \lfloor p/n \rfloor + 1, \ldots, p - n + 1$. This establishes invariant **h-1**.

Invariants d, **e** and **f**. By assumption, (invariant **f** for $t-1$), $C_0(t-1), \ldots, C_{t-1}(t-1)$ is a partition of $[p]$. The set S_t is obtained by first taking all the elements of $C_0(t-1)$, (which is non-empty only if $t = \lfloor p/n \rfloor + 1$), and then, in lines 16, 17, 18 and 22, 23, 24, transferring some subsets of $C_0(t-1), \ldots, C_{t-1}(t-1)$ into S_t. Hence, by construction, the sets $C_1(t), \ldots, C_{t-1}(t)$ and S_t are a partition of $[p]$ i.e., invariant **f** is true. This implies also that $C_j(t) = C_j(t-1) - S_t = C_j(t-1) \cap S_t^c$ for every $j, 1 \leq j \leq t-1$. By invariant **d** for $t-1$, we have $C_j(t-1) = S_j \cap (S_1 \cup \ldots \cup S_{t-1})^c$. Hence $C_j(t) = S_j \cap (S_1 \cup \ldots \cup S_t)^c$ for every $j, 1 \leq j \leq t-1$. Furthermore, by line 26, $C_t(t)$ is equal to S_t. Hence invariant **d** holds for t. We also easily deduce **e**.

Invariants i and **h-2**. By lines 16, 18 and 22, 24, for every $j, 1 \leq j \leq t-1$, the quantity $|C_j(t)|$ is equal to $\alpha(t)$ or $\alpha(t)+1$. This along with invariant **b** already proven shows that invariant **i** is true for t. Furthermore, the set of indices $j, 1 \leq j \leq t-1$, such that $C_j(t)$ is of size $\alpha(t)+1$ is the set I determined in line 12. (See lines 12 through 19 of the protocol.) This set is equal to the value allocated to J in line 13. As J is not further changed in round t this value is $J(t)$. This establishes invariant **h-2-i**. As mentioned in the proof of invariant **a**, for every t, $p - n - \lfloor \frac{p-n}{t-1} \rfloor (t-1) < t-1$. Hence the value allocated to I in line 12 is not the whole set $[t-1]$. Consequently, at the end of line 13 I' is not equal to $\emptyset$ and for every $k \in I'$ the **while** loop from line 20 to 25 produces a set $C_k(t)$ of size $\alpha(t)$, i.e., by invariant **b**, a set of size $\lfloor \frac{p-n}{t-1} \rfloor$. This establishes invariant **h-2-ii**.

Invariants c and **h-3**. Let b denote $p - n - \lfloor \frac{p-n}{t-1} \rfloor (t-1)$ and let a denote $t-1-b$. Note that $a > 0$. We just established that the family $C_1(t), \ldots, C_{t-1}(t), S_t$ is a partition of $[p]$. Therefore, $C_1(t), \ldots, C_{t-1}(t)$ is a partition of $[p] - S_t$. By invariants **h** and **i** this partition is composed of b elements of size $\alpha(t)+1$ and of a elements of size $\alpha(t)$. Hence $p - |S_t| = q = a\alpha(t) + b(\alpha(t)+1)$. On the other hand, by Lemma 7.4.3, $p - n = a\alpha(t) + b(\alpha(t)+1)$. This shows that $|S_t| = n$ and hence, by line 26, that $|C_t(t)| = n$.

Case B-II. Assume $\lfloor \frac{p-n}{t-1} \rfloor < \alpha$ in line 10 is false (i.e., $\lfloor \frac{p-n}{t-1} \rfloor = \alpha$ in line 27 is true).

Invariant a. In line 28, the value of the program variable C_{t-1} is $C_{t-1}(t-1)$ which is of size n by invariant **c** and **d** for $t-1$. As $t \geq \lfloor p/n \rfloor + 1$, by Lemma 7.4.2, $\lfloor \frac{p-n}{t-1} \rfloor + 1 \leq n$ which shows that the invocation $S := \texttt{uniform}(\alpha + 1; C_{t-1})$ is licit.

Note that, in line 27, α has value $\alpha(t-1)$ so that the condition $\lfloor \frac{p-n}{t-1} \rfloor = \alpha$ of line 27 exactly means that $\lfloor \frac{p-n}{t-1} \rfloor = \alpha(t-1)$. By invariant **b**, $\alpha(t-1)$ is either n or $\lfloor \frac{p-n}{t-2} \rfloor$. We prove that only the latter form arises in the equality $\lfloor \frac{p-n}{t-1} \rfloor = \alpha(t-1)$. Recall the two following facts established in the proof of invariant **b** given in page 157. 1) $\lfloor \frac{p-n}{\lfloor p/n \rfloor} \rfloor < n$. 2) $\alpha(\lfloor p/n \rfloor) = n$. From these two facts we deduce that the equality $\lfloor \frac{p-n}{t-1} \rfloor = \alpha(t-1)$ does not hold for $t = \lfloor p/n \rfloor + 1$ as $\lfloor \frac{p-n}{\lfloor p/n \rfloor} \rfloor < n = \alpha(\lfloor p/n \rfloor)$. Hence equality in line 27 can occur only for $t > \lfloor p/n \rfloor + 1$. By Lemma 7.4.2, $t > \lfloor p/n \rfloor + 1$ implies that $\lfloor \frac{p-n}{t-1} \rfloor < n$ and hence that $\alpha(t-1) = \lfloor \frac{p-n}{t-2} \rfloor$. To summarize: equality holds in line 27 only for $t > \lfloor p/n \rfloor + 1$ and then implies that $\lfloor \frac{p-n}{t-1} \rfloor = \lfloor \frac{p-n}{t-2} \rfloor$.

In line 31, the value of the program variable J is $J(t-1)$. By induction, (invariant **i** for $t-1$), $J(t-1)$ is a subset of $[t-2]$ which is of size $|J(t-1)| = p - n - \lfloor \frac{p-n}{t-2} \rfloor (t-2)$. The element $t-1$ is not in $J(t-1)$ (see invariant **i** for $t-1$) so that $|J(t-1) \cup \{t-1\}| = |J(t-1)| + 1$. Recall that in line 31, the program variable α is equal to $\alpha(t) = \lfloor \frac{p-n}{t-1} \rfloor$. We have:

$$\begin{aligned} \alpha(t) + 1 &= \lfloor \tfrac{p-n}{t-1} \rfloor + 1 \\ &\leq p - n - \lfloor \tfrac{p-n}{t-2} \rfloor (t-2) + 1 \quad \text{(by Lemma 7.4.4)} \\ &= |J(t-1)| + 1 \\ &= |J(t-1) \cup \{t-1\}| \,. \end{aligned}$$

This shows that the invocation $I' := \texttt{arbitrary}(\alpha + 1; J \cup \{t-1\})$ of line 31 is licit.

Line 34 is within the **while**-loop originated in line 33. Each of the invocation $K := \texttt{arbitrary}(1; J)$ of line 34 is licit as occurring while J is non-empty.

In line 35, C_K is of size $\alpha(t-1)+1$ because, by lines 31 and 34, K is an element of $J(t-1) \cup \{t-1\}$ and because, by invariant **i**, $K \in J(t-1)$ implies that $|C_K(t-1)| = \alpha(t-1)+1$ which is $\alpha(t)+1$ by line 27 – if $K = t-1$, lines 28 and 30 show directly that $|C_K| = \alpha + 1$. This shows that the invocation $S := \texttt{uniform}(\alpha; C_K)$ in line 35 is licit.

This finishes to establish that invariant **a** is true for t.

Invariants c and **h-3**. In lines 28, 29, $|C_{t-1}| - (\alpha(t-1)+1) = n - (\alpha(t-1)+1)$ elements are allocated to S_t. Let I'_{init} denote the value held by I' at the end of line 31. In the **while**-loop (line 33 to 38), $\alpha(t-1)+1$ additional elements are allocated to S_t. ($\alpha(t-1)+1$ is the size of I'_{init}.) Hence a total of n elements are allocated to S_t in round t. This shows that $|S_t| = n$ and hence, by line 39, that $|C_t(t)| = n$.

Invariants d, **e**, **f**, and **h-1**. As in the Case I, $C_j(t) = C_j(t-1) - S_t$ for every $j, 1 \leq j \leq t-1$, and $C_1, \ldots, C_{t-1}(t), S_t$ is a partition of $[p]$. As in Case I, this implies that invariants **d**, **e** and **f** hold for t. The proof for invariant **h-1** is also the same as in Case I.

Invariant i and **h-2**. By invariant **h-2** for $t-1$, for $t-1$, all the sets $C_J(t-2), 1 \leq j \leq t-2$, are of one of the two sizes $\alpha(t-1)$ and $\alpha(t-1)+1$, i.e., (recall that the condition of line 27 is true), of size $\alpha(t)$ or $\alpha(t)+1$. Also, by invariant **i** for $t-1$, $J(t-1)$ is the set of indices $i, 1 \leq i \leq t-2$ for which $C_i(t-1)$ is of size $\alpha(t-1)+1$ i.e., of size $\alpha(t)+1$. At the end of line 30, the set of indices $i, 1 \leq i \leq t-1$ for which the value of C_i is of size $\alpha(t)+1$ is the set $J(t-1) \cup \{t-1\}$. Let I'_{init} denote the value held by I' at the end of line 31. In the **while**-loop of line 33, (finishing in line 38), for every index k in I'_{init}, an element is transferred from C_k to S_t. Hence, at the end of the **while**-loop all the sets $C_i, 1 \leq i \leq t-1$ are of size $\alpha(t)$ or $\alpha(t)+1$. This proves invariant **h-2** for t. Furthermore, in the **while**-loop of line 33, the set of indices i for which the value of C_i is of size $\alpha(t)+1$ is reduced to $J(t-1) \cup \{t-1\} - I'_{\text{init}}$. The value $J(t-1) \cup \{t-1\} - I'_{\text{init}}$ is the value $J(t)$ given to J in line 32. (J is not altered further in round t and hence the value allocated to J on line 32 is the value $J(t)$.) This establishes the first part of invariant **i**: $J(t) = \{i \in [t-1]; |C_i(t)| = \alpha(t)+1\}$.

We compute $|J(t)|$.

$$\begin{aligned}
|J(t)| &= |J(t-1)| + 1 - |I'_{\text{init}}| \\
&= |J(t-1)| + 1 - (\alpha(t)+1) \\
&= p - n - \lfloor \tfrac{p-n}{t-2} \rfloor (t-2) - \alpha(t) \quad \text{(by invariant } \mathbf{i} \text{ for } t-1) \\
&= p - n - \lfloor \tfrac{p-n}{t-1} \rfloor (t-2) - \alpha(t) \quad \text{(because } \alpha(t) = \lfloor \tfrac{p-n}{t-1} \rfloor = \lfloor \tfrac{p-n}{t-2} \rfloor \text{)} \\
&= p - n - \lfloor \tfrac{p-n}{t-1} \rfloor (t-1) \, .
\end{aligned}$$

This finishes the proof of invariant **i** for t. □

Lemma 7.4.2 *Let t be a positive integer. Then $\lfloor \frac{p-n}{t} \rfloor < n$ if and only if $t \geq \lfloor p/n \rfloor$.*

PROOF. Write $p = an + b$ with $0 \leq b < n$. Assume that $t \geq \lfloor p/n \rfloor$. Then

$$\left\lfloor \tfrac{p-n}{t} \right\rfloor \leq \left\lfloor \tfrac{p-n}{\lfloor p/n \rfloor} \right\rfloor = (an + b - n)/a = n + (b-n)/a < n\,.$$

We now prove that $[1 \leq t \leq \lfloor p/n \rfloor - 1] \Rightarrow \lfloor \frac{p-n}{t} \rfloor \geq n$. If $\lfloor p/n \rfloor = 1$ the implication is trivially true. Assume that $\lfloor p/n \rfloor > 1$. We have:

$$\left\lfloor \tfrac{p-n}{t} \right\rfloor \geq \left\lfloor \tfrac{p-n}{\lfloor p/n \rfloor - 1} \right\rfloor = (an + b - n)/(a-1) = n + b/(a-1) \geq n\,.$$

□

Lemma 7.4.3 *For every integers q and t the system of equations*

$$\begin{cases} q = a\alpha + b(\alpha + 1) \\ a + b = t \\ a > 0, b \geq 0 \end{cases}$$

has exactly one integer-valued system of solutions: $\alpha = \lfloor \frac{q}{t} \rfloor$, $b = q - \lfloor \frac{q}{t} \rfloor t$ and $a = t - b$.

PROOF. **Uniqueness:** The equation $q/t = (a/t)\alpha + (b/t)(\alpha + 1)$ shows that q/t is a convex combination of α and $\alpha + 1$. Hence α must be equal to $\lfloor \frac{q}{t} \rfloor$. Also, the equation $q = (a+b)\alpha + b = t\alpha + b$ shows that b must be equal to $q - t\lfloor \frac{q}{t} \rfloor$.

Existence: Write q/t as the convex combination of $\alpha = \lfloor \frac{q}{t} \rfloor$ and of $\alpha + 1 = \lfloor \frac{q}{t} \rfloor + 1$: $q/t = u\alpha + v(\alpha + 1)$ with $u + v = 1$. We have $q/t = (u+v)\alpha + v = \alpha + v = \lfloor \frac{q}{t} \rfloor + v$. This shows that v is equal to $q/t - \lfloor \frac{q}{t} \rfloor$ and that $u = 1 - v = 1 - q/t + \lfloor \frac{q}{t} \rfloor$. Hence we can chose $b = vt = q - \lfloor \frac{q}{t} \rfloor t$ and $a = ut = (1-v)t = t - b$. □

Lemma 7.4.4 *Let n, p and t be three positive integers such that $n \leq p$. Assume that $\lfloor \frac{p-n}{t-1} \rfloor = \lfloor \frac{p-n}{t-2} \rfloor$. Then $\lfloor \frac{p-n}{t-1} \rfloor \leq p - n - \lfloor \frac{p-n}{t-2} \rfloor (t-2)$.*

PROOF. Obviously, $0 \leq p - n - \lfloor \frac{p-n}{t-1} \rfloor (t-1)$ so that $\lfloor \frac{p-n}{t-1} \rfloor \leq p - n - \lfloor \frac{p-n}{t-1} \rfloor (t-1) + \lfloor \frac{p-n}{t-1} \rfloor$. By assumption, $\lfloor \frac{p-n}{t-1} \rfloor = \lfloor \frac{p-n}{t-2} \rfloor$ and hence $\lfloor \frac{p-n}{t-1} \rfloor \leq p - n - \lfloor \frac{p-n}{t-2} \rfloor (t-2)$. □

We now establish that, for every t, the quantity $\prod_{j=1}^{t-1} |C_j(t)|$ is deterministic i.e., does not depend on the successive values returned by the randomized invocations `arbitrary` and `uniform`. This means in particular that, for every t, the quantity $\prod_{j=1}^{t-1} |C_j(t)|$ is the same for all *prog* in $Prog_0$ and does not depend on the values taken by the random variables $\mathcal{S}_t, J_1, J_2, \ldots$

Lemma 7.4.5 *For every prog* $\in$ *Prog*$_0$*, for every* $t, 1 \leq t \leq p-n+1$*, the product* $\prod_{j=1}^{t-1} |C_j(t)|$ *is uniquely determined as follows. (For conciseness we let* $a = p - n - \lfloor \frac{p-n}{t-1} \rfloor (t-1)$ *and* $b = t - 1 - a = t - 1 - p + n + \lfloor \frac{p-n}{t-1} \rfloor (t-1)$*.)*

$$\begin{array}{rcll} \prod_{j=1}^{t-1} |C_j(t)| & = & n^{t-1} & \text{if } t \leq \lfloor \frac{p}{n} \rfloor, \\ & = & (\lfloor \frac{p-n}{t-1} \rfloor + 1)^a \lfloor \frac{p-n}{t-1} \rfloor^b & \text{if } \lfloor \frac{p}{n} \rfloor + 1 \leq t \leq p-n+1 \ . \end{array}$$

Proof. If $t \leq \lfloor \frac{p}{n} \rfloor$ the result is an immediate consequence of invariant **g**. If $\lfloor \frac{p}{n} \rfloor + 1 \leq t \leq p-n+1$ the result is a consequence of invariants **h** and **i**. □

7.4.4 The probability space associated to *Prog*$_0$

By definition, a program *prog* in *Prog*$_0$ has randomized invocations. The output values of *prog* are the final values of $S_1, \ldots, S_p$. The randomized invocations are *internal* actions. For each *prog* in *Prog*$_0$ we can construct a probability space $(\Omega', \mathcal{G}', P_{\text{prog}})$ allowing to measure (in a probabilistic sense) not only the output – the schedules $S_1, \ldots, S_p$ – but also all the randomized invocations made by *prog*.

We will need in Section 7.7 to analyze programs in *Prog*$_0$ and to manipulate some events from $(\Omega', \mathcal{G}', P_{\text{prog}})$. We therefore describe informally the construction of this probability space. The sample space Ω' contains the sequences of values produced by randomized invocations during executions of *Prog*$_0$, i.e., the realizations of the sequence (randomized-invocation$_1$, randomized-invocation$_2$, ...). The σ-field $\mathcal{G}'$ is the power set of Ω'. The measure P_{prog} is the measure characterized by the relations

$$P_{\text{prog}}\Big[\ \cdot \mid \text{past randomized invocations are } r_1, \ldots, r_t \Big] = P_{r_1, \ldots, r_t},$$

one for each sequence $r_1, \ldots, r_t$. By integration this means that, for every sequence $r_1, \ldots, r_t$, the probability $P_{\text{prog}}[(r_1, \ldots, r_t)]$ is given by $P_e[r_1]\, P_{r_1}[r_2] \cdots P_{r_1, \ldots, r_{t-1}}[r_t]$, where we let P_e denote the the probability attached to the empty sequence e.

In contrast, recall that in Section 7.2 we constructed the sample space associated to protocols π to contain the sequences of values taken by random schedules i.e., the realizations of the sequence $(S_1, S_2, \ldots)$. As each set S_t outputted by *prog* can be constructed from the sequence

(randomized-invocation$_1$, randomized-invocation$_2$, ...),

we see that the σ-field $\mathcal{G}'$ is bigger then the one considered for protocols. For every *prog* in *Prog*$_0$, the measure P_{prog} extends the measure π_{prog} defined on the set of schedules produced by *prog* onto this bigger space $(\Omega', \mathcal{G}')$.

Recall that, for every t, every t-schedule σ and every t-fault sequence adapted to σ, the conditional probability $\pi[\ \cdot \mid \mathcal{S}_t = \sigma]$ is well defined whereas $\pi[\ \cdot \mid \mathcal{S}_t = \sigma, \mathcal{F}_t = \phi]$

is not. This is because the event $\{\mathcal{F}_t = \phi\}$ is not expressible in terms of the schedule $S_1, S_2, \ldots$ and hence is not in the σ-field over which the probability distribution π is defined. In Lemma 7.3.3 we formally made sense of this conditional probability for events in $\mathcal{G}'_t$, i.e., for events describable in terms of the set S_{t+1} and in terms of the decisions σ and ϕ taken up to round t by both the protocol and the adversary. We let $P_\pi\left[\cdot \mid \mathcal{S}_t = \sigma, \mathcal{F}_t = \phi\right]$ denote this extension.

We can make a similar extension with P_{prog} so as to compute the probability of events depending on the randomized invocations done by *prog* up to round $t+1$ – the current round – and of past decisions (i.e., up to round t), conditioned on the past decisions taken by both the protocol and the adversary. To simplify we use also P_{prog} to denote this extension. For instance, in the proof of Lemma 7.7.4 we will consider the expression

$$P_{\text{prog}}\Big[\, T \geq t+1 \;\Big|\; \mathcal{S}_t = \sigma,\, \mathcal{F}_t = \phi,\, T \geq t,\, J(t+1) = \mathcal{J} \,\Big] \ .$$

7.4.5 The optimality property

The optimization problem $\sup_\pi \inf_{\mathcal{A}} E_{\pi,\mathcal{A}}[T]$ is called the *primal problem*. We let t_{opt} denote its value. Similarly the optimization problem $\inf_{\mathcal{A}} \sup_\pi E_{\pi,\mathcal{A}}[T]$ is called the *dual problem* and we let t'_{opt} denotes its value. A protocol π_{opt} *solves* the primal problem if $t_{\text{opt}} = t(\pi_{\text{opt}})$. [9] The existence of such a protocol implies in particular that the sup is attained in the primal problem and that $\max_\pi \inf_{\mathcal{A}} E_{\pi,\mathcal{A}}[T] = \inf_{\mathcal{A}} E_{\pi_0,\mathcal{A}}[T]$. An adversary $\mathcal{A}_{\text{opt}}$ solves the dual problem if $t'_{\text{opt}} = t'(\mathcal{A}_{\text{opt}}) \stackrel{\text{def}}{=} \sup_\pi E_{\pi,\mathcal{A}_{\text{opt}}}[T]$. The existence of such an adversary implies in particular that the inf is attained in the dual problem and that $\min_{\mathcal{A}} \sup_\pi E_{\pi,\mathcal{A}}[T] = \sup_\pi E_{\pi,\mathcal{A}_{\text{opt}}}[T]$.

The following adversary $\mathcal{A}_0$ plays a fundamental role in the understanding and the analysis of protocols in $Prot(Prog_0)$.

Definition 7.4.1 *We let $\mathcal{A}_0$ denote the adversary that selects at each round t an element chosen uniformly at random from $s_t - \{f_1, \ldots, f_{t-1}\}$ if this set is not empty, and selects an arbitrary element from s_t otherwise.*

This formally means that $\mathcal{A}_0 = (Q_v)_{v \in V}$ where V is the family of t-odd-executions and where, for every $v = (s_1, f_1, \ldots, s_t)$, Q_v is equal to $\mathcal{U}_{s_t - \{f_1,\ldots,f_{t-1}\}}$ when $s_t - \{f_1, \ldots, f_{t-1}\} \neq \emptyset$ – and Q_v arbitrary when $s_t - \{f_1, \ldots, f_{t-1}\} = \emptyset$.

Our main result is described in the next theorem.

[9] We presented in Page 143 the definition of of $t(\pi)$ for a protocol π: $t(\pi) = \inf_{\mathcal{A}} E_{\pi,\mathcal{A}}[T]$. For an adversary $\mathcal{A}$, $t'(\mathcal{A})$ is defined symmetrically by $t'(\mathcal{A}) = \sup_\pi E_{\pi,\mathcal{A}}[T]$.

Theorem 7.4.6 *Every protocol π_0 in Prot (Prog$_0$) solves the primal problem while the adversary $\mathcal{A}_0$ solves the dual problem. These two problems have the same value, equal to $E_{\pi_0,\mathcal{A}_0}[T]$.*

The two following lemmas are the crucial ingredients to the proof of Theorem 7.4.6. We defer their proof after the one of Theorem 7.4.6. The first lemma expresses that the protocols in *Prot*(*Prog*$_0$) are optimal against adversary $\mathcal{A}_0$.

Lemma 7.4.7 *Let π_0 be a protocol in Prot(Prog$_0$). Then* $\max_\pi E_{\pi,\mathcal{A}_0}[T] = E_{\pi_0,\mathcal{A}_0}[T]$.

The next lemma expresses that, when a protocol π_0 in *Prot*(*Prog*$_0$) is used, the expected time of survival of the system is independent of the adversary $\mathcal{A}$ used in conjunction with π_0. This implies in particular that $\mathcal{A}_0$ is optimal against the protocol π_0.

Lemma 7.4.8 *Let π_0 be a protocol in Prot(Prog$_0$). Then $E_{\pi_0,\mathcal{A}}[T]$ is independent of $\mathcal{A}$.*

We are now ready to prove our main result.

PROOF of Theorem 7.4.6:
Let π_0 be a protocol in *Prot*(*Prog*$_0$). Then

$$\begin{array}{rcll} \sup_\pi E_{\pi,\mathcal{A}_0}[T] & = & E_{\pi_0,\mathcal{A}_0}[T] & \text{(by Lemma 7.4.7)} \\ & = & \inf_{\mathcal{A}} E_{\pi_0,\mathcal{A}}[T] & \text{(by Lemma 7.4.8).} \end{array}$$

By Lemma 8.2.2, $\sup_\pi \inf_{\mathcal{A}} E_{\pi,\mathcal{A}}[T] = \inf_{\mathcal{A}} E_{\pi_0,\mathcal{A}}[T]$ i.e., π_0 solves the primal problem, and similarly $\mathcal{A}_0$ solves the dual problem. Furthermore these two problems have the same value, equal to $E_{\pi_0,\mathcal{A}_0}[T]$. □

Note: As discussed in Section 7.1.2, the equality of the values of the primal and the dual problem is a direct consequence of Von-Neumann's theorem.

7.5 The Random Variables $C_j(t)$ and $L_i(t)$

In Section 7.4, for every program *prog* in $Prog_0$, for every t and j, $1 \leq j \leq t \leq p$, we defined $C_j(t)$ to be the value held by the program variable C_j at the end of round t of *prog*. In this section we define the values $C_j(t)$ to be random variables expressed in terms of the schedule $\mathcal{S}_t$. This definition is valid for arbitrary protocols and is compatible with the definition given in Section 7.4 in the special case of protocols in $Prot(Prog_0)$. (See invariant **d** of Lemma 7.4.1.)

For each j and t, $1 \leq j \leq t$, $C_j(t)$ is the set of elements selected at time j and which are not used in ulterior rounds $j+1, j+2, \ldots, t$. $C_0(t)$ is the set of elements of $[p]$ that are not used in the first t rounds. We also introduce the random variables $L_i(t)$ counting the number of sets $C_j(t), 1 \leq j \leq t-1$, which are of size i.

An indication of the relevance of these random variables is indicated by Lemma 7.4.5 which expresses that every *prog* in $Prog_0$ is such that for every $t, 1 \leq t \leq \lfloor p/n \rfloor$, $L_n(t) = t-1$ and $L_i(t) = 0$ if $i \neq n$; and such that for every $t, t \geq \lfloor p/n \rfloor + 1$, $L_i(t) = p - n - \lfloor \frac{p-n}{t-1} \rfloor (t-1)$ if $i = \lfloor \frac{p-n}{t-1} \rfloor + 1$, $L_i(t) = t - 1 - p + n + \lfloor \frac{p-n}{t-1} \rfloor (t-1)$ if $i = \lfloor \frac{p-n}{t-1} \rfloor$ and $L_i(t) = 0$ for any other value of i. We will see that these properties do in fact characterize optimal schedules. The following definition formalizes these notions. For a set Φ in $\{1, \ldots, p\}$, we use the notation Φ^c to denote the complement of Φ in $\{1, \ldots, p\}$.

Definition 7.5.1 *For every integers j and t, $1 \leq j \leq t \leq p$ and every $k, 0 \leq k \leq n$ we define the following random variables:*

$$\begin{aligned}
C_1(t) &= S_1 \cap \Big(S_2 \cup \ldots \cup S_t\Big)^c \\
C_2(t) &= S_2 \cap \Big(S_3 \cup \ldots \cup S_t\Big)^c \\
&\ldots \\
C_j(t) &= S_j \cap \Big(S_{j+1} \cup \ldots \cup S_t\Big)^c \\
&\ldots \\
C_t(t) &= S_t.
\end{aligned}$$

We extend the definition of $C_j(t)$ to $j = 0$ and set

$$C_0(t) = \Big(\cup_{j=1}^{t} C_j(t)\Big)^c.$$

We say that $(C_0(t), \ldots, C_t(t))$ is the c-sequence derived from the schedule $(S_1, \ldots, S_t)$. For $1 \leq t \leq p$ and $0 \leq i \leq n$, we let $L_i(t)$ denote the number of sets $C_j(t), 1 \leq j \leq$

$t-1$, *which are of size* i:

$$L_i(t) \stackrel{\text{def}}{=} \left|\left\{C_j(t);\ 1 \le j \le t-1, |C_j(t)| = i\right\}\right| .$$

We extend the definition of $L_i(t)$ *to* $i = n+1$ *and set*

$$L_{n+1}(t) \stackrel{\text{def}}{=} |C_0(t)| .$$

As usual, we use lower case letters and write for instance $c_0(t), c_1(t), \ldots, c_t(t)$ to denote the realizations of the random variables $C_0(t), C_1(t), \ldots, C_t(t)$ and $l_0(t), \ldots, l_{n+1}(t)$ to denote the realizations of the random variables $L_0(t), \ldots, L_{n+1}(t)$.

The next properties are simple but fundamental. Property 3 means that the system is still alive by time t if and only if for every $j, j \le t-1$, the j-th fault is in the set C_j. Let us emphasize that this property would not hold for $m \ge 2$.

Lemma 7.5.1 *Let* $t, 1 \le t \le p$ *be arbitrary. Then:*

1. *The family* $(C_j(t))_{0 \le j \le t}$ *forms a partition of* $[p]$.
2. $\sum_{i=0}^{n} L_i(t) = t-1$.
3. $\{T \ge t\} = \bigcap_{j=1}^{t-1} \{F_j \in C_j(t)\}$ $(= \bigcap_{u=1}^{t} \{S_u \cap \{F_1, \ldots, F_{u-1}\} = \emptyset\})$.

Proof. We easily check that the family $(C_j(t))_{0 \le j \le t}$ is a partition of $[p]$. The condition $\sum_{i=0}^{n} L_i(t) = t-1$ just expresses that the $t-1$ sets $C_1(t), \ldots, C_{t-1}(t)$ are different and of cardinality between 0 and n. As expressed by the formula $\{T \ge t\} = \cap_{u=1}^{t} \{S_u \cap \{F_1, \ldots, F_{u-1}\} = \emptyset\}$ that we recall for completeness, the survival time T is at least t if and only if, for every time $u, 2 \le u \le t$, S_u does not contain any of $F_1, \ldots, F_{u-1}$. (Note that this fact uses the hypothesis $m = 1$.) Equivalently, $T \ge t$ if and only if, for every $j, 1 \le j \le t-1$, F_j is not contained in any of $S_{j+1}, \ldots, S_t$, i.e., if $F_j \in (S_{j+1} \cup \ldots \cup S_t)^c$. On the other hand, by definition, $F_j \in S_j$ for every j. Therefore $T \ge t$ if and only if $F_j \in C_j(t)$ for every $j, 1 \le j \le t-1$. □

Definition (7.5.1) establishes how a c-sequence $(c_j(t))_j$ can be derived from a given schedule $(s_1, \ldots, s_t)$. The following lemma conversely characterizes the sequences $(\gamma_j)_j$ that can be realized as a c-sequence from some schedule $(s_1, \ldots, s_t)$. This result will be used in Lemma 7.6.11 to characterize the optimal schedules.

Lemma 7.5.2 *Let $\gamma_1, \ldots, \gamma_{t-1}$ be integers in the interval $[0, n]$. Then the condition*

$$n \leq N - (\gamma_1 + \ldots + \gamma_{t-1})$$

is a necessary and sufficient condition for the existence of a schedule $(s_1, \ldots, s_t)$ satisfying $|c_j(t)| = \gamma_j$ for all $j, 1 \leq j \leq t-1$.

PROOF. We first establish the necessity of the condition. By Lemma 7.5.1, for all schedules $(s_1, \ldots, s_t)$, the family $(c_j(t))_{0 \leq j \leq t}$ forms a partition of $[p]$. This clearly implies that $\sum_0^t |c_j(t)| = p$ and hence that $\sum_1^t |c_j(t)| \leq p$.

Conversely, we prove by induction on t that, for every sequence $\gamma_1, \ldots, \gamma_{t-1}$ of integers in $[0, n]$ such that $n \leq p - (\gamma_1 + \ldots + \gamma_{t-1})$, there exists a schedule $(s_1, \ldots, s_t)$ whose associated c-sequence is given by $c_j(t) = \gamma_j, 1 \leq j \leq t-1$ and $c_0(t) = p - (\gamma_1 + \ldots + \gamma_{t-1}) - n$.

• The property is trivially true for $t = 1$: in this case the family of conditions $|c_j(t)| = \gamma_j, 1 \leq j \leq t-1$ is empty, and the set $c_0(1)$ of processors not used has size $p - |s_1| = p - n$.

• Assume the property verified for t, and consider a sequence $\gamma_1, \ldots, \gamma_t$ of integers in $[0, n]$ such that $n \leq p - (\gamma_1 + \ldots + \gamma_t)$. This condition trivially implies that $n \leq p - (\gamma_1 + \ldots + \gamma_{t-1})$. Therefore, the result at the induction level t being true, there exists a schedule $(s_1, \ldots, s_t)$ for which $c_j(t) = \gamma_j, 1 \leq j \leq t-1$ and such that the number of processors still unused at time t is

$$|c_0(t)| \quad = \quad p - (\gamma_1 + \ldots + \gamma_{t-1}) - n \ .$$

We then construct a set s_{t+1} of n processors by taking any $n - \gamma_t$ elements from the set s_t and any γ_t elements from the set $c_0(t)$ of unused processors. This construction is possible exactly when $\gamma_t \leq |c_0(t)|$ i.e., when

$$\gamma_t \leq p - (\gamma_1 + \ldots + \gamma_{t-1}) - n \ ,$$

which is true: this is the induction hypothesis. Note that, by construction, the sets $c_j(t), 1 \leq j \leq t-1$ are unaffected by the selection of the set s_{t+1} and hence:

$$|c_j(t+1)| = |c_j(t)| = \gamma_j, \quad 1 \leq j \leq t-1 \ .$$

On the other hand,

$$\begin{aligned} |c_t(t+1)| &= |s_t| - (n - \gamma_t) \\ &= \gamma_t \ . \end{aligned}$$

To finish the induction from t to $t+1$, we note that

$$\begin{aligned} |c_0(t+1)| &= |c_0(t)| - \gamma_t \\ &= p - (\gamma_1 + \ldots + \gamma_{t-1} + \gamma_t) - n \ . \end{aligned}$$

□

The schedules in normal form that we now define will be shown in the sequel to be the schedules maximizing the expected survival time. In essence, a t-schedule σ is in normal form if the sizes of the sets $c_j(t), 1 \leq j \leq t-1$, (of the c-sequence associated to σ), differ by at most one, and if $c_0(t)$ is not empty only if every $c_j(t), 1 \leq j \leq t-1$ is of size n. Formally:

Definition 7.5.2 *Let $t \geq 1$ be a time. We say that a t-schedule σ is in* normal form *and write*

$$\sigma \in \mathcal{N}_t$$

if there exists $\alpha_t,\ 0 \leq \alpha_t \leq n$, s.t.,

$$\begin{cases} \forall j, 1 \leq j \leq t-1, \quad |c_j(t)| = \alpha_t \ \textit{or} \ |c_j(t)| = \alpha_t + 1 \ , \\ |c_0(t)| > 0 \Rightarrow \alpha_t = n \ . \end{cases}$$

We then also say that $(c_0(t), \ldots, c_{t-1}(t))$ is in normal form.

Invariants **g** and **h** of Lemma 7.4.1 express that the programs in $Prog_0$ all produce schedules in normal form. We next show that a t-schedule σ is in normal form if and only if the associated sequence $(l_0(t), \ldots, l_{n+1}(t))$ has at most two non-zero terms, which are consecutive.

Lemma 7.5.3 *A t-schedule σ is in normal form if and only if there exists $\alpha_t, 0 \leq \alpha_t \leq n+1$ such that the associated sequence $(l_0(t), \ldots, l_{n+1}(t))$ satisfies the equality*

$$\Big(l_0(t), \ldots, l_{n+1}(t)\Big) = \Big(0, \ldots, 0, l_{\alpha_t}(t), l_{\alpha_t+1}(t), 0, \ldots, 0\Big) \ .$$

Proof. Simple consequence of Definitions 7.5.1 and 7.5.2. □

The next lemma shows that, for every t, t-schedules in normal form have a unique associated sequence $(l_0(t), \ldots, l_{n+1}(t))$ and that, conversely, this value characterizes t-schedules in normal form. We will use this property in Lemma 7.6.11 to show that a property is specific to schedules in normal form.

Lemma 7.5.4 1. *Let $1 \leq t \leq \lfloor p/n \rfloor$. Then a t-schedule σ is in $\mathcal{N}_t$ if and only if the only non zero terms of the sequence $(l_0(t), \ldots, l_{n+1}(t))$ are $l_n(t) = t-1$ and $l_{n+1}(t) = p - tn$. (And hence, $\alpha_t = n$.)*

2. *Let $t > \lfloor p/n \rfloor$. Then a t-schedule σ is in $\mathcal{N}_t$ if and only if the only non zero terms of the sequence $(l_0(t), \ldots, l_{n+1}(t))$ are*

$$l_{\alpha_t}(t) = \lfloor \tfrac{p-n}{t-1} \rfloor (t-1) + t + n - p - 1$$

and, if $p-n$ is not a multiple of $t-1$,

$$l_{\alpha_t+1}(t) = p - n - \lfloor \tfrac{p-n}{t-1} \rfloor (t-1)\,,$$

where

$$\alpha_t \stackrel{\text{def}}{=} \lfloor \tfrac{p-n}{t-1} \rfloor\,.$$

PROOF. • If $t \leq \lfloor p/n \rfloor$ then $tn \leq p$. Working by contradiction, assume that there exists $i < n$ such that $l_i(t) > 0$, i.e., that there exists a set $c_{j_0}(t)$ such that $|c_{j_0}(t)| < n$. Then

$$\begin{aligned} |\bigcup_{j=1}^{t} c_j(t)| &\leq \sum_{j=1}^{t} |c_j(t)| \\ &= \sum_{j \neq j_0} |c_j(t)| + |c_{j_0}(t)| \\ &< tn \leq p\,. \end{aligned}$$

As the family $(c_j(t)), 0 \leq j \leq t$, is a partition of $[p]$, $c_0(t)$ must be not empty which, by the normality property, implies that all the sets $c_j(t), 1 \leq j \leq t-1$, must have size n. Hence

$$\begin{aligned} l_n(t) &\stackrel{\text{def}}{=} \left|\{c_j(t);\ 1 \leq j \leq t-1, |c_j(t)| = n\}\right| \\ &= t-1\,. \end{aligned}$$

This in turn implies that

$$\begin{aligned} l_{n+1}(t) \stackrel{\text{def}}{=} |c_0(t)| &= p - |\bigcup_{j=1}^{t} c_j(t)| \\ &= p - nt\,. \end{aligned}$$

• Consider now the case $t > \lfloor p/n \rfloor$. Working by contradiction, assume that $l_{n+1}(t) > 0$. Then, by the normality condition, all the sets $c_j(t), 1 \leq j \leq t-1$ must have size n.

We have:

$$\begin{aligned} |\bigcup_{j=1}^{t} c_j(t)| &= \sum_{j=1}^{t} |c_j(t)| \qquad \text{(since the sets } c_j(t) \text{ are disjoint)} \\ &= tn \\ &\geq (\lfloor p/n \rfloor + 1)n \\ &> p\,, \end{aligned}$$

a contradiction. Hence $l_{n+1}(t) = 0$, i.e., $c_0(t) = \emptyset$. Hence the $t-1$ sets $c_1(t), \ldots, c_{t-1}(t)$ must divide $p - |s_t| = p - n$ elements among themselves. Lemma 7.4.3 (where we replace q by $p-n$ and t by $t-1$) shows that $\alpha_t \stackrel{\text{def}}{=} \lfloor \frac{p-n}{t-1} \rfloor$, $l_{\alpha_t}(t) = \lfloor \frac{p-n}{t-1} \rfloor (t-1)+t+n-p-1$ and $l_{\alpha_t+1}(t) = p - n - \lfloor \frac{p-n}{t-1} \rfloor (t-1)$. □

7.6 π_0 is optimal against $\mathcal{A}_0$

This section is devoted to the proof of Lemma 7.4.7.

7.6.1 Sketch of the proof and intuitions

For a given adversary $\mathcal{A}$, an optimal protocol is one that, at each time $t+1$, exploits to its fullest all the information available so as to optimize the choice of S_{t+1}. Therefore to construct an optimal protocol, we first analyze the notion of "information available to the protocol". We distinguish between the *explicit* information, formally described in the model, and the *implicit* information that an *optimal* protocol is able to deduce. Consider for instance the case of the identity of the adversary $\mathcal{A}$. In Section 7.2, when modeling protocols and adversaries, we did not provide a mechanism allowing a protocol to be informed of the identity of the adversary that it plays against. This means that the protocol does not know explicitly the identity of the adversary. Nevertheless, for a given adversary $\mathcal{A}$, there is one protocol that always assumes that the adversary is $\mathcal{A}$ and takes optimal decisions based on this assumption. This protocol is by construction optimal if the adversary *is* $\mathcal{A}$. We then say that the optimal protocol knows implicitly the identity of the adversary.

In Section 7.2 we modeled a protocol to be an entity deciding the whole schedule $(S_1, \ldots, S_p)$ ahead of time, i.e., in an off-line fashion. S_1 is the first selected set, and for every t, $t \geq 1$, S_{t+1} is the set selected to be used after the occurrence of the t-th fault. In this model the adversary "sees" the sets S_t only when they come in operation.

Alternatively, we could have modeled a protocol to be an entity interacting in an on-line fashion with the adversary: in this model, at each occurrence of a fault, the adversary informs the protocol of the occurrence of a fault *and* whether the system is still alive at this point. If the system is still alive the protocol then selects the set S_{t+1} to be used next and communicates its choice to the adversary.

It might seem that in our model – the off-line model – the protocol is weaker then in the on-line model, as, for every t, it has to select the set S_{t+1} without knowing whether the system is alive after the t-th fault. Nevertheless we easily see that, in the off-line model, the protocol can assume without loss of generality that $T \geq t$, i.e., that the system is alive after the t-th fault, while selecting the set S_{t+1} for time $t+1$. Indeed, if this happens not to be the case and the system dies before time $t+1$, all the decisions and assumptions made by the protocol for time $t+1$ are irrelevant.

This discussion shows that the off-line and the on-line model of a protocol are equivalent so that we can adopt the on-line model in this informal presentation. From the on-line description, it is clear that, for every t, the information about the past execution available to the protocol upon selecting the set S_{t+1} consists of its own past decisions, i.e., of $\mathcal{S}_t = \sigma$, and of the fact that the system is still alive at this point, i.e., of $T \geq t$. Note that the information $T \geq t$ is given explicitly to the protocol in the on-line model and only given implicitly in the off-line model that we chose in Section 7.2.

For every t, upon selecting a new set S_{t+1}, an optimal protocol can use all the information available and guess first what are the locations $F_1, \ldots, F_t$ of the faults already committed by the adversary. To "guess" the protocol uses the probability distribution

$$P_{\mathcal{A}}[(F_1, \ldots, F_t) = \cdot \mid T \geq t,\, \mathcal{S}_t = \sigma]\,,$$

i.e., the probability of the value allocated to $(F_1, \ldots, F_t)$ by the *adversary* $\mathcal{A}$, conditioned on the knowledge of the past execution held by the *protocol*.

By Lemma 7.5.1, the system is alive at time t, i.e., $T \geq t$, *exactly* if F_j is in $C_j(t)$ for every $j, 1 \leq j \leq t-1$. (This is not true for $m \geq 2$.) Hence the previous probability can be rewritten $P_{\mathcal{A}}[(F_1, \ldots, F_t) = \cdot \mid \cap_{j=1}^{t} \{F_j \in c_j(t)\}, \mathcal{S}_t = \sigma]$, which shows the relevance to the analysis of the sets $C_j(t)$. In this section the adversary is the random adversary $\mathcal{A}_0$ defined in Definition 7.4.1. In this case, by definition, at each time t, a fault occurs uniformly at random in S_t and we can then further establish in Section 7.6.2 that $P_{\mathcal{A}_0}[(F_1, \ldots, F_t) = \cdot \mid T \geq t,\, \mathcal{S}_t = \sigma] = \bigotimes_{j=1}^{t} \mathcal{U}_{c_j(t)}$.

This result fully elucidates the notion of "information available to the protocol" and we can say that a protocol optimal against $\mathcal{A}_0$ is one that, for each t, uses most efficiently the probabilistic guess $P_{\mathcal{A}_0}[(F_1, \ldots, F_t) = \cdot \mid T \geq t,\, \mathcal{S}_t = \sigma]$ in order to chose a "most appropriate" set S_{t+1} for time $t+1$. The next challenge towards the construction of optimal protocols is to understand how, for every t, such a "most appropriate" set S_{t+1} is selected. For this we use the general equality

$$E_{\pi,\mathcal{A}_0}[T] = \sum_{t \geq 1} P_{\pi,\mathcal{A}_0}[T \geq t]\,,$$

established in Lemma 8.1.2. A natural idea is to try the following greedy strategy. Select a set s maximizing the quantity $P_{\mathcal{A}_0}[T \geq 1 \mid \mathcal{S}_1 = s]$. Then select a set s maximizing the quantity $P_{\mathcal{A}_0}[T \geq 2 \mid T \geq 1,\, \mathcal{S}_2 = (s_1, s)]$. Generally, assuming that the schedule $\mathcal{S}_t = (s_1, \ldots, s_t)$ has already been chosen and assuming that the system is still alive at time t, i.e., that $T \geq t$, we select a set s maximizing the probability

$$P_{\mathcal{A}_0}[T \geq t+1 \mid T \geq t,\, \mathcal{S}_{t+1} = (s_1, \ldots, s_t, s)]$$

of being alive one more time. If the protocol π defined by this procedure maximizes $P_{\pi,\mathcal{A}_0}[T \geq t]$ for every t, it also maximizes the sum $\sum_{t\geq 1} P_{\pi,\mathcal{A}_0}[T \geq t]$ and hence is a protocol optimal against $\mathcal{A}_0$. As is discussed in Section 7.6.4, this is true if the schedule $\sigma = (s_1, \ldots, s_t)$ greedily chosen as described above maximizes the quantity $P_{\mathcal{A}_0}[T \geq t \mid \mathcal{S}_t = \sigma]$.

We therefore compute this quantity $P_{\mathcal{A}_0}[T \geq t \mid \mathcal{S}_t = \sigma]$ for every σ and show that it is equal to $\prod_{j=1}^{t-1} \frac{|c_j(t)|}{n}$, where the values $c_j(t)$, $1 \leq j \leq t$ are uniquely derived from σ. This computation uses critically the relation $P_{\mathcal{A}_0}[(F_1, \ldots, F_t) = \cdot \mid T \geq t, \mathcal{S}_t = \sigma] = \bigotimes_{j=1}^{t} \mathcal{U}_{c_j(t)}$ discussed above. We establish that the schedules maximizing this value are the schedules for which all the associated terms $|c_j(t)|, 1 \leq j \leq t-1$, differ by at most one, i.e., the schedules in normal form. (See Definition 7.5.2.) By invariants **g** and **h** of Lemma 7.4.1, all protocols in $Prot(Prog_0)$ produce such schedules and hence are optimal against $\mathcal{A}_0$. This is formally established in Section 7.6.5.

The results established in Section 7.6.4 also show that the schedules produced by the greedy procedure previously described are in normal form, and hence that the greedy procedure is optimal against $\mathcal{A}_0$. Actually, the protocols produced by the greedy procedure are exactly those in $Prot(Prog_0)$.

7.6.2 Analysis of the distribution of the random variables F_j

Let t and j such that $0 \leq t \leq p-1$, and $1 \leq j \leq t$. Assume that the adversary is the random adversary $\mathcal{A}_0$ defined in Definition 7.4.1. The next lemma says that, conditioned on the past schedule and on the fact that the system is still alive at time t,

- each random variable F_j has a uniform distribution on the set $c_j(t)$,
- the random variables $(F_j)_{1\leq j\leq t}$ are independent.

This lemma will be a crucial tool for the estimation of the probability $P_{\mathcal{A}_0}[T \geq t | \mathcal{S}_t = \sigma]$ done in Lemma 7.6.10.

Lemma 7.6.1 *Let $1 \leq t \leq p$ and $\sigma \in Feas_{\mathcal{A}_0}$.* [10] *Then, for every family $(\Phi_1, \ldots, \Phi_t)$ of subsets of $[p]$,*

$$P_{\mathcal{A}_0}\Big[(F_1, \ldots, F_t) \in (\Phi_1, \ldots, \Phi_t) \;\Big|\; T \geq t, \mathcal{S}_t = \sigma\Big] = \prod_{j=1}^{t} \mathcal{U}_{c_j(t)}\Big[\Phi_j\Big],$$

[10] See Page 149 for the definition of $\text{Feas}_{\mathcal{A}_0}$.

where $c_j(t), 1 \leq j \leq t$, are the values of $C_j(t), 1 \leq j \leq t$, uniquely determined by the condition $\mathcal{S}_t = \sigma$.

Lemma 7.3.5, page 148, justifies the well-formedness of this statement.

PROOF. Define $(\Omega_1, \mathcal{G}_1, P_1)$ to be the probability space $(\Omega, \mathcal{G}, P_{\mathcal{A}_0})$. Using the notions recalled in Definition 8.1.2 we can reformulate the statement of Lemma 7.6.1 into the concise form: for all t, $0 \leq t \leq p-1$ and all t-schedule σ in $\mathrm{Feas}_{\mathcal{A}_0}$,

$$\begin{aligned} \mathcal{L}\Big((F_1, \ldots, F_t) \;\Big|\; T \geq t, \mathcal{S}_t = \sigma \Big) &= \bigotimes_{j=1}^{t} \mathcal{L}\Big(F_i \;\Big|\; T \geq t, \mathcal{S}_t = \sigma \Big) \\ &= \bigotimes_{j=1}^{t} \mathcal{U}_{c_j(t)}. \end{aligned} \tag{7.1}$$

Equation 7.1 holds vacuously for $t = 0$: the family $(F_i)_{1 \leq i \leq t}$ is empty and there is nothing to prove. We now work by induction on $t, 0 \leq t < p-1$: assume that Equation (7.1) has been established for time t. To progress from t to $t+1$, we first prove that, for every t-schedule σ and and every $s \in \mathcal{P}_n(p)$ such that (σ, s) is in $\mathrm{Feas}_{\mathcal{A}_0}$, for every j, $1 \leq j \leq t+1$,

$$\mathcal{L}\Big(F_j \;\Big|\; T \geq t+1, \mathcal{S}_{t+1} = (\sigma, s) \Big) = \mathcal{U}_{c_j(t+1)}. \tag{7.2}$$

• Consider the case $j = t+1$. This case corresponds to the processor F_{t+1} selected at time $t+1$ by the adversary. We want to prove that, $\mathcal{L}\Big(F_{t+1} \;\Big|\; T \geq t+1, \mathcal{S}_{t+1} = (\sigma, s) \Big) = \mathcal{U}_{c_{t+1}(t+1)}$. Conditioning on $T \geq t+1$ ensures that the adversary did not select a processor of s at a time prior to $t+1$. Hence, at the end of time $t+1$, there is exactly one faulty processor, F_{t+1}, amongst the set s. As $\mathcal{A}_0$ is the random adversary, F_{t+1} is uniformly distributed in s, i.e., has a law equal to $\mathcal{U}_s \stackrel{\text{def}}{=} \mathcal{U}_{c_{t+1}(t+1)}$.

• Let j be any integer, $1 \leq j \leq t$. We let F'_j denote the random variable $\Big(F_j \;\Big|\; T \geq t, \mathcal{S}_t = \sigma \Big)$. Then

$$\begin{aligned} &\mathcal{L}\Big(F_j \Big| T \geq t+1, \mathcal{S}_{t+1} = (\sigma, s) \Big) \\ &\quad = P_{\mathcal{A}_0}\Big[F_j \in \cdot \Big| T \geq t, \mathcal{S}_t = \sigma,\ F_1 \notin s, \ldots, F_t \notin s \Big] \qquad (7.3) \\ &\quad = P_{\mathcal{A}_0}\Big[F_j \in \cdot \Big| T \geq t, \mathcal{S}_t = \sigma,\ F_j \notin s \Big] \qquad (7.4) \end{aligned}$$

$$= P_{\mathcal{A}_0}\Big[F'_j \in \cdot \;\Big|\; F'_j \notin s \Big] \tag{7.5}$$

$$= \mathcal{U}_{c_j(t)}\Big[\cdot \;\Big|\; F'_j \notin s \Big] \tag{7.6}$$

$$= \mathcal{U}_{c_j(t)\cap s^c_{t+1}}\Big[\cdot \Big] \tag{7.7}$$

$$= \mathcal{U}_{c_j(t+1)} \ . \tag{7.8}$$

Equality 7.3 comes from the fact that, conditioned on the schedule $\mathcal{S}_t = \sigma$ and the fact that the system has survived up to time t, the system survives at time $t+1$ exactly when the set s selected at time $t+1$ contains no processor broken at a previous time. Equality 7.4 comes from the fact that, by our induction hypothesis at level t, $\Big(F_j \;\Big|\; T \geq t, \mathcal{S}_t = \sigma\Big)$ is independent from $\Big(F_i \;\Big|\; T \geq t, \mathcal{S}_t = \sigma\Big)$; $1 \leq i \leq t, i \neq j$. Equality 7.5 comes from the fact that, conditioned on $\{T \geq t, \mathcal{S}_t = \sigma\}$, the condition $F_j \notin s$ is equivalent to $F'_j \notin s$. Equality 7.6 comes from the fact that, by our induction hypothesis at level t,

$$P_{\mathcal{A}_0}[\, F'_j \in \cdot\,] = \mathcal{U}_{c_j(t)}[\,\cdot\,] \ .$$

Equality 7.7 is a simple application of Lemma 8.1.3: again, remember that $\mathcal{U}_{c_j(t)}$ is the law of F'_j. Equality 7.8 comes from the definition of $c_j(t+1)$. This finishes to establish that Equation 7.2 holds for every j, $1 \leq j \leq t+1$.

A consequence of Equation 7.2 is that, for every j, $1 \leq j \leq t$, the support of $\Big(F_j \;\Big|\; T \geq t, \mathcal{S}_t = \sigma\Big)$ is equal to $c_j(t)$. By Lemma 7.5.1 the sets $c_j(t)$ are all disjoint. This trivially implies the independence of the random variables and concludes the proof by induction of Formula 7.1. □

The proof of the next result can be omitted in first reading. It will be used in the next section and in Lemma 7.6.11 where optimal schedules are characterized.

Corollary 7.6.2 *Let σ be a schedule and let $(l_0(t), \ldots, l_{n+1}(t))$ be the associated sequence as described in Definition 7.5.1. Then σ is in $\mathrm{Feas}_{\mathcal{A}_0}$ if and only if $l_0(t) = 0$.*

PROOF.

• Assume that $\sigma \in \mathrm{Feas}_{\mathcal{A}_0}$. Then, by Equation 7.1,

$$\mathcal{L}\Big((F_1, \ldots, F_t) \;\Big|\; T \geq t, \mathcal{S}_t = \sigma \Big) = \bigotimes_{i=1}^{t} \mathcal{U}_{c_j(t)}.$$

The uniform distributions $\mathcal{U}_{c_j(t)}$ are well defined probability distributions only if $c_j(t) \neq \emptyset$ for $j, 1 \leq j \leq t-1$, i.e., if $l_0(t) = 0$.

• We now prove the converse and assume that $(s_1, \ldots, s_t)$ is a schedule which is not in $\text{Feas}_{\mathcal{A}_0}$. We want to establish that $l_0(t) > 0$.

Note first that every set s_1 of size n is clearly in $\text{Feas}_{\mathcal{A}_0}$. (Consider the protocol π such that $S_t = s_1$ for all t. Then $P_{\pi,\mathcal{A}_0}[T \geq 1, S_1 = s_1] = P_{\pi,\mathcal{A}_0}[T \geq 1] = 1$, because $m = 1$.) This establishes that $t > 1$. Let $v, 1 < v \leq t$ be the smallest u such that $(s_1, \ldots, s_u) \notin \text{Feas}_{\mathcal{A}_0}$. Using the formulation given in Remark 1 after Definition 7.3.3, page 149, and using Lemma 7.3.2 we obtain

$$P_{\mathcal{A}_0}\Big[T \geq v \;\Big|\; \mathcal{S}_v = (s_1, \ldots, s_v)\Big] = 0$$

whereas

$$P_{\mathcal{A}_0}\Big[T \geq v-1 \;\Big|\; \mathcal{S}_{v-1} = (s_1, \ldots, s_{v-1})\Big] > 0\,. \tag{7.9}$$

Therefore:

$$\begin{aligned}
0 &= P_{\mathcal{A}_0}\Big[T \geq v \;\Big|\; \mathcal{S}_v = (s_1, \ldots, s_v)\Big] \\
&= P_{\mathcal{A}_0}\Big[T \geq v, T \geq v-1 \;\Big|\; \mathcal{S}_v = (s_1, \ldots, s_v)\Big] \\
&= P_{\mathcal{A}_0}\Big[T \geq v \;\Big|\; T \geq v-1, \mathcal{S}_v = (s_1, \ldots, s_v)\Big] \cdot P_{\mathcal{A}_0}\Big[T \geq v-1 \;\Big|\; \mathcal{S}_v = (s_1, \ldots, s_v)\Big] \\
&= P_{\mathcal{A}_0}\Big[T \geq v \;\Big|\; T \geq v-1, \mathcal{S}_v = (s_1, \ldots, s_v)\Big] \cdot P_{\mathcal{A}_0}\Big[T \geq v-1 \;\Big|\; \mathcal{S}_{v-1} = (s_1, \ldots, s_{v-1})\Big] \qquad (7.10)
\end{aligned}$$

where the last equality is a consequence of Lemma 7.3.6.

Using Equations 7.9 and 7.10 allows us to derive the first next equality.

$$\begin{aligned}
0 &= P_{\mathcal{A}_0}\Big[T \geq v \;\Big|\; T \geq v-1, \mathcal{S}_v = (s_1, \ldots, s_v)\Big] \\
&= P_{\mathcal{A}_0}\Big[s_v \cap \{F_1, \ldots, F_{v-1}\} = \emptyset \;\Big|\; T \geq v-1,\ \mathcal{S}_v = (s_1, \ldots, s_v)\Big] \\
&= P_{\mathcal{A}_0}\Big[s_v \cap \{F_1, \ldots, F_{v-1}\} = \emptyset \;\Big|\; T \geq v-1,\ \mathcal{S}_{v-1} = (s_1, \ldots, s_{v-1})\Big] \qquad (7.11) \\
&= P_{\mathcal{A}_0}\Big[\bigcap_{j=1}^{v-1} \{F_j \in s_v^c\} \;\Big|\; T \geq v-1,\ \mathcal{S}_{v-1} = (s_1, \ldots, s_{v-1})\Big] \\
&= \prod_{j=1}^{v-1} \mathcal{U}_{c_j(v-1)}[s_v^c]\,, \qquad (7.12)
\end{aligned}$$

where Equation 7.11 comes from Lemma 7.3.6 and Equation 7.12 from Lemma 7.6.1. The nullity of the product in 7.12 implies that there exists $j, 1 \leq j \leq v-1$, such

that

$$\begin{aligned} & \mathcal{U}_{c_j(v-1)}[s_v^c] = 0 \\ & \quad \text{i.e.,} \quad s_v^c \cap c_j(v-1) = \emptyset \\ & \quad \text{i.e.,} \quad c_j(v) = \emptyset \ , \end{aligned}$$

and this implies that $l_0(v) > 0$. To finish, note that the sequence $(l_0(u))_u$ is a non-decreasing sequence so that $l_0(v) > 0$ implies that $l_0(t) > 0$, which was to be proven. □

7.6.3 A max-min equality for the maximal survival times

Our final proofs will rely on the equality $E_{\pi,\mathcal{A}}[T] = \sum_{t\geq 1} P_{\pi,\mathcal{A}}[T \geq t]$ and will entail an analysis of each of the non-zero terms of this summation. This section is concerned with finding which of these terms are non-zero.

Following the definitions given in Lemma 8.2.4 we define $t_{\max}(\mathcal{A}) \stackrel{\text{def}}{=} \max\{\, t; \sup_\pi P_{\pi,\mathcal{A}}[T \geq t] > 0 \,\}$ and $t_{\min}(\pi) \stackrel{\text{def}}{=} \max\{\, t; \inf_{\mathcal{A}} P_{\pi,\mathcal{A}}[T \geq t] > 0 \,\}$. (Note that, for every $\mathcal{A}$ and every π, $P_{\pi,\mathcal{A}}[T \geq t] = 0$ if $t > p$. This justifies that $\sup\{\, t; \sup_\pi P_{\pi,\mathcal{A}}[T \geq t] > 0 \,\} = \max\{\, t; \sup_\pi P_{\pi,\mathcal{A}}[T \geq t] > 0 \,\}$. We similarly show that the supremum is achieved in $t_{\min}(\pi)$.)

We easily see that, for a given adversary $\mathcal{A}$, we can limit the range of summation of the series $\sum_{t\geq 1} P_{\pi,\mathcal{A}}[T \geq t]$ to within the first $t_{\max}(\mathcal{A})$ terms. For a given protocol π, the interpretation of $t_{\min}(\pi)$ is in general a bit more complicated: if $\inf_{\mathcal{A}} P_{\pi,\mathcal{A}}[T \geq t_{\min}(\pi)+1] = \min_{\mathcal{A}} P_{\pi,\mathcal{A}}[T \geq t_{\min}(\pi)+1] (= 0)$, then there exists an adversary $\mathcal{A}$ for which only $t_{\min}(\pi)$ terms of the series $\sum_{t\geq 1} P_{\pi,\mathcal{A}}[T \geq t]$ are non-zero.

Note that all the values $t_{\max}(\mathcal{A})$ are a priori bounded by p. Hence Lemma 8.2.4 shows that the quantities $t_{\max}(\mathcal{A})$ and $t_{\min}(\pi)$ satisfy the max-min equality

$$\max_\pi t_{\min}(\pi) \leq \min_{\mathcal{A}} t_{\max}(\mathcal{A}) \,.$$

In this section we show that we can strengthen this inequality into an equality in the special case where $m = 1$. (See Corollary 7.6.9.) Lemma 7.6.8 will also be useful in the sequel of this work.

Let $t, t \leq p$ be a time, σ be a t-schedule and ϕ be a t-sequence of faults adapted to σ. We let $t_{\sigma,\phi}$ be the survival time of schedule σ used in conjunction with ϕ. By definition, $t_{\sigma,\phi}$ is less then equal to t.

Lemma 7.6.3 *Let $t, t \leq p$ be a time, σ be a t-schedule and ϕ be a t-sequence of faults adapted to σ such that $t_{\sigma,\phi} = t$. Then $\mathcal{A}_0 \in Agenerating_\sigma(\phi)$.*

PROOF. This a direct consequence of the definition of $\mathcal{A}_0$. At each time, $\mathcal{A}_0$ selects with non zero probability every element non already selected. Hence $\mathcal{A}_0$ selects σ with non-zero probability. □

Lemma 7.6.4 *Let $t, t \leq p$ be a time, σ be a t-schedule and ϕ be a t-sequence of faults adapted to σ. Then $P_{\pi,\mathcal{A}}[\, T \geq t, \mathcal{S}_t = \sigma, \mathcal{F}_t = \phi\,] > 0$ if and only if $t_{\sigma,\phi} \geq t$, $\pi \in Pgenerating(\sigma)$ and $\mathcal{A} \in Agenerating_\sigma(\phi)$.*

PROOF. This is a direct consequence of the fact that, conditioned on $\mathcal{S}_t = \sigma$ and $\mathcal{F}_t = \phi$, $T \geq t$ is equivalent to $t_{\sigma,\phi} \geq t$. □

Lemma 7.6.5 *For every $t > p - n + 1$ there is no t-schedule σ and no t-fault sequence ϕ such that $t_{\sigma,\phi} \geq t$.*

PROOF. Working by contradiction, consider $t > p-n+1$ and assume the existence of such a schedule σ and of such a fault sequence ϕ. Consider the point just after the selection of the t-th fault f_t by the adversary. The hypothesis $t_{\sigma,\phi} \geq t$ implies that, at this point, s_t contains $n-1$ non-faulty elements. On the other hand the t elements $f_1, \ldots, f_t$ are faulty. Hence we must have $p \geq t + n - 1$ which is a contradiction. □

Lemma 7.6.6 *Let t be a time and π a protocol. Then $P_{\pi,\mathcal{A}}[T \geq t] > 0$ only if $P_{\pi,\mathcal{A}_0}[T \geq t] > 0$.*

PROOF. The condition $P_{\pi,\mathcal{A}}[T \geq t] > 0$ implies that there must exist a t-schedule σ and a t-sequence of faults ϕ, adapted to σ, such that $P_{\pi,\mathcal{A}}[\, T \geq t, \mathcal{S}_t = \sigma, \mathcal{F}_t = \phi\,] > 0$. By the only if part of Lemma 7.6.4, it must then be the case that $t_{\sigma,\phi} \geq t$, that $\pi \in Pgenerating(\sigma)$ and that $\mathcal{A} \in Agenerating_\sigma(\phi)$. By Lemma 7.6.3, $\mathcal{A}_0 \in Agenerating_\sigma(\phi)$. By the if part of Lemma 7.6.4, $P_{\pi,\mathcal{A}_0}[\, T \geq t, \mathcal{S}_t = \sigma, \mathcal{F}_t = \phi\,] > 0$ and hence $P_{\pi,\mathcal{A}_0}[\, T \geq t\,] > 0$. □

In the following lemma, $l_0(t)$ is the value related to σ as defined in Definition 7.5.1.

Lemma 7.6.7 *Let $t, t \leq p$ be a time and π a protocol. Then $t \leq t_{\min}(\pi)$ if and only if there exists a t-schedule σ such that $l_0(t) = 0$ and such that $P_\pi[\, \mathcal{S}_t = \sigma\,] > 0$.*

PROOF. **If** part.

$$\begin{aligned} P_{\pi,\mathcal{A}_0}[\, T \geq t\,] &\geq P_{\pi,\mathcal{A}_0}[\, T \geq t, \mathcal{S}_t = \sigma\,] \\ &= P_{\mathcal{A}_0}[\, T \geq t \mid \mathcal{S}_t = \sigma\,]\, P_\pi[\, \mathcal{S}_t = \sigma\,]. \end{aligned}$$

By assumption, the term $P_\pi[\,\mathcal{S}_t = \sigma\,]$ is positive. On the other hand, by Corollary 7.6.2, the assumption $l_0(t) = 0$ implies that $\sigma \in \mathrm{Feas}_{\mathcal{A}_0}$ i.e., that $P_{\mathcal{A}_0}[\,T \geq t \mid \mathcal{S}_t = \sigma\,] > 0$. Hence

$$\inf_{\mathcal{A}} P_{\pi,\mathcal{A}}[\,T \geq t\,] \geq P_{\pi,\mathcal{A}_0}[\,T \geq t\,] > 0\,.$$

This inequality, along with the definition of $t_{\min}(\pi)$ shows that $t \leq t_{\min}(\pi)$.

Only if part. By definition of $t_{\min}(\pi)$, there exists an adversary $\mathcal{A}$ such that $P_{\pi,\mathcal{A}}[\,T \geq t\,] > 0$. By Lemma 7.6.6, this implies that $P_{\pi,\mathcal{A}_0}[\,T \geq t\,] > 0$. Hence there must exist a t-schedule σ such that $P_{\pi,\mathcal{A}_0}[\,T \geq t, \mathcal{S}_t = \sigma\,] > 0$. This implies that $P_{\pi,\mathcal{A}_0}[\,\mathcal{S}_t = \sigma\,] > 0$ and that $\sigma \in \mathrm{Feas}_{\mathcal{A}_0}$ i.e., by Corollary 7.6.2, that $l_0(t) = 0$. □

Lemma 7.6.8 *For all $\mathcal{A}$, for all π_0 in $Prot(Prog_0)$, $t_{\min}(\pi_0) = t_{\max}(\mathcal{A}) = p - n + 1$.*

Proof. For every adversary $\mathcal{A}$ and every protocol π, the inequality $t_{\min}(\pi) \leq t_{\max}(\mathcal{A})$ is true for any $m \geq 1$ and stems from the general result of Lemma 8.2.4 We prove that the converse inequality holds in the special case where $m = 1$ and where the protocol is in $Prot(Prog_0)$.

Let $t \leq p - n + 1$ and let $\pi_0 \in Prot(Prog_0)$. Every t-schedule σ selected by π_0 is such that $l_0(t) = 0$: by invariants **g** and **h** of Lemma 7.4.1, for every $j, 1 \leq j \leq t-1$, the set $c_j(t)$ has size at least $\min(n, \lfloor \frac{p-n}{t-1} \rfloor) \geq 1$. This fact together with Lemma 7.6.7 implies that $t_{\min}(\pi_0) \geq p - n + 1$. On the other hand, Lemmas 7.6.5 and 7.6.4 imply that $P_{\pi,\mathcal{A}}[\,T > p - n + 1\,] = 0$ for every protocol π and every adversary $\mathcal{A}$. This fact implies that $t_{\min}(\pi) \leq p - n + 1$ and that $t_{\max}(\mathcal{A}) \leq p - n + 1$ for every π and $\mathcal{A}$. Hence, for every $\mathcal{A}$, $t_{\max}(\mathcal{A}) \leq p - n + 1 = t_{\min}(\pi_0)$. This, along with the inequality $t_{\min}(\pi_0) \leq t_{\max}(\mathcal{A})$ previously established implies the equality $t_{\min}(\pi_0) = t_{\max}(\mathcal{A}) = p - n + 1$, valid for every $\mathcal{A}$. □

Corollary 7.6.9 $\max_\pi t_{\min}(\pi) = \min_{\mathcal{A}} t_{\max}(\mathcal{A})$.

Proof. By Lemma 8.2.4, $\max_\pi t_{\min}(\pi) \leq \min_{\mathcal{A}} t_{\max}(\mathcal{A})$. Conversely, let π_0 be any protocol in $Prot(Prog_0)$. We have:

$$\begin{aligned} \min_{\mathcal{A}} t_{\max}(\mathcal{A}) &= t_{\min}(\pi_0) \qquad \text{(by Lemma 7.6.8)} \\ &\leq \max_{\pi} t_{\min}(\pi)\,. \end{aligned}$$

It follows that $\max_\pi t_{\min}(\pi) = \min_{\mathcal{A}} t_{\max}(\mathcal{A})$. □

7.6.4 Schedules in normal form are most efficient against $\mathcal{A}_0$

The next lemma is crucial to evaluate the performance of a schedule. In essence, for a given schedule σ, it allows to compute for each time t the probability that the system survives one more time when following the schedule σ, provided that the system survived thus far and that the adversary is $\mathcal{A}_0$. It will allow us in Lemma 7.6.11 to characterize the schedules that perform best against $\mathcal{A}_0$.

Lemma 7.6.10 *For every $t \geq 1$ and every t-schedule σ the two following equalities hold:*

$$P_{\mathcal{A}_0}\Big[\, T \geq t \,\Big|\, \mathcal{S}_t = \sigma, T \geq t-1 \Big] = \prod_{j=1}^{t-1} \frac{|c_j(t)|}{|c_j(t-1)|} \tag{7.13}$$

$$P_{\mathcal{A}_0}\Big[\, T \geq t \,\Big|\, \mathcal{S}_t = \sigma \Big] = \prod_{j=1}^{t-1} \frac{|c_j(t)|}{n} = \prod_{i=0}^{n} \frac{i^{l_i(t)}}{n} \;, \tag{7.14}$$

where $c_j(u)$, $1 \leq j \leq u \leq t$ are uniquely derived from σ and where, by convention, we set 0^0 to be equal to 1 and $0/0$ to be equal to 0.

Corollary 7.3.5, page 148, justifies the well-foundedness of the probabilities computed in Lemma 7.6.10.

Proof. Consider first the case where $\sigma \notin \mathrm{Feas}_{\mathcal{A}_0}$. By Remark 1 on page 149, the left-hand side $P_{\mathcal{A}_0}[\, T \geq t \,|\, \mathcal{S}_t = \sigma]$ of Equation 7.14 is equal to 0. As

$$P_{\mathcal{A}_0}\Big[\, T \geq t \,\Big|\, T \geq t-1, \mathcal{S}_t = \sigma \Big] = \frac{P_{\mathcal{A}_0}[\, T \geq t,\, \mathcal{S}_t = \sigma]}{P_{\mathcal{A}_0}[\, T \geq t-1,\, \mathcal{S}_t = \sigma]} \;,$$

the left hand side of Equation 7.13 is also 0. (Note that, by Convention 8.1.1, the left-hand side $P_{\mathcal{A}_0}[\, T \geq t \,|\, \mathcal{S}_t = \sigma, T \geq t-1]$ is automatically set to 0 in the special case where $P_{\mathcal{A}_0}[\, T \geq t-1, \mathcal{S}_t = \sigma] = 0$. On the other hand, in this case, the convention $0/0 = 0$ also gives 0 to the right-hand side.) On the other hand, by Corollary 7.6.2, the condition $\sigma \notin \mathrm{Feas}_{\mathcal{A}_0}$ implies that $l_0(t) > 0$, i.e., that there exists $j, 1 \leq j \leq t-1$ such that $|c_j(t)| = 0$ and hence that $\prod_{j=1}^{t-1} |c_j(t)| = 0$. This implies that the right hand side of both Equations 7.13 and 7.14 is equal to zero.

We can therefore restrict ourselves in the sequel to the case where $\sigma = (s_1, \ldots, s_t) \in \mathrm{Feas}_{\mathcal{A}_0}$.

a. We begin with the proof of Equation 7.13.

$$
\begin{aligned}
& P_{\mathcal{A}_0}\Big[\, T \geq t \,\Big|\, \mathcal{S}_t = \sigma, T \geq t-1 \Big] \\
&= P_{\mathcal{A}_0}\Big[\cap_{j=1}^{t-1} \{F_j \in s_t^c\} \,\Big|\, \mathcal{S}_{t-1} = (s_1, \ldots, s_{t-1}), T \geq t-1 \Big] && (7.15) \\
&= \prod_{j=1}^{t-1} \mathcal{U}_{c_j(t-1)}[s_t^c] && (7.16) \\
&= \prod_{j=1}^{t-1} \frac{|s_t^c \cap c_j(t-1)|}{|c_j(t-1)|} && (7.17) \\
&= \prod_{j=1}^{t-1} \frac{|c_j(t)|}{|c_j(t-1)|} \ . && (7.18)
\end{aligned}
$$

Equation 7.15 is a consequence of Lemma 7.3.6. Equation 7.16 is a simple application of Lemma 7.6.1. Equation 7.17 is a simple application of the fact that $\mathcal{U}_{c_j(t-1)}$ is the uniform distribution on $c_j(t-1)$. Equation 7.18 comes from the definition of $c_j(t)$. This finishes to establish Equation 7.13.

b. We now turn to the proof of Equation 7.14.

$$
\begin{aligned}
& P_{\mathcal{A}_0}\Big[\, T \geq t \mid \mathcal{S}_t = (s_1, \ldots, s_t) \Big] \\
&= P_{\mathcal{A}_0}\Big[\, T \geq t, T \geq t-1, \ldots, T \geq 1 \mid \mathcal{S}_t = (s_1, \ldots, s_t) \Big] && (7.19) \\
&= P_{\mathcal{A}_0}\Big[\, T \geq t \mid T \geq t-1,\ \mathcal{S}_t = (s_1, \ldots, s_t) \Big] \\
&\quad \cdot P_{\mathcal{A}_0}\Big[\, T \geq t-1 \mid T \geq t-2,\ \mathcal{S}_t = (s_1, \ldots, s_t) \Big] && (7.20) \\
&\quad \ldots\ P_{\mathcal{A}_0}\Big[\, T \geq 2 \mid T \geq 1,\ \mathcal{S}_t = (s_1, \ldots, s_t) \Big] \cdot P_{\mathcal{A}_0}\Big[\, T \geq 1 \mid\ \mathcal{S}_t = (s_1, \ldots, s_t) \Big] \\
&= P_{\mathcal{A}_0}\Big[\, T \geq t \mid T \geq t-1,\ \mathcal{S}_t = (s_1, \ldots, s_t) \Big] \\
&\quad \cdot P_{\mathcal{A}_0}\Big[\, T \geq t-1 \mid T \geq t-2,\ \mathcal{S}_{t-1} = (s_1, \ldots, s_{t-1}) \Big] \\
&\quad \ldots\ P_{\mathcal{A}_0}\Big[\, T \geq 2 \mid T \geq 1, \mathcal{S}_2 = (s_1, s_2) \Big] \cdot P_{\mathcal{A}_0}\Big[\, T \geq 1 \mid \mathcal{S}_1 = s_1 \Big] && (7.21) \\
&= \prod_{u=2}^{t} \prod_{j=1}^{u-1} \frac{|c_j(u)|}{|c_j(u-1)|} && (7.22) \\
&= \prod_{j=1}^{t-1} \prod_{u=j+1}^{t} \frac{|c_j(u)|}{|c_j(u-1)|}
\end{aligned}
$$

$$= \prod_{j=1}^{t-1} |c_j(t)|/|c_j(j)|$$
$$= \prod_{j=1}^{t-1} |c_j(t)|/n \ .$$

Equation 7.19 is justified by the equality $\{T \geq t\} = \{T \geq t, T \geq t-1, \ldots, T \geq 1\}$. Equation 7.20 is obtained by successive conditioning. Equation 7.21 is a consequence of Lemma 7.3.6. Equation 7.22 is a consequence of Equation 7.13 and of the simple property $P_{\mathcal{A}_0}[\, T \geq 1 \mid \mathcal{S}_1 = s_1] = 1$. □

The following lemma establishes that, for every time $t, 1 \leq t \leq t_{\max}(\mathcal{A}_0)$, the set of t-schedules σ maximizing the probability $P_{\mathcal{A}_0}[\, T \geq t \,|\, \mathcal{S}_t = \sigma]$ is equal to the set $\mathcal{N}_t$ of t-schedules σ in normal form.

Lemma 7.6.11 *For any $t, 1 \leq t \leq t_{\max}(\mathcal{A}_0)$, the value $P_{\mathcal{A}_0}[\, T \geq t \,|\, \mathcal{S}_t = \sigma]$ is the same for all t-schedules σ in $\mathcal{N}_t$. We let $P_{\mathcal{A}_0}[\, T \geq t \,|\, \mathcal{S}_t \in \mathcal{N}_t]$ denote this common value. Furthermore, let σ' be a t-schedule. We have:*
1. *If $\sigma' \in \mathcal{N}_t$, then*

$$P_{\mathcal{A}_0}[\, T \geq t \,|\, \mathcal{S}_t = \sigma'] \quad = \quad \max_{\sigma} P_{\mathcal{A}_0}[\, T \geq t \,|\, \mathcal{S}_t = \sigma] \,. \tag{7.23}$$

2. *Conversely, if $\sigma' \notin \mathcal{N}_t$, then*

$$P_{\mathcal{A}_0}[\, T \geq t \,|\, \mathcal{S}_t = \sigma'] \quad < \quad \max_{\sigma} P_{\mathcal{A}_0}[\, T \geq t \,|\, \mathcal{S}_t = \sigma] \,. \tag{7.24}$$

Proof. By Lemma 7.6.10, that, for every t-schedule σ,

$$P_{\mathcal{A}_0}[\, T \geq t \,|\, \mathcal{S}_t = \sigma] = \prod_{j=1}^{t-1} \frac{|c_j(t)|}{n} = \prod_{i=0}^{n} \frac{i^{l_i(t)}}{n} \ .$$

A first consequence of this fact is that the value $P_{\mathcal{A}_0}[\, T \geq t \,|\, \mathcal{S}_t = \sigma]$ depends on σ only through the values $(l_0(t), \ldots, l_n(t))$. By Lemma 7.5.4, there is a unique and well defined sequence $(l_0(t), \ldots, l_n(t))$ to which all schedules in $\mathcal{N}_t$ are associated. This implies that the probability $P_{\mathcal{A}_0}[\, T \geq t \,|\, \mathcal{S}_t = \sigma]$ takes only one value when σ varies in $\mathcal{N}_t$. This fact justifies the notation

$$P_{\mathcal{A}_0}[\, T \geq t \,|\, \mathcal{S}_t \in \mathcal{N}_t] = P_{\mathcal{A}_0}[\, T \geq t \,|\, \mathcal{S}_t = \sigma'] \,,$$

for any arbitrary schedule σ' in $\mathcal{N}_t$.

Taking into account that all schedules $\sigma' \in \mathcal{N}_t$ give rise to the same value $P_{\mathcal{A}_0}[\, T \geq t \,|\, \mathcal{S}_t = \sigma']$, we see that Equation 7.23 is a consequence of Equation 7.24 that we now set out to prove. By definition of $t_{\max}(\mathcal{A}_0)$, the condition $t \leq t_{\max}(\mathcal{A}_0)$ implies that $0 < \sup_\pi P_{\pi,\mathcal{A}_0}[\, T \geq t \,]$, so that, by Lemmas 8.1.4 and 7.3.2

$$\begin{aligned} 0 &< \sup_\pi P_{\pi,\mathcal{A}_0}[\, T \geq t \,] \\ &\leq \sup_\pi \max_\sigma P_{\pi,\mathcal{A}_0}[\, T \geq t \,|\, \mathcal{S}_t = \sigma] \\ &= \max_\sigma \sup_\pi P_{\pi,\mathcal{A}_0}[\, T \geq t \,|\, \mathcal{S}_t = \sigma] \\ &= \max_\sigma P_{\mathcal{A}_0}[\, T \geq t \,|\, \mathcal{S}_t = \sigma] \ . \end{aligned} \tag{7.25}$$

Working by contradiction, assume that there exists a t-schedule σ not in normal form but which maximizes the probability $P_{\mathcal{A}_0}[\, T \geq t \,|\, \mathcal{S}_t = \sigma]$. The non-normality of σ implies that

a. either there exists j_1 and j_2 in $\{1, \ldots, t-1\}$ such that $|c_{j_1}(t)| \leq |c_{j_2}(t)| - 2$,

b. or $c_0(t) \neq \emptyset$ and there exists j in $\{1, \ldots, t-1\}$ such that $|c_j(t)| \leq n - 1$.

We first consider case **a.** In this case:

$$\begin{aligned} P_{\mathcal{A}_0}[\, T \geq t \,|\, \mathcal{S}_t = \sigma] &= \prod_{j=1}^{t-1} \frac{|c_j(t)|}{n} \\ &= \prod_{j \in \{1,\ldots,t-1\}; j \neq j_1, j_2} \frac{|c_j(t)|}{n} \cdot \frac{|c_{j_1}(t)|}{n} \cdot \frac{|c_{j_2}(t)|}{n} \ . \end{aligned}$$

Define the sequence $(\gamma_1, \ldots, \gamma_{t-1})$ derived from the sequence $(|c_1(t)|, \ldots, |c_{t-1}(t)|)$ by replacing $|c_{j_1}(t)|$ by $|c_{j_1}(t)| + 1$ and $|c_{j_2}(t)|$ by $|c_{j_2}(t)| - 1$. Note that

$$\sum_{j=1}^{t-1} \gamma_j = \sum_{j=1}^{t-1} |c_j(t)| \ . \tag{7.26}$$

By the only-if direction of Lemma 7.5.2 we have that $n \leq p - (|c_1(t)| + \ldots + |c_{t-1}(t)|)$. By Equation 7.26 we therefore also have $n \leq p - (\gamma_1 + \ldots + \gamma_{t-1})$. By the if direction of Lemma 7.5.2 there exists a schedule $\sigma' = (s'_1, \ldots, s'_t)$ whose associated c-sequence $(c'_1(t), \ldots, c'_{t-1}(t))$ satisfies $|c'_j(t)| = \gamma_j, 1 \leq j \leq t-1$. We compute:

$$\begin{aligned} (|c_{j_1}(t)| + 1) \cdot (|c_{j_2}(t)| - 1) &= |c_{j_1}(t)| \cdot |c_{j_2}(t)| + |c_{j_2}(t)| - |c_{j_1}(t)| - 1 \\ &\geq |c_{j_1}(t)| \cdot |c_{j_2}(t)| + 1 \ , \end{aligned}$$

because, by assumption, $|c_{j_1}(t)| \leq |c_{j_2}(t)| - 2$.

Then

$$\begin{aligned}
& P_{\mathcal{A}_0}[T \geq t \,|\, \mathcal{S}_t = \sigma'] \\
&= \prod_{j=1}^{t-1} \frac{|c'_j(t)|}{n} \\
&= \prod_{j \in \{1,\ldots,t-1\}; j \neq j_1, j_2} \frac{|c_j(t)|}{n} \cdot \frac{|c_{j_1}(t)+1|)}{n} \cdot \frac{|c_{j_2}(t)-1|)}{n} \\
&> \prod_{j \in \{1,\ldots,t-1\}; j \neq j_1, j_2} \frac{|c_j(t)|}{n} \cdot \frac{|c_{j_1}(t)|}{n} \cdot \frac{|c_{j_2}(t)|}{n} \qquad (7.27) \\
&= P_{\mathcal{A}_0}[T \geq t \,|\, \mathcal{S}_t = \sigma] \ .
\end{aligned}$$

In Equation 7.27, the inequality is strict because, by Equation 7.25 and the fact that

$$P_{\mathcal{A}_0}[\, T \geq t \,|\, \mathcal{S}_t = \sigma] = \prod_{j=1}^{t-1} \frac{|c_j(t)|}{n},$$

(see Equation 7.14), all the terms $|c_j(t)|$, $1 \leq j \leq t-1$ must be non-zero.

But this contradicts the fact that, by assumption, the schedule σ maximizes the probability $P_{\mathcal{A}_0}[\, T \geq t \,|\, \mathcal{S}_t = \sigma]$. This concludes the analysis of case **a.**

Case **b** is similar but easier. We could proceed as in **a** and use Lemma 7.5.2 to prove non-constructively the existence of an improving schedule. Just for the sake of enjoyment, we adopt another another proof technique and provide an explicit modification of the schedule σ.

By assumption $c_0(t) \neq \emptyset$ and there exists j_0 in $\{1, \ldots, t-1\}$ such that $|c_{j_0}(t)| \leq n-1$. Note that the sequence $(c_{j_0}(u))_{u \geq j_0}$ is a non-increasing sequence of sets such that $|c_{j_0}(j_0)| \stackrel{\text{def}}{=} |s(j_0)| = n$. Hence there must exist a time $j_1, j_0 < j_1 \leq t$ such that $|c_{j_0}(j_1)| < |c_{j_0}(j_1)| - 1$. Consider the smallest such j_1. By the definition of the set $c_{j_0}(j_1)$, (see Definition 7.5.1), this implies the existence of an element $x \in s(j_0) \cap s(j_1)$. Let y be any element in $c_0(t)$. Define

$$s'(j_1) \stackrel{\text{def}}{=} (s(j_1) - \{x\}) \cup \{y\} \ ,$$

and $s'(j) = s(j)$ for all $j \neq j_1$. Let $c'_1(t), \ldots, c'_{t-1}(t)$ be its associated c-sequence. We easily check that $|c'_{j_0}(t)| = |c_{j_0}(t)| + 1$ and that $c'_j(t) = c_j(t)$ for all j, $1 \leq j \leq t, j \neq j_0$.

Hence

$$\begin{aligned} & P_{\mathcal{A}_0}[T \geq t \,|\mathcal{S}_t = (s'_1, \ldots, s'_t)] \\ &= \prod_{j=1}^{t-1} \frac{|c'_j(t)|}{n} \\ &= \prod_{j \in \{1,\ldots,t-1\}, j \neq j_0} \frac{|c_j(t)|}{n} \cdot \frac{|c_{j_0}(t)|+1}{n} \\ &> \prod_{j=1}^{t-1} \frac{|c_j(t)|}{n} \\ &= P_{\mathcal{A}_0}[T \geq t \,|\, \mathcal{S}_t = \sigma] \ . \end{aligned}$$

As in case **a**, this contradicts the assumption that the schedule σ maximizes the probability $P_{\mathcal{A}_0}[\,T \geq t \,|\, \mathcal{S}_t = \sigma\,]$. This concludes the analysis of case **b.** □

7.6.5 Protocols in $Prot(Prog_0)$ are optimal against $\mathcal{A}_0$

Recall that the notation $P_{\mathcal{A}_0}[\,T \geq t \,|\, \mathcal{S}_t \in \mathcal{N}_t]$ was introduced and defined in Lemma 7.6.11.

Lemma 7.6.12 *For every t, $\sup_\pi P_{\pi,\mathcal{A}_0}[\,T \geq t\,] = P_{\mathcal{A}_0}[\,T \geq t \,|\, \mathcal{S}_t \in \mathcal{N}_t]$. Furthermore, for $t \leq t_{\max}(\mathcal{A}_0)$, a protocol π maximizes the probability $P_{\pi,\mathcal{A}_0}[\,T \geq t\,]$ if and only if $P_\pi[\,\mathcal{S}_t \in \mathcal{N}_t] = 1$.*

Proof.
• Let σ' be some schedule in $\mathcal{N}_t$. Let π' be a protocol such that $P_{\pi'}[\,\mathcal{S}_t = \sigma'] = 1$. Then

$$\begin{aligned} \sup_\pi P_{\pi,\mathcal{A}_0}[\,T \geq t\,] &\geq P_{\pi',\mathcal{A}_0}[\,T \geq t\,] \\ &= P_{\pi',\mathcal{A}_0}[\,T \geq t \ \mid \ \mathcal{S}_t = \sigma'] \\ &= P_{\mathcal{A}_0}[\,T \geq t \ \mid \ \mathcal{S}_t = \sigma'] \\ &= P_{\mathcal{A}_0}[\,T \geq t \,|\, \mathcal{S}_t \in \mathcal{N}_t]\,, \end{aligned}$$

where the last equality is a consequence of lemma 7.6.11.

• Conversely, let π be any protocol. The beginning of the following argument repeats the proof of Lemma 8.1.4.

$$P_{\pi,\mathcal{A}_0}[\,T \geq t\,] = \sum_\sigma P_{\pi,\mathcal{A}_0}[\,T \geq t \ \mid \ \mathcal{S}_t = \sigma] \cdot P_{\pi,\mathcal{A}_0}[\,\mathcal{S}_t = \sigma]$$

$$
\begin{aligned}
&= \sum_{\sigma} P_{\mathcal{A}_0}[\, T \geq t \;\mid\; \mathcal{S}_t = \sigma\,] \cdot P_{\pi}[\, \mathcal{S}_t = \sigma\,] \\
&\leq \max_{\sigma} P_{\mathcal{A}_0}[\, T \geq t \;\mid\; \mathcal{S}_t = \sigma\,] \cdot \sum_{\sigma} P_{\pi}[\, \mathcal{S}_t = \sigma\,] \qquad (7.28) \\
&= \max_{\sigma} P_{\mathcal{A}_0}[\, T \geq t \;\mid\; \mathcal{S}_t = \sigma\,] \\
&= P_{\mathcal{A}_0}[\, T \geq t \;\mid\; \mathcal{S}_t \in \mathcal{N}_t\,]\,,
\end{aligned}
$$

where here also the last equality is a consequence of lemma 7.6.11. Hence

$$\sup_{\pi} P_{\pi,\mathcal{A}_0}[\, T \geq t\,] \;\leq P_{\mathcal{A}_0}[\, T \geq t \,|\, \mathcal{S}_t \in \mathcal{N}_t]$$

which finishes to establish the first part of the lemma. Furthermore, note that the inequality 7.28 is an equality exactly when

$$P_{\pi}\Big[\, \mathcal{S}_t \in \Big\{\sigma \;;\sigma \text{ maximizes } P_{\mathcal{A}_0}[\, T \geq t \,|\, \mathcal{S}_t = \sigma\,]\Big\}\Big] = 1\,.$$

By Lemma 7.6.11, for $t \leq t_{\max}(\mathcal{A}_0)$, this happens exactly when $P_{\pi}[\, \mathcal{S}_t \in \mathcal{N}_t] = 1$. □

Lemma 7.6.13 *For every $\pi_0 \in Prot(Prog_0)$, for every $t, t \leq t_{\max}(\mathcal{A}_0)$, $P_{\pi_0}[\, \mathcal{S}_t \in \mathcal{N}_t] = 1$.*

PROOF. Let $\pi_0 \in Prot(Prog_0)$. By Lemma 7.6.8, $t_{\max}(\mathcal{A}_0) = t_{\min}(\pi_0) = p - n + 1$. Invariants **g** and **h** of Lemma 7.4.1 show that for every $t, t \leq p - n + 1$, π_0 only selects t-schedules in normal form. □

The next result expresses that, for all t, the protocols in $Prot(Prog_0)$ maximize the probability of survival up to time t when the adversary is the random adversary $\mathcal{A}_0$.

Corollary 7.6.14 *For every $\pi_0 \in Prot(Prog_0)$, for every t, $\max_{\pi} P_{\pi,\mathcal{A}_0}[T \geq t] = P_{\pi_0,\mathcal{A}_0}[T \geq t]$.*

PROOF. The equality is trivially true if $t > t_{\max}(\mathcal{A}_0)$: in this case both terms of the equation are equal to 0. On the other hand, if $t \leq t_{\max}(\mathcal{A}_0)$, the equality is a direct consequence of Lemmas 7.6.12 and 7.6.13. □

We can finally provide the proof of Lemma 7.4.7, stating that π_0 is optimal against adversary $\mathcal{A}_0$.

Corollary 7.6.15 *For every $\pi_0 \in Prot(Prog_0)$, $\max_\pi E_{\pi,\mathcal{A}_0}[T] = E_{\pi_0,\mathcal{A}_0}[T]$.*

PROOF. Let π_0 be an arbitrary protocol in $Prot(Prog_0)$. By Lemma 8.1.2, for every protocol π, $E_{\pi,\mathcal{A}_0}[T] = \sum_{t\geq 1} P_{\pi,\mathcal{A}_0}[T \geq t]$. Hence,

$$\begin{aligned} \sup_\pi E_{\pi,\mathcal{A}_0}[T] &= \sup_\pi \sum_{t\geq 1} P_{\pi,\mathcal{A}_0}[T \geq t] \\ &\leq \sum_{t\geq 1} \sup_\pi P_{\pi,\mathcal{A}_0}[T \geq t] \\ &= \sum_{t\geq 1} P_{\pi_0,\mathcal{A}_0}[T \geq t] \qquad (7.29) \\ &= E_{\pi_0,\mathcal{A}_0}[T] \ . \end{aligned}$$

Equality 7.29 is a consequence of Corollary 7.6.14. □

7.7 $\mathcal{A}_0$ is optimal against π_0

This section is devoted to the proof of Lemma (7.4.8). It uses in detail the code describing $Prog_0$ given in Page 151 to which the reader is referred.

Notations: We use the convention established in page 156 and for each program variable X let $X(t)$ denote the value held by X at the end of round t. Hence $J(t)$ is the value held by the program variable J at the end of round t. Recall also that $S_t(t) = S_t(t+1) = \ldots = S_t(n-p+1)$ so that we abuse language and let S_t also denote the value $S_t(t)$ held by the program variable S_t at the end of round t.

For every $t, 1 \leq t < p-n+1$ and every $k, 1 \leq k \leq t$, we let $\Delta_k(t+1)$ denote the value $|C_k(t)| - |C_k(t+1)|$. As usual, we use lower-case to denote realizations of a random variable and write $\delta_k(t+1)$ for a realization of $\Delta_k(t+1)$.

By invariants **g**, **h** and **i** given in Lemma 7.4.1, for every program *prog* in $Prog_0$, the value $\delta_k(t+1)$ of $\Delta_k(t+1)$ is uniquely determined for given values of $J(t)$ and $J(t+1)$, and hence for given values of $\mathcal{S}_t$ and $J(t+1)$.

Remember also that $\mathcal{U}_{\mathcal{P}_{\delta_k(t)}(c_k(t))}$ denotes the uniform distribution on the set $\mathcal{P}_{\delta_k(t)}(c_k(t))$ of subsets of $c_k(t)$ which are of size $\delta_k(t)$.

The next two lemmas, 7.7.1 and 7.7.2, characterize what is at each time t the probability distribution induced on $\mathcal{P}_n(p)$ by the choice S_{t+1} made for round $t+1$ by a program in $Prog_0$. This distribution depends on the t-schedule σ selected by the program in the first t rounds and on the value allocated to $J(t+1)$.

As discussed in Section 7.4.4, $J(t+1)$ is an *internal* variable of the program and is not measurable in the probability space $(\Omega, \mathcal{G})$ presented in Section 7.2. (In this space, only events describable in terms of the schedule $S_1, \ldots, S_p$ and in terms of the fault sequence $F_1, \ldots, F_p$ are measurable.) To be able to measure probabilistically the values allocated to this variable we therefore need to use the probability space $(\Omega', \mathcal{G}', P_{\text{prog}})$ defined in Section 7.4.4. In a figurative sense, programs in $Prog_0$ appear as black boxes when analyzed in the probability space $(\Omega, \mathcal{G})$: this probability space allows only to measure the output $(S_1, \ldots, S_p)$ of the programs. In contrast, the probability space $(\Omega', \mathcal{G}')$ allows to look inside each program *prog* in $Prot(Prog_0)$ and to analyze the internal actions it undertakes.

Lemma 7.7.1 *Let $t, \lfloor p/n \rfloor \leq t < p-n+1$, be arbitrary, let $prog \in Prog_0$, let σ be a t-schedule and $\mathcal{J}, \mathcal{J} \subseteq [t]$ such that $P_{\text{prog}}[\mathcal{S}_t = \sigma, J(t+1) = \mathcal{J}\,] > 0$. Then the*

random variables $S_{t+1} \cap c_k(t), 1 \leq k \leq t$ are independent and such that

$$P_{\text{prog}}\Big[\, S_{t+1} \cap c_k(t) = \cdot \;\Big|\; \mathcal{S}_t = \sigma,\, J(t+1) = \mathcal{J} \,\Big] = \mathcal{U}_{\mathcal{P}_{\delta_k(t+1)}(c_k(t))}\Big[\, \cdot \,\Big] \;,$$

where $\delta_k(t+1)$ and $c_k(t)$ are the values of $\Delta_k(t+1)$ and $C_k(t)$ uniquely determined by the conditions $\mathcal{S}_t = \sigma$ and $J(t+1) = \mathcal{J}$.

PROOF. Consider round $t+1$, and let σ denote the t-schedule $\mathcal{S}_t$ selected up to this point.

Conditioned on $\mathcal{S}_t = \sigma$, for every $k, 0 \leq k \leq t$, the program variable C_k is determined and equal to $c_k(t)$ – by definition of $c_k(t)$. Hence, in the randomized invocations $S := \texttt{uniform}(\alpha; c_K)$ or $S := \texttt{uniform}(\alpha+1; c_K)$ of lines 16, 22 and 35 of the code of $Prog_0$, the variable C_K is equal to $c_K(t)$. Note also that no further change is brought in round t to a variable C_K once one of the lines 18, 24 or 37 is executed. Hence the value allocated to S in these invocations is equal to $C_K(t+1)$. These two facts imply that the randomized invocation of line 16 can be rewritten $C_K(t+1) := \texttt{uniform}(\alpha; c_K(t))$, and that the randomized invocations of lines 22 and 35 can be rewritten $C_K(t+1) := \texttt{uniform}(\alpha+1; c_K(t))$.

By invariant **g** of Lemma 7.4.1, the sets $c_k(t), 0 \leq k \leq t$ form a partition of $[p]$ and hence are *disjoint*. Hence the random draws associated to the various randomized invocations $C_K(t+1) := \texttt{uniform}(\alpha; c_K(t))$ and $C_K(t+1) := \texttt{uniform}(\alpha+1; c_K(t))$ are *independent* and uniformly distributed. Therefore the random variables $c_k(t) - C_k(t+1)$, $0 \leq k \leq t$, are also *independent* of each other.

The value $\mathcal{J}$ of $J(t+1)$ determines precisely what are the values of $k, 1 \leq k \leq t$ for which $|C_K(t+1)| = \alpha + 1$ and what are the values of $k, 1 \leq k \leq t$ for which $|C_K(t+1)| = \alpha$. After **further conditioning** on $J(t+1) = \mathcal{J}$, each random variable $c_k(t) - C_k(t+1)$, is drawn uniformly from the set $\mathcal{P}_{\delta_k(t+1)}(c_k(t))$ of subsets of $c_k(t)$ which are of size $\delta_k(t+1)$. (Recall that the value $\delta_k(t+1)$ of $\Delta_k(t+1)$ is uniquely determined for the values σ of $\mathcal{S}_t$ and $\mathcal{J}$ of $J(t+1)$.) Hence, conditioned on $\{\mathcal{S}_t = \sigma,\, J(t+1) = \mathcal{J}\}$, the family $(c_0(t) - C_0(t+1), \ldots, c_t(t) - C_t(t+1))$ is distributed according to the product distribution

$$\bigotimes_{k=0}^{t} \mathcal{U}_{\mathcal{P}_{\delta_k(t+1)}(c_k(t))} \,.$$

On the one hand, by invariant **f** of Lemma 7.4.1, the random set S_{t+1} is equal to the disjoint union $\cup_{k=0}^{t}(c_k(t) \cap S_{t+1})$. On the other hand, by construction, (see the code in Page 151), it is equal to the disjoint union $\cup_{k=0}^{t}(c_k(t) - C_k(t+1))$. Hence,

for all $k, 0 \leq k \leq t$, $c_k(t) \cap S_{t+1}$ is equal to $c_k(t) - C_k(t+1)$. This (trivially) implies that the random variables $c_k(t) \cap S_{t+1}, 0 \leq k \leq t$, are independent and that, for every $k, 0 \leq k \leq t$, $c_k(t) \cap S_{t+1}$ has the same distribution as $c_k(t) - C_k(t+1)$. In particular, for every $k, 0 \leq k \leq t$,

$$P_{\text{prog}}\Big[\, c_k(t) \cap S_{t+1} = \cdot \;\Big|\; \mathcal{S}_t = \sigma,\, J(t+1) = \mathcal{J} \,\Big] = \mathcal{U}_{\mathcal{P}_{\delta_k(t+1)}(c_k(t))}\Big[\, \cdot \,\Big] .$$

□

Lemma 7.7.2 *Let $t, \lfloor p/n \rfloor \leq t < p - n + 1$, be arbitrary, let $prog \in Prog_0$, let σ be a t-schedule and $\mathcal{J}, \mathcal{J} \subseteq [t]$ such that $P_{\text{prog}}[\, \mathcal{S}_t = \sigma, J(t+1) = \mathcal{J}\,] > 0$. Then*

$$P_{\text{prog}}\Big[\, S_{t+1} = \;\cdot\; \Big|\; \mathcal{S}_t = \sigma,\, J(t+1) = \mathcal{J} \,\Big] = \prod_{k=0}^{t} \mathcal{U}_{\mathcal{P}_{\delta_k(t+1)}(c_k(t))}\Big[\, c_k(t) \cap \;\cdot\, \Big] ,$$

where $\delta_k(t+1)$ and $c_k(t)$ are the values of $\Delta_k(t+1)$ and $C_k(t)$ uniquely determined by the conditions $\mathcal{S}_t = \sigma$ and $J(t+1) = \mathcal{J}$.

Proof.

$$\begin{aligned}
& P_{\text{prog}}\Big[\, S_{t+1} = \cdot \;\Big|\; \mathcal{S}_t = \sigma,\, J(t+1) = \mathcal{J} \,\Big] \\
&= P_{\text{prog}}\Big[\, \cup_{k=0}^{t}(c_k(t) \cap S_{t+1}) = \cup_{k=0}^{t}(c_k(t) \cap \cdot\,) \;\Big|\; \mathcal{S}_t = \sigma,\, J(t+1) = \mathcal{J} \,\Big] \\
&= \prod_{k=0}^{t} P_{\text{prog}}\Big[\, c_k(t) \cap S_{t+1} = (c_k(t) \cap \cdot\,) \;\Big|\; \mathcal{S}_t = \sigma,\, J(t+1) = \mathcal{J} \,\Big] \qquad (7.30) \\
&= \prod_{k=0}^{t} \mathcal{U}_{\mathcal{P}_{\delta_k(t+1)}(c_k(t))}\Big[\, c_k(t) \cap \cdot \,\Big] . \qquad (7.31)
\end{aligned}$$

Equation 7.30 comes from the fact that the random variables $c_k(t) \cap S_{t+1}$ are independent. Equation 7.31 is a direct consequence of Lemma 7.7.1. □

The next lemma is a simple consequence of the preceding ones and expresses what is the probability that the set next selected by a protocol in $Prot(Prog_0)$ does not contain an element already faulty.

Lemma 7.7.3 *Let $t, \lfloor p/n \rfloor \leq t < p - n + 1$ and $j, 1 \leq j \leq t$ be arbitrary. Let $prog \in Prog_0$, let σ be a t-schedule and $\mathcal{J}, \mathcal{J} \subseteq [t]$ such that $P_{\text{prog}}[\, \mathcal{S}_t = \sigma, J(t+1) = \mathcal{J}\,] > 0$. Let $f_j \in c_j(t)$ be an arbitrary element. Then*

$$P_{\text{prog}}\Big[\, S_{t+1} \not\ni f_j \;\Big|\; \mathcal{S}_t = \sigma,\, J(t+1) = \mathcal{J} \,\Big] = \mathcal{U}_{\mathcal{P}_{\delta_j(t+1)}(c_j(t))}\Big[\, \cdot \not\ni f_j \,\Big] .$$

PROOF. Recall that for every k, $\delta_k(t+1)$ and $c_k(t)$ are the values of $\Delta_k(t+1)$ and $C_k(t)$ uniquely determined by the conditions $\mathcal{S}_t = \sigma$ and $J(t+1) = \mathcal{J}$.

By invariant **f** of Lemma 7.4.1, S_{t+1} is equal to the disjoint union $S_{t+1} = \cup_k (S_{t+1} \cap c_k(t))$. By assumption, f_j is in $c_j(t)$. Hence f_j is in S_{t+1} if and only if it is in $S_{t+1} \cap c_j(t)$. This justifies the first following equality. The second one is a direct consequence of Lemma 7.7.1. Hence

$$\begin{aligned} & P_{\text{prog}}\Big[S_{t+1} \not\ni f_j \;\Big|\; \mathcal{S}_t = \sigma,\, J(t+1) = \mathcal{J} \Big] \\ &= P_{\text{prog}}\Big[(S_{t+1} \cap c_j(t)) \not\ni f_j \;\Big|\; \mathcal{S}_t = \sigma,\, J(t+1) = \mathcal{J} \Big] \\ &= \mathcal{U}_{\mathcal{P}_{\delta_j(t+1)}(c_j(t))}\Big[\cdot \not\ni f_j \Big] \,. \end{aligned}$$

□

The following lemma is the crux of this section and reveals the role played by the uniform probability distributions used in the design of $Prog_0$. Particularly revelatory is the computation allowing to pass from Equation 7.36 to Equation 7.37.

Lemma 7.7.4 *Let $t, 1 \leq t < p - n + 1$. Then the value of the probability*

$$P_{\pi_0}\Big[T \geq t+1 \;\Big|\; T \geq t, \mathcal{S}_t = \sigma, \mathcal{F}_t = \phi \Big]$$

is the same for all protocols $\pi_0 \in Prot(Prog_0)$ [11], for all t-schedules $\sigma \in \mathcal{N}_t$ and for all t-fault-sequences ϕ adapted to σ such that $P_{\pi_0}\Big[T \geq t, \mathcal{S}_t = \sigma, \mathcal{F}_t = \phi \Big] > 0$.

PROOF. Throughout the proof we consider π_0, σ and ϕ such that $P_{\pi_0}[T \geq t, \mathcal{S}_t = \sigma, \mathcal{F}_t = \phi] > 0$ and we let *prog* be an element in $Prog_0$ such that $\pi_0 = \pi_{\text{prog}}$.

Recall that we let $f_1, \ldots, f_t$ denote the realizations of $F_1, \ldots, F_t$. Hence the condition $\mathcal{F}_t = \phi$ means that $(F_1, \ldots, F_t)$ is determined and equal to $(f_1, \ldots, f_t) = \phi$. Similarly, $\mathcal{S}_t = \sigma$ means that the t-schedule $(S_1, \ldots, S_t)$ is determined and equal to $(s_1, \ldots, s_t) = \sigma$. The random sets $C_j(u)$, $1 \leq j \leq u \leq t$ are then also uniquely determined and we let $c_j(u)$ denote their realizations. Recall that, by Lemma 7.5.1, $\{T \geq t\} = \cap_{k=1}^{t} \{F_k \in C_k(t)\}$. Hence, by conditioning on $\{T \geq t, \mathcal{S}_t = \sigma, \mathcal{F}_t = \phi\}$, we restrict ourselves to the case where f_k is in $c_k(t)$ for every $k, 1 \leq k \leq t$. This fact, along with the fact that the family $(c_k(t))_{1 \leq k \leq t}$ is a partition of $[p]$ implies that for every $S, S \subseteq [p]$, and every $k, 1 \leq k \leq t$, the condition $f_k \in S$ is equivalent to $f_k \in (S \cap c_k(t))$.

[11] See Page 150 for a definition of $Prot(Prog_0)$.

Case I. Assume that $t < \lfloor p/n \rfloor$. Then $P_{\pi_0}\Big[\, T \geq t+1 \;\Big|\; T \geq t,\, \mathcal{S}_t = \sigma, \mathcal{F}_t = \phi \,\Big] = 1$ and hence the result holds trivially.

Case II. Assume that $t \geq \lfloor p/n \rfloor$. We have:

$$
\begin{aligned}
& P_{\pi_0}\Big[\, T \geq t+1 \;\Big|\; T \geq t,\; \mathcal{S}_t = \sigma, \mathcal{F}_t = \phi \,\Big] \\
&= P_{\pi_0}\Big[\, S_{t+1} \not\ni F_1, \ldots, S_{t+1} \not\ni F_t \;\Big|\; T \geq t,\, \mathcal{S}_t = \sigma,\, \mathcal{F}_t = \phi \,\Big] \\
&= P_{\pi_0}\Big[\, S_{t+1} \not\ni f_1, \ldots, S_{t+1} \not\ni f_t \;\Big|\; T \geq t,\, \mathcal{S}_t = \sigma,\, \mathcal{F}_t = \phi \,\Big] \\
&= P_{\pi_0}\Big[\, \Big(S_{t+1} \cap c_1(t)\Big) \not\ni f_1, \ldots, \Big(S_{t+1} \cap c_t(t)\Big) \not\ni f_t \;\Big|\; T \geq t,\, \mathcal{S}_t = \sigma,\, \mathcal{F}_t = \phi \,\Big] && (7.32) \\
&= P_{\pi_0}\Big[\, \Big(S_{t+1} \cap c_1(t)\Big) \not\ni f_1, \ldots, \Big(S_{t+1} \cap c_t(t)\Big) \not\ni f_t \;\Big|\; \mathcal{S}_t = \sigma \,\Big] && (7.33) \\
&= \sum_{\mathcal{J}} P_{\text{prog}}\Big[\, \Big(S_{t+1} \cap c_1(t)\Big) \not\ni f_1, \ldots, \Big(S_{t+1} \cap c_t(t)\Big) \not\ni f_t \;\Big|\; \mathcal{S}_t = \sigma,\, J(t+1) = \mathcal{J} \,\Big] \\
&\qquad \cdot\; P_{\text{prog}}\Big[\, J(t+1) = \mathcal{J} \mid \mathcal{S}_t = \sigma \,\Big] && (7.34) \\
&= \sum_{\mathcal{J}} \prod_{k=1}^{t} P_{\text{prog}}\Big[\, \Big(S_{t+1} \cap c_k(t)\Big) \not\ni f_k \;\Big|\; \mathcal{S}_t = \sigma,\, J(t+1) = \mathcal{J} \,\Big] \\
&\qquad \cdot\; P_{\text{prog}}\Big[\, J(t+1) = \mathcal{J} \mid \mathcal{S}_t = \sigma \,\Big] && (7.35) \\
&= \sum_{\mathcal{J}} \prod_{k=1}^{t} \mathcal{U}_{\mathcal{P}_{\delta_k(t+1)}(c_k(t))}\Big[\, \cdot \not\ni f_k \,\Big]\, P_{\text{prog}}\Big[\, J(t+1) = \mathcal{J} \mid \mathcal{S}_t = \sigma \,\Big] && (7.36) \\
&= \sum_{\mathcal{J}} \prod_{k=1}^{t} \frac{|c_k(t+1)|}{|c_k(t)|}\, P_{\text{prog}}\Big[\, J(t+1) = \mathcal{J} \mid \mathcal{S}_t = \sigma \,\Big] && (7.37) \\
&= \frac{\prod_{k=1}^{t} |c_k(t+1)|}{\prod_{k=1}^{t} |c_k(t)|} \;\cdot\; \sum_{\mathcal{J}} P_{\text{prog}}\Big[\, J(t+1) = \mathcal{J} \mid \mathcal{S}_t = \sigma \,\Big] && (7.38) \\
&= \frac{\prod_{k=1}^{t} |c_k(t+1)|}{\prod_{k=1}^{t} |c_k(t)|} \;.
\end{aligned}
$$

Equation 7.32 stems from the fact that, as we saw at the beginning of this proof, for every $k, 1 \leq k \leq t$, f_k is an element of $c_k(t)$ and hence the condition $S_{t+1} \not\ni f_k$ is equivalent to $(S_{t+1} \cap c_k(t)) \not\ni f_k$.

We used in the beginning of the proof the formula $\{T \geq t\} = \cap_{k=1}^{t} \{F_k \in C_k(t)\}$. We now find it convenient to use the alternate expression $\{T \geq t\} = \cap_{u=1}^{t} \{S_u \cap \{F_1, \ldots, F_{u-1}\} = \emptyset\}$. (Both expressions are presented in Lemma 7.5.1). The event $\{\mathcal{F}_t = \phi, T \geq t\}$ is equal to the event $\{\mathcal{F}_t = \phi, \bigcap_{u=1}^{t} \{S_u \cap \{f_1, \ldots, f_{u-1}\} = \emptyset\}\}$

which,when conditioned on $\mathcal{S}_t = \sigma$, is itself equal to the event $\{\mathcal{F}_t = \phi, \bigcap_{u=1}^{t}\{s_u \cap \{f_1, \ldots, f_{u-1}\} = \emptyset\}\}$. Note that the relation $\bigcap_{u=1}^{t}\{s_u \cap \{f_1, \ldots, f_{u-1}\} = \emptyset\}$ only expresses a relation between the *deterministic* quantities $s_1, f_1, \ldots, s_t$. Hence the probabilistic event $\{\mathcal{F}_t = \phi, \bigcap_{u=1}^{t}\{s_u \cap \{f_1, \ldots, f_{u-1}\} = \emptyset\}\}$ is actually equal to $\{\mathcal{F}_t = \phi\}$. As, by Lemma 7.3.1, conditioned on $\mathcal{S}_t = \sigma$, the random variables $\mathcal{F}_t$ and S_{t+1} are independent, we therefore similarly have that, conditioned on $\mathcal{S}_t = \sigma$, the random variable S_{t+1} is independent from the event $\{\mathcal{F}_t = \phi, T \geq t\}$. This justifies Equation 7.33.

In Equation 7.34, we condition on the value taken by $J(t+1)$, i.e., on the outcome of an internal action of the program *prog* associated to π_0. As discussed at the beginning of this section, the random variable $J(t+1)$ is not measurable in the σ-field associated to protocols, and we must therefore consider the probability space $(\Omega', \mathcal{G}', P_{\text{prog}})$ allowing to measure the randomized invocations made by *prog*. This explains why we consider P_{prog} in place of P_{π_0}.

By Lemma 7.7.1 the random variables $S_{t+1} \cap c_k(t), 1 \leq k \leq t$ are independent. This justifies Equation 7.35. Using Lemma 7.7.3 along with the fact that $S_{t+1} \not\ni f_k$ is equivalent to $(S_{t+1} \cap c_k(t)) \not\ni f_k$ justifies Equation 7.36.

Note that we replaced in Equation 7.32 the conditions $S_{t+1} \not\ni f_k$ by the conditions $(S_{t+1} \cap c_k(t)) \not\ni f_k$ and that we replaced back each condition $(S_{t+1} \cap c_k(t)) \not\ni f_k$ by $S_{t+1} \not\ni f_k$ while establishing Equation 7.36. The reason for the introduction of the sets $c_k(t)$ is to be able to use the (conditional) independence of the random variables $S_{t+1} \cap c_k(t)$ and to obtain the product introduced in Equation 7.35.

Recall that we defined $\delta_k(t+1)$ to be equal to the value $|c_k(t)| - |c_k(t+1)|$. Hence $\mathcal{U}_{\mathcal{P}_{\delta_k(t+1)}(c_k(t))}[f_k \notin \cdot]$ is equal to $\mathcal{U}_{\mathcal{P}_{|c_k(t+1)|}(c_k(t))}[f_k \in \cdot]$ which is equal to $|c_k(t+1)|/|c_k(t)|$. This establishes Equation 7.37. Note that, for each k, the value $c_k(t+1)$ *does* depend on the values σ and $\mathcal{J}$ taken by S_t and $J(t+1)$. Nevertheless, by Lemma 7.4.5, the product $\prod_{k=1}^{t}|c_k(t+1)|$ (resp. $\prod_{k=1}^{t}|c_k(t)|$) is the same for all programs $prog \in Prog_0$ and all values σ, $\mathcal{J}$ and ϕ taken by $\mathcal{S}_t$, $J(t+1)$ and $\mathcal{F}_t$. We can therefore factor the term $\prod_{k=1}^{t}|c_k(t+1)|/\prod_{k=1}^{t}|c_k(t)|$ out of the summation over $\mathcal{J}$. This justifies Equation 7.38.

This establishes that $P_{\pi_0}[T \geq t+1 \mid T \geq t, \mathcal{S}_t = \sigma, \mathcal{F}_t = \phi]$ is the same for all protocols $\pi_0 \in Prot(Prog_0)$, all t-schedules $\sigma \in \mathcal{N}_t$ and all t-fault-sequences ϕ adapted to σ. □

Note that if $k \leq \lfloor p/n \rfloor$ then $c_k(u) = S_k$ for all times $u, k \leq u \leq \lfloor p/n \rfloor$. Therefore $|c_k(t+1)|/|c_k(t)| = 1$ for all t and k, $1 \leq k \leq t < \lfloor p/n \rfloor$.

Assume now that $t \geq p - n + 1$. Let σ be a t-schedule and let $s \in \mathcal{P}_n(p)$ be such

that (σ, s) is a $t+1$-schedule. Let π_0 be a protocol in $Prot(Prog_0) \cap Pgenerating(\sigma)$. By Lemma 7.6.8, $t_{\max}(\mathcal{A}_0) = p - n + 1$. By definition of $t_{\max}(\mathcal{A}_0)$ this implies that $P_{\mathcal{A}_0}[T \geq t+1 \mid \mathcal{S}_{t+1} = (\sigma, s)] = 0$ and by consequence that $(\sigma, s) \notin \text{Feas}_{\mathcal{A}_0}$. Hence, as is established at the beginning of the proof of Lemma 7.6.10, $\prod_{j=1}^{t} |c_j(t+1)| = 0$. Using the convention $0/0 = 0$ (see Convention 8.1.1, page 198) we see that $(\prod_{k=1}^{t} |c_k(t+1)|)/(\prod_{k=1}^{t} |c_k(t)|) = 0$. On the other hand, again by Lemma 7.6.8 and Convention 8.1.1, $P_{\pi_0}[\, T \geq t+1 \mid T \geq t,\, \mathcal{S}_t = \sigma,\, \mathcal{F}_t = \phi \,] = 0$ if $t > p - n + 1$.

This shows that the equality

$$P_{\pi_0}\Big[\, T \geq t+1 \;\Big|\; T \geq t,\, \mathcal{S}_t = \sigma,\, \mathcal{F}_t = \phi \,\Big] = \frac{\prod_{k=1}^{t} |c_k(t+1)|}{\prod_{k=1}^{t} |c_k(t)|}$$

holds for every $t, t \geq 1$.

This result, along with Formula 7.13 in Lemma 7.6.10 establishes the following remarkable fact:[12]

For every $t, t \geq 1$, every $t+1$-schedule $(\sigma, s) \in \mathcal{N}_{t+1}$, every t-fault-sequence ϕ adapted to σ and every protocol $\pi_0 \in Prot(Prog_0) \cap Pgenerating(\sigma)$:

$$\begin{aligned} & P_{\mathcal{A}_0}\Big[\, T \geq t+1 \mid T \geq t,\, \mathcal{S}_{t+1} = (\sigma, s) \,\Big] \\ &\quad = \prod_{j=1}^{t} \frac{|c_j(t+1)|}{|c_j(t)|} \\ &\quad = P_{\pi_0}\Big[\, T \geq t+1 \mid T \geq t,\, \mathcal{S}_t = \sigma,\, \mathcal{F}_t = \phi \,\Big] \end{aligned}$$

where the values $c_j(t), 1 \leq j \leq t$, and $c_j(t+1), 1 \leq j \leq t$, are the c-values related to the $(t+1)$-schedule (σ, s).

This result constitutes the core technical result of this chapter, whose consequences ultimately lead to the fundamental equality

$$\sup_{\pi} E_{\pi, \mathcal{A}_0}[T] = E_{\pi_0, \mathcal{A}_0}[T] = \inf_{\mathcal{A}} E_{\pi_0, \mathcal{A}}[T]\,,$$

establishing Theorem 7.4.6.

Lemma 7.7.5 *Let $t, 1 \leq t < p - n + 1$. Then, for every $\pi_0 \in Prot(Prog_0)$, the probability $P_{\pi_0, \mathcal{A}}[\, T \geq t+1 \mid T \geq t \,]$ is independent of the adversary $\mathcal{A}$.*

[12] Recall that, by convention, we set $0/0 = 0$.

PROOF.

$$\begin{aligned}
&P_{\pi_0,\mathcal{A}}\Big[\, T \geq t+1 \;\Big|\; T \geq t \,\Big] \\
&= \sum_{\sigma,\phi} P_{\pi_0}\Big[\, T \geq t+1 \;\Big|\; T \geq t,\, \mathcal{S}_t = \sigma,\, \mathcal{F}_t = \phi \,\Big]\; P_{\pi_0,\mathcal{A}}\Big[\, \mathcal{S}_t = \sigma, \mathcal{F}_t = \phi \,\Big] \\
&= P_{\pi_0}\Big[\, T \geq t+1 \;\Big|\; T \geq t,\, \mathcal{S}_t = \sigma,\, \mathcal{F}_t = \phi \,\Big]\; \sum_{\sigma,\phi} P_{\pi_0,\mathcal{A}}\Big[\, \mathcal{S}_t = \sigma, \mathcal{F}_t = \phi \,\Big] \qquad (7.39) \\
&= P_{\pi_0}\Big[\, T \geq t+1 \;\Big|\; T \geq t,\, \mathcal{S}_t = \sigma,\, \mathcal{F}_t = \phi \,\Big].
\end{aligned}$$

Equation 7.39 holds because, by Lemma 7.7.4, the value $P_{\pi_0}[\, T \geq t+1 \mid T \geq t,\, \mathcal{S}_t = \sigma,\, \mathcal{F}_t = \phi\,]$ is the same for all σ and ϕ (adapted to σ). Furthermore, the fact that the value $P_{\pi_0}[\, T \geq t+1 \mid T \geq t,\, \mathcal{S}_t = \sigma,\, \mathcal{F}_t = \phi\,]$ does not depend on ϕ immediately implies that it similarly does not depend on $\mathcal{A}$, which concludes the proof. □

Lemma 7.7.6 *Let $t, 1 \leq t \leq p - n + 1$. Then, for every $\pi_0 \in Prot(Prog_0)$, the probability $P_{\pi_0,\mathcal{A}}[\, T \geq t\,]$ is independent of the adversary $\mathcal{A}$.*

PROOF. $P_{\pi_0,\mathcal{A}}[T \geq t] = P_{\pi_0,\mathcal{A}}[T \geq t \mid T \geq t-1] \ldots P_{\pi_0,\mathcal{A}}[T \geq 2 \mid T \geq 1]$. The result then follows from Lemma 7.7.5. □

We are now finally able to prove Lemma (7.4.8).

Corollary 7.7.7 *For every $\pi_0 \in Prot(Prog_0)$, the expectation $E_{\pi_0,\mathcal{A}}[T]$ is independent of the adversary $\mathcal{A}$.*

PROOF.

$$\begin{aligned}
E_{\pi_0,\mathcal{A}}[T] &= \sum_{t \geq 1} P_{\pi_0,\mathcal{A}}[T \geq t] && (7.40) \\
&= \sum_{t=1}^{t_{\max}(\mathcal{A})} P_{\pi_0,\mathcal{A}}[T \geq t] && (7.41) \\
&= \sum_{t=1}^{t_{\min}(\pi_0)} P_{\pi_0,\mathcal{A}}[T \geq t] && (7.42)
\end{aligned}$$

Equality 7.40 is justified by Lemma 8.1.2. Equality 7.41 by the fact that $P_{\pi_0,\mathcal{A}}[T \geq t] = 0$ if $t > t_{\max}(\mathcal{A})$. By Lemma 7.6.8, for all $\mathcal{A}$, $t_{\max}(\mathcal{A}) = t_{\min}(\pi_0) = p - n + 1$. This

justifies Equality 7.42. By Lemma 7.7.6, for every $t, 1 \leq t \leq t_{\min}(\pi_0)$, each of the terms $P_{\pi_0,\mathcal{A}}[T \geq t]$ is independent of $\mathcal{A}$. Hence the whole sum $\sum_{t=1}^{t_{\min}(\pi_0)} P_{\pi_0,\mathcal{A}}[T \geq t]$ is independent of $\mathcal{A}$. □

Chapter 8

General Mathematical Results

8.1 Some Useful Notions in Probability Theory

Notation: 1) For every finite set $\Phi, \Phi \neq \emptyset$, we let $\mathcal{U}_\Phi$ denote the uniform distribution of the set Φ. We will find it convenient to extend this definition to $\Phi = \emptyset$ and let $\mathcal{U}_\emptyset(A)$ be identically equal to 0 for every set A. (Hence $\mathcal{U}_\emptyset$ is not a probability distribution.)
2) For every set X we let $\mathcal{P}(X)$ denote the power set of X. For every integer δ we let $\mathcal{P}_\delta(X)$ denote the set of subsets of X of size δ:

$$\mathcal{P}_\delta(X) \stackrel{\text{def}}{=} \{Y \subseteq X; |Y| = \delta\}.$$

Hence, $\mathcal{P}_0(X) = \{\emptyset\}$ and $\mathcal{P}_\delta(X) = \emptyset$ for all $\delta > |X|$. To simplify the notations we write $\mathcal{P}_n(p)$ in place of $\mathcal{P}_n([N])$.
3) For every integers k and l, $k \leq l$, we let $[l]$ denote the set $\{1, \ldots, l\}$ and let $\mathcal{P}_k(l) \stackrel{\text{def}}{=} \{s; s \subseteq [l], |s| = k\}$.

We can mix the two previous definitions and consider $\mathcal{U}_{\mathcal{P}_\delta(X)}$, the uniform distribution on $\mathcal{P}_\delta(X)$. The following lemma investigates some special cases associated to this situation.

Lemma 8.1.1 *1) $\mathcal{U}_{\mathcal{P}_\delta(X)} \equiv 0$ if $\delta > |X|$. 2) $\mathcal{U}_{\mathcal{P}_0(X)} = \delta_\emptyset$, the Dirac measure on $\emptyset$. Equivalently, $\mathcal{U}_{\mathcal{P}_0(X)}$ is the probability measure selecting the empty set with probability 1.*

Proof. Assume that $\delta > |X|$. Then, for every set A, $\mathcal{U}_{\mathcal{P}_\delta(X)}(A) = \mathcal{U}_\emptyset(A) = 0$. On the other hand, $\mathcal{U}_{\mathcal{P}_0(X)} = \mathcal{U}_{\{\emptyset\}}$ is the uniform distribution on the singleton set $\{\emptyset\}$, i.e., the distribution selecting $\emptyset$ with probability one. □

Convention 8.1.1 *Let $(\Omega, \mathcal{G}, P)$ be some probability space. For all A and B in $\mathcal{G}$ we set $P[B|A] = 0$ whenever $P[A] = 0$. We accordingly set to 0 the ratio $0/0$.*

Definition 8.1.1 *Let $(\Omega, \mathcal{G}, P)$ be a probability space. For every sets A and B in $\mathcal{G}$ we say that A and B are P-equivalent if $P[A\Delta B] = 0$, where $A\Delta B$ denotes the symmetric difference of A and B. An element A of $\mathcal{G}$ is called a P-atom if, for every B in $\mathcal{G}$, B is a subset of A only if B is P-equivalent to A or to $\emptyset$.*

Definition 8.1.2 *Let $(\Omega_1, \mathcal{G}_1, P_1)$ be some probability space and let $X : (\Omega_1, \mathcal{G}_1, P_1) \to (\Omega_2, \mathcal{G}_2)$ be a random variable. Then the* law *of X is the probability distribution on $(\Omega_2, \mathcal{G}_2)$ defined by:*

$$\forall B \in \mathcal{G}_2,\ P_2[B] \stackrel{\text{def}}{=} P_1[X \in B].$$

We write

$$\mathcal{L}(X) = P_2.$$

For $A \in \mathcal{G}_1$, $P[A] \neq 0$, the random variable $X|A$ is by definition a random variable X' whose law P_3 is given by:

$$\begin{aligned} \forall B \in \mathcal{G}_2,\ P_3[B] &= P_1[X' \in B] \\ &\stackrel{\text{def}}{=} P_1[X \in B \mid A]. \end{aligned}$$

Following Convention 8.1.1, we set $X|A \equiv 0$ if $P[A] = 0$.

Lemma 8.1.2 *Let T be any non-negative random variable defined on some probability space $(\Omega, \mathcal{G}, P)$. Then*

$$E[T] = \int_0^\infty P[T > t]\, dt\,.$$

If T is integer valued, the preceding translates into

$$E[T] = \sum_{t=1}^{\infty} P[T \geq t]\,.$$

PROOF. For every interval $[0, T[$ we let $\chi_{[0,T[}$ denote the function equal to 1 on $[0, T[$ and 0 elsewhere. Also, for simplicity of notation, we think of the expectation E as being an operator and let Ef denote $E[f]$.

$$ET = E \int_0^T dt \tag{8.1}$$

$$
\begin{aligned}
&= E\int_0^\infty \chi_{[0,T[}(t)\,dt \\
&= \int dP(x)\int_0^\infty \chi_{[0,T(x)[}(t)\,dt \qquad (8.2)\\
&= \int_0^\infty dt\int dP(x)\,\chi_{[0,T(x)[}(t) \qquad (8.3)\\
&= \int_0^\infty P[T>t]\,dt\,.
\end{aligned}
$$

Equation 8.1 comes from the simple observation that $T = \int_0^T dt$. Equation 8.2 is a particular case of the general formula $E[f] = \int f(x)dP(x)$. Equation 8.3 is an application of Fubini's theorem. In the case where T is integer valued,

$$
\begin{aligned}
E[T] &= \int_0^\infty P[T>t]\,dt \\
&= \sum_{n=1}\int_{n-1}^{n} P[T>t]\,dt \\
&= \sum_{n=1} P[T\geq n]\ .
\end{aligned}
$$

□

The following lemma states a well-known property of the uniform distribution.

Lemma 8.1.3 *Let A be a finite set and let B be a non empty subset of A. Then $\mathcal{U}_A[\ \cdot \mid B] = \mathcal{U}_{A\cap B}$.*

Lemma 8.1.4 *Let $(\Omega,\mathcal{G},P)$ be some probability space, let $S : (\Omega,\mathcal{G}) \to (E,\mathcal{P}(E))$ be a random variable defined on Ω. Then for every $B \in \mathcal{G}$,*

$$P[B] \leq \max_{s\in E} P[B \mid S = s].$$

Proof. $\mathcal{P}(E)$ being the σ-field attached to E the set $\{S = s\}$ is measurable for every s in E. Hence $P[B \mid S = s]$ is well-defined if $P[S = s] \neq 0$. On the other hand, by Convention 8.1.1, $P[B \mid S = s]$ is set to 0 when $P[S = s] = 0$. Hence $P[B \mid S = s]$ is well-defined for all s in E.

$$
\begin{aligned}
P[B] &= \sum_{s\in E} P[B \mid S = s]P[S = s] \\
&\leq \max_{s\in E} P[B \mid S = s]\sum_{s\in E} P[S = s] \\
&= \max_{s\in E} P[B \mid S = s].
\end{aligned}
$$

□

Lemma 8.1.5 *Let B and A be any real-valued random variables. Then*

$$\forall x \in \mathbb{R},\ P[\, B \geq x \mid B \leq A\,] \leq P[\, B \geq x\,].\ ^{1}$$

PROOF. The result is obvious if $P[\, B \leq A\,] = 0$. We can therefore restrict ourselves to the case $P[\, B \leq A\,] > 0$. Assume first that A is a constant a.

If $a < x$ the result is trivially true, so we assume that $a \geq x$. Set $\rho = P[B > a]$ and $\beta = P[B \geq x]$. Then:

$$\begin{aligned} P[\, B \geq x \mid B \leq a\,] &= \tfrac{P[\, x \leq B \leq a\,]}{P[B \leq a]} \\ &= \frac{\beta - \rho}{1 - \rho} \stackrel{\text{def}}{=} \phi_\beta(\rho). \end{aligned}$$

For $\beta \in [0,1]$, The function ϕ_β is not increasing on $[0,1)$. Hence:

$$\begin{aligned} P[\, B \geq x \mid B \leq a\,] &\leq \phi_\beta(0) \\ &= P[B \geq x\,]. \end{aligned}$$

We then turn to the case where A is a general random variable.[2] We will let dP_A denote the distribution of A so that, for any measurable set U, $dP_A[U] = P[A \in U]$. Then:

$$\begin{aligned} P[\, A \geq B \geq x\,] &= \int_a P[\, a \geq B \geq x\,]\, dP_A(a) \\ &= \int_a P[\, B \geq x \mid B \leq a\,]\, P[B \leq a]\, dP_A(a) \\ &\leq \int_a P[B \geq x]\, P[B \leq a]\, dP_A(a) \end{aligned}$$

(We use here the result valid for $A = a$ constant)

$$\begin{aligned} &= P[B \geq x] \int_a P[B \leq a]\, dP_A(a) \\ &= P[B \geq x]\, P[B \leq A]\ . \end{aligned}$$

Then we just need to divide by $P[B \leq A]$ to get the result we are after. □

[1] Recall that, using Convention 8.1.1, we set $0/0 = 0$ whenever this quantity arises in the computation of conditional probabilities.

[2] Note that we cannot simply extend the previous proof in this case: A can sometimes be less then x and then it is not true anymore that $\{B > A\} \subseteq \{x \leq B\}$. We used this when saying that $P[\, x \leq B \leq a\,] = P[x \leq B] - P[B > a]$.

If A is a discrete variable, its distribution is absolutely continuous with respect to the counting measure, so that the expression $\int_a P[a \geq B \geq x] dP_A(a)$ reduces to $\sum_a P[a \geq B \geq x] P[A = a]$.

We recall in the following definition the notion of stochastic (partial) ordering: this ordering is defined within the set of real random variables.

Definition 8.1.3 *Let X and Y be two real random variables. Then the following conditions are equivalent:*

1. *For all $x \in R$, $P[X \geq x] \leq P[Y \geq x]$.*
2. *For all continuous and bounded function f, $\int f(x) dP_X(x) \leq \int f(x) dP_Y(x)$ (i.e., $dP_X \leq dP_Y$).*

We then say that the X is stochastically smaller *then Y (or alternatively* smaller in law*) and we write that: $X \leq_{\mathcal{L}} Y$.*

Note that if $X \leq Y$ in the usual sense (i.e., almost surely), then X is also stochastically smaller then Y. Actually, among all the usual orders that are usually considered on the set of random variables (e.g., almost surely, for some L^p-norm, for the essential norm L^∞) the stochastic ordering is the weakest.

Formulated in this language, Lemma 8.1.5 just says that

$$(B \mid B \leq A) \leq_{\mathcal{L}} B.$$

The result of Lemma 8.1.5 can be generalized into:

Lemma 8.1.6 *Let A_1 , A_2 and B be any real-valued random variables.* [3] *Assume that $A_1 \leq_{\mathcal{L}} A_2$. Then*

$$(B \mid B \leq A_1) \leq_{\mathcal{L}} (B \mid B \leq A_2).$$

PROOF. The proof is similar to the one of Lemma 8.1.5. (Lemma 8.1.5 corresponds to the case $A_2 \equiv \infty$.) □

Lemma 8.1.7 *Let A, B and C be any real-valued random variables. Then*

$$P[B \geq C \mid B \leq A] \leq P[B \geq C].$$

[3] As is customary in measure theory, we allow the random variables to take the ∞ value.

PROOF. We integrate the inequality of Lemma 8.1.5:

$$P[\,B \geq C \mid B \leq A\,] = \int P[\,B \geq x \mid B \leq A\,]\, dP_C(x)$$
$$\leq \int P[B \geq x]\, dP_C(x) = P[B \geq C].$$

□

Lemma 8.1.8 *Let A_1 , A_2 and B be any real-valued random variables. Assume that $A_1 \leq_{\mathcal{L}} A_2$. Then*

$$P[A_1 \geq B] \leq P[A_2 \geq B].$$

PROOF. $P[A_1 \geq B] = \int P[A_1 \geq x]\, dP_B(x) \leq \int P[A_2 \geq x]\, dP_B(x) = P[A_2 \geq B]$.
□

8.2 Max-Min Inequalities

Lemma 8.2.1 *Let $f(x, y)$ be a reel function of two variables defined over a domain $X \times Y$. Then*

$$\sup_{x \in X} \inf_{y \in Y} f(x, y) \leq \inf_{y \in Y} \sup_{x \in X} f(x, y).$$

PROOF. Let x_0 and y_0 be two arbitrary elements in X and Y respectively. Obviously, $f(x_0, y_0) \leq \sup_x f(x, y_0)$. This inequality is true for every y_0 so that $\inf_y f(x_0, y) \leq \inf_y \sup_x f(x, y)$. Note that $\inf_y \sup_x f(x, y)$ is a constant. The last inequality is true for every x_0 so that $\sup_x \inf_y f(x_0, y) \leq \inf_y \sup_x f(x, y)$. □

Lemma 8.2.2 *Let $f(x, y)$ be a reel function of two variables defined over a domain $X \times Y$. For every $x_0 \in X$ and $y_0 \in Y$ we have $\inf_{y \in Y} f(x_0, y) \leq \sup_{x \in X} f(x, y_0)$. Furthermore, the previous inequality is an equality only if*

$$\max_{x \in X} \inf_{y \in Y} f(x, y) = \min_{y \in Y} \sup_{x \in X} f(x, y) = f(x_0, y_0).$$

PROOF.

$$\begin{aligned} \inf_{y \in Y} f(x_0, y) &\leq \sup_{x \in X} \inf_{y \in Y} f(x, y) \\ &\leq \inf_{y \in Y} \sup_{x \in X} f(x, y) \quad \text{(by Lemma 8.2.1)} \\ &\leq \sup_{x \in X} f(x, y_0). \end{aligned}$$

If $\inf_{y\in Y} f(x_0,y) = \sup_{x\in X} f(x,y_0)$ the previous inequalities are equalities so that $\inf_{y\in Y} f(x_0,y) = \sup_{x\in X}\inf_{y\in Y} f(x,y)$. This shows that the previous sup is achieved (for $x = x_0$) and hence that $\sup_{x\in X}\inf_{y\in Y} f(x,y) = \max_{x\in X}\inf_{y\in Y} f(x,y)$. Similarly $\inf_{y\in Y}\sup_{x\in X} f(x,y) = \min_{y\in Y}\sup_{x\in X} f(x,y)$. To finish, note that, obviously, $\inf_{y\in Y} f(x_0,y) \le f(x_0,y_0) \le \sup_{x\in X} f(x,y_0)$. Hence the equality $\inf_{y\in Y} f(x_0,y) = \sup_{x\in X} f(x,y_0)$ occurs only if $\max_{x\in X}\inf_{y\in Y} f(x,y) = \min_{y\in Y}\sup_{x\in X} f(x,y) = f(x_0,y_0)$. □

The following result strengthens the preceding one.

Proposition 8.2.3 *Let $f(x,y)$ be a reel function of two variables defined over a domain $X\times Y$. Define $O_1 = \{x'\in X;\, \inf_{y\in Y} f(x',y) = \sup_{x\in X}\inf_{y\in Y} f(x,y)\}$ and similarly $O_2 = \{y'\in Y;\, \sup_{x\in X} f(x,y') = \inf_{y\in Y}\sup_{x\in X} f(x,y)\}$. Then there exists $x_0\in X$ and $y_0\in Y$ such that*

$$\inf_{y\in Y} f(x_0,y) = \sup_{x\in X} f(x,y_0)$$

if and only if

$$\max_{x\in X}\inf_{y\in Y} f(x,y) = \min_{y\in Y}\sup_{x\in X} f(x,y).$$

Furthermore, if this condition is satisfied, O_1 and O_2 are both non-empty and for every $x_0\in O_1$ and every $y_0\in O_2$ we have

$$\inf_y f(x_0,y) = \sup_x f(x,y_0) = f(x_0,y_0) = \max_{x\in X}\inf_{y\in Y} f(x,y) = \min_{y\in Y}\sup_{x\in X} f(x,y)\,.$$

Proof. By Lemma 8.2.2, there exists $x_0\in X$ and $y_0\in Y$ such that

$$\inf_{y\in Y} f(x_0,y) = \sup_{x\in X} f(x,y_0)$$

only if $\max_{x\in X}\inf_{y\in Y} f(x,y) = \min_{y\in Y}\sup_{x\in X} f(x,y)$. Conversely assume that $\max_{x\in X}\inf_{y\in Y} f(x,y) = \min_{y\in Y}\sup_{x\in X} f(x,y)$. Part of the assumption is that $\sup_{x\in X}\inf_{y\in Y} f(x,y) = \max_{x\in X}\inf_{y\in Y} f(x,y)$ which means that O_1 is non-empty. Symmetrically, O_2 is non-empty. Let x_0 and y_0 be any two elements of O_1 and O_2 respectively. By definition, $\inf_y f(x_0,y) = \max_{x\in X}\inf_{y\in Y} f(x,y)$ and $\sup_x f(x,y_0) = \min_{y\in Y}\sup_{x\in X} f(x,y)$. By assumption these two values are equal so that $\inf_y f(x_0,y) = \sup_x f(x,y_0)$. By Lemma 8.2.2, this common value is equal to $f(x_0,y_0)$. □

Lemma 8.2.4 *Let X and Y be two sets such that for every $(x,y)\in X\times Y$, $(a_{x,y}(t))_{t\in\mathbb{N}}$ is a sequence of non-negative numbers. For every $x\in X$ we define*

$$t_{\min}(x) \stackrel{\text{def}}{=} \sup\{\, t;\, \inf_y a_{x,y}(t) > 0\,\}.$$

Similarly, for every $y \in Y$ we define

$$t_{\max}(y) \stackrel{\text{def}}{=} \sup \{\, t;\, \sup_x a_{x,y}(t) > 0 \,\}.$$

Then

$$\sup_x t_{\min}(x) \leq \inf_y t_{\max}(y)\,.$$

PROOF. Define Y' to be the set $Y' \stackrel{\text{def}}{=} \{y \in Y; t_{\max}(y) < \infty\}$. The result is obvious if $t_{\max}(y) = \infty$ for every $y \in Y$. We therefore consider the case where $t_{\max}(y) < \infty$ for some y in Y, i.e., when Y' is non empty. Furthermore, the equality $\inf_{y\in Y} t_{\max}(y) = \inf_{y\in Y'} t_{\max}(y)$ shows that we can restrict our analysis to within Y' in place of Y. Let x_1 and y_1 be two arbitrary elements from X and Y' respectively. The definition of $t_{\max}(y_1)$ implies that $a_{x_1,y_1}(t_{\max}(y_1)+1) = 0$ and hence that $\inf_y a_{x_1,y}(t_{\max}(y_1)+1) = 0$. By definition of $t_{\min}(x_1)$ this implies that $t_{\min}(x_1) \leq t_{\max}(y_1)$. The elements x_1 and y_1 being arbitrary, we obtain that

$$\sup_x t_{\min}(x) \leq \inf_y t_{\max}(y)\,.$$

By assumption, all the numbers $a_{x,y}(t)$ are non-negative and $\inf_y t_{\max}(y) < \infty$ so that $0 \leq \sup_x t_{\min}(x) \leq \inf_y t_{\max}(y) < \infty$. The inequality $\sup_x t_{\min}(x) < \infty$ shows that $\sup_x t_{\min}(x) = \max_x t_{\min}(x)$. Similarly, the inequality $0 < \inf_y t_{\max}(y)$ implies that $\inf_y t_{\max}(y) = \min_y t_{\max}(y)$. Therefore, in the case where $t_{\max}(y) < \infty$, the max-min inequality can be strengthened into

$$\max_x t_{\min}(x) \leq \min_y t_{\max}(y)\,.$$

□

8.3 Some Elements of Game Theory

Recall that, for all column vector X, X^T denotes the transpose of X. For any integer k we let $\mathcal{T}_k$ denote the k-dimensional fundamental simplex and $\mathcal{T}'_k$ denote the set of extreme points of this simplex:

$$\mathcal{T}_k \stackrel{\text{def}}{=} \Big\{(\lambda_1, \ldots, \lambda_n) \in [0,1]^n\,;\, \sum_i \lambda_i = 1\Big\}.$$

$$\mathcal{T}'_k \stackrel{\text{def}}{=} \Big\{(1,0,\ldots,0),(0,1,\ldots,0),\ldots,(0,0,\ldots,1)\Big\}.$$

The following theorem is the celebrated result of Von Neumann which initiated the field of game theory.

Theorem 8.3.1 (Von Neumann) *For any $k \times l$ real matrix $M = (M_{ij})_{ij}$, the equality*

$$\max_{X \in \mathcal{T}_k} \min_{Y \in \mathcal{T}_l} X^T M Y = \min_{Y \in \mathcal{T}_l} \max_{X \in \mathcal{T}_k} X^T M Y$$

is valid. Furthermore, this common value is equal to:

$$\max_{X \in \mathcal{T}_k} \min_{Y \in \mathcal{T}'_l} X^T M Y \text{ and } \min_{Y \in \mathcal{T}_l} \max_{X \in \mathcal{T}'_k} X^T M Y.$$

Note that, $(X, Y) \to X^T M Y$ is continuous on the compact $\mathcal{T}_k \times \mathcal{T}_l$. Hence for every X we can find an element Y_X in $\mathcal{T}_l$ such that $\inf_{Y \in \mathcal{T}_k} X^T M Y = X^T M Y_X$. Furthermore we easily check that we we can select Y_X in such a way that $X \to Y_X$ defines a continuous function on $\mathcal{T}_k$. This immediately implies that

$$\begin{aligned} \sup_{X \in \mathcal{T}_k} \inf_{Y \in \mathcal{T}_l} X^T M Y &= \sup_{X \in \mathcal{T}_k} X^T M Y_X \\ &= \max_{X \in \mathcal{T}_k} X^T M Y_X \\ &= \max_{X \in \mathcal{T}_k} \min_{Y \in \mathcal{T}_l} X^T M Y \, . \end{aligned}$$

We similarly easily establish that $\inf_{Y \in \mathcal{T}_l} \sup_{X \in \mathcal{T}_k} X^T M Y = \min_{Y \in \mathcal{T}_l} \max_{X \in \mathcal{T}_k} X^T M Y$. Note also that the inequality $\sup_X \inf_Y X^T M Y \leq \inf_Y \sup_X X^T M Y$ is a direct consequence of Lemma 8.2.1. A proof of the converse can easily be derived with the use of the duality result in Linear Programming.

The game theory interpretation of this result is as follows. Consider two players *Player*(1) and *Player*(2) involved in a game (G, Π, A) with a performance function f. G is the set of rules of the game describing how the game is played between the two players, which actions (and in which order) are to be undertaken by the players and in particular which information is traded between the two players during the execution of a game. Π is the set of allowable strategies of *Player*(1), A is the set of allowable strategies of *Player*(2): Π (resp. A) is a subset of the set of strategies of *Player*(1) (resp. *Player*(2)) compatible with the rules of the game G. Note that, by definition, a strategy of either player is defined independently of the strategy chosen by the other player. $f : \Pi \times A \to \mathbb{R}$ is a performance function measuring "how well" a given strategy π of *Player*(1) does when implemented again a given strategy A of *Player*(2). We assume that the game is a *zero-sum noncooperative* game which means that one of the two players, say *Player*(1), wants to chose its strategy so as to maximize the performance $f(\pi, \mathcal{A})$ and that the other player, *Player*(2), wants to chose its strategy so as to minimize the performance.

We consider the *sequential* case where one player chooses first a strategy and where the other player then chooses his. Hence, if *Player*(1) plays first, for every $\varepsilon > 0$,

the two competing players can choose their respective strategies π and $\mathcal{A}$ so as to bring $f(\pi, \mathcal{A})$ to within ε of $\sup_\pi \inf_\mathcal{A} f(\pi, \mathcal{A})$. Conversely, if *Player*(2) plays first, for every $\varepsilon > 0$, the two competing players can choose their respective strategies π and $\mathcal{A}$ so as to bring $f(\pi, \mathcal{A})$ to within ε of $\inf_\mathcal{A} \sup_\pi f(\pi, \mathcal{A})$. By Lemma 8.2.1,

$$\sup_\pi \inf_\mathcal{A} f(\pi, \mathcal{A}) \leq \inf_\mathcal{A} \sup_\pi f(\pi, \mathcal{A}), \tag{8.4}$$

which expresses that, for either player, selecting its strategy last can only be beneficial. Generally, the inequality is strict, i.e., there is a definite advantage in choosing its strategy last.

An interpretation of this inequality is that, eventhough no *explicit* exchange of information is performed between the two players when they select their respective strategies, the player selecting its strategy last can be assumed to know the strategy selected by the player selecting its strategy first. The reason for it is that, for every given choice π made by *Player*(1), there is a *Player*(2) that "always assumes" that *Player*(1) chooses π and that makes an optimized decision $\mathcal{A}$ based on this assumption. Such a *Player*(2) is by construction optimal if *Player*(1) does chose π. Based on this remark we say that an optimal *Player*(2) knows *implicitly* the identity of *Player*(1) in the expression $\sup_\pi \inf_\mathcal{A} f(\pi, \mathcal{A})$. Symmetrically, we say that an optimal *Player*(1) knows *implicitly* the identity of *Player*(2) in the expression $\inf_\mathcal{A} \sup_\pi f(\pi, \mathcal{A})$.

In this setting, the strict inequality in Equation 8.4 expresses precisely that allocating to one or the other player the possibility of spying on the competitor's strategy affects the performance of the game. This interpretation of Inequality 8.4 will be very useful in the rest of the discussion.

Consider now the case where *Player*(1) is provided with a set I of size k and *Player*(2) is provided with a set J of size l. Assume that to each pair (i, j) in $I \times J$ is associated a *cost* $M_{i,j}$ in $\mathbb{R}$. Let $((G, \Pi, A), f)$ be a game with a performance function where $\Pi = I$, $A = J$, $f(i, j) = M_{i,j}$ and where the rules are the trivial rules: "do nothing". In the case where *Player*(1) and *Player*(2) both choose their strategies optimally and where *Player*(1) chooses first, the performance associated to the game is $\max_i \min_j M_{i,j}$. Conversely, if *Player*(2) plays first, the performance associated to the game is $\min_j \max_i M_{i,j}$. As discussed above, $\max_i \min_j M_{i,j} \leq \min_j \max_i M_{i,j}$ and, generally, the inequality is strict, i.e., there is a definite advantage in making its choice last.

We consider now the case where the players are allowed to make random choices. The following theorem formalizes the fact that, in a game of cards played by two players, knowing the opponent's strategy confers no advantage for winning any single hand, provided that the hand finishes in a bounded number of transactions.

We abuse language and identify a probability distribution with the procedure consisting of drawing an element at random according to this probability distribution.

Theorem 8.3.2 *Consider a two-players game* (G, Π, A) *having the property that there exists two* finite *sets* I *and* J *such that* Π *is the set of probability distributions on* I *and such that* A *is the set of probability distributions on* J*. Let* T *be a function on* $I \times J$*. (*T *is often called the cost function.) For every* π *in* Π *and every* $\mathcal{A}$ *in* A *we let* $E_{\pi,\mathcal{A}}[T]$ *denote the expected value of* T *when Player*(1) *selects an element in* I *according to the distribution* π *and when Player*(2) *selects an element in* J *according to the distribution* $\mathcal{A}$*. Then*

$$\max_{\pi \in \Pi} \min_{\mathcal{A} \in A} E_{\pi,\mathcal{A}}[T] = \min_{\mathcal{A} \in A} \max_{\pi \in \Pi} E_{\pi,\mathcal{A}}[T] \,.$$

The sets I and J are often called the set of *pure strategies*. The sets Π and A are often called the set of *mixed strategies*.

PROOF. A probability distribution on I is represented by a k-tuple $(\lambda_1, \ldots, \lambda_k)$ of non-negative numbers summing up to one. Equivalently, $\mathcal{T}_k$ is the set of probability distributions on I and similarly $\mathcal{T}_l$ is the set of probability distributions on J. By assumption Π can be identified to $\mathcal{T}_k$ and a strategy π in Π can be represented by element X in $\mathcal{T}_k$. Similarly, A can be identified to $\mathcal{T}_l$ and a strategy $\mathcal{A}$ in A can be represented by element Y in $\mathcal{T}_l$. For every $(i,j) \in I \times J$ we write $M_{i,j} = T(i,j)$ and we let M represent the associated matrix: by construction M is symmetric. Using these associations, consider the case where *Player*(1) chooses the strategy $X \in \mathcal{T}_k$ and where *Player*(2) chooses the strategy $Y \in \mathcal{T}_l$. The quantity $X^T M Y$ represents the expected value of the cost T obtained under these strategies. Theorem 8.3.2 is therefore a direct application of Theorem 8.3.1. □

A game as one described in Theorem 8.3.2 is is often called a *matrix game*.

Explicit/Implicit knowledge. In the course of the previous discussion we introduced the notion of implicit knowledge. We present here an abstract summarizing this concept.

We say that a player, say *Player*(2), receives *explicitly* some information x during the course of an execution when the rules of the game (i.e., when considering algorithms, the Π/A-structure) specify that that *Player*(2) be informed of x. We can say figuratively that *Player*(2) "receives a message" carrying the information x. Note that, in this situation, *Player*(2) receives the information x independently of the strategy that it follows.

Consider now a function of two variables $f(x, \mathcal{A}), (x, \mathcal{A}) \in X \times A$ and consider the expression $\inf_{\mathcal{A}} f(x, \mathcal{A})$. We can consider this situation as a game parameterized by x played by *Player*(2): *Player*(2) tries to decide $\mathcal{A}$ so as to minimize $f(x, \mathcal{A})$. Eventhough there might be no mechanism letting *Player*(2) explicitly know what the parameter x is, when considering the expression $\inf_{\mathcal{A}} f(x, \mathcal{A})$ we can assume that an optimal (and lucky) *Player*(2) selects non-deterministically an $\mathcal{A}$ bringing the function $f(x, \mathcal{A})$ arbitrarily close to $\inf_{\mathcal{A}} f(x, \mathcal{A})$: we then say that an optimal *Player*(2) knows implicitly the parameter x selected. This knowledge is not a real knowledge and is of course nothing more then a heuristic meaning that some choice of $\mathcal{A}$ corresponds ("by chance") to the choice that some informed *Player*(2) would make. Note that, in contrast to the case of explicit knowledge, *Player*(2) is said to "have the implicit knowledge" when it chooses a "good" strategy $\mathcal{A}$.

We provide two examples of this situation. When considering the formula

$$\sup_{\pi \in \Pi} \inf_{\mathcal{A} \in A} f(\pi, \mathcal{A})$$

we will say that *Player*(2) knows implicitly the value π. Also when considering the formula

$$\inf_{\mathcal{A}} P_{\mathcal{A}}[C \mid B]$$

we will say that *Player*(2) knows implicitly that the sample space is restricted to within B.

Bibliography

[1] N. Alon and M. Rabin. Biased coins and randomized algorithms. *Advances in Computing Research, JAI Press*, 5:499–507, 1989.

[2] R. Alur, C. Courcoubetis, and D. Dill. Model checking for probabilistic real-time systems. In *Proceedings* 18^{th} *Int. Coll. on Automata Languages and Programming (ICALP)*, 1991.

[3] J. Aspnes and M. Herlihy. Fast randomized consensus using shared memory. *Journal of Algorithms*, 15(1):441–460, September 1990.

[4] J. Aspnes and O. Waarts. Randomized consensus in expected $o(n \log^2 n)$ operations per processor. In 33^{rd} *Annual Symposium on Foundations of Computer Science*, 1992.

[5] H. Attiya and M. Snir. Better computing on the anonymous ring. *Journal of Algorithms*, 12:204–238, 1991.

[6] J.P. Aubin. *Mathematical Methods of Game and Economic Theory.* North-Holland Pub. Co., Studies Mathematics and its Applications, 1982.

[7] Y. Azar, A. Broder, and A. Karlin. On-line load balancing. In 18^{th} *Annual Symposium on Foundations of Computer Science,* Providence, Rhode Island, pages 218–225, 1992.

[8] A. Benveniste, B. Levy, E. Fabre, and P. LeGuernic. A calculus of stochastic systems: their specifications, simulation, and hidden state simulation. *Tech. Rep. 606-Irisa-France*, 1994.

[9] A. Benveniste, M. Métivier, and P. Priouret. *Adaptive Algorithms and Stochastic Approximations.* Applications of Mathematics. Springer Verlag, 1990.

[10] P. Billingsley. *Convergence of Probability Measures.* Wiley Series in Probability and Mathematical Statistics. John Wiley, 1968.

[11] A. Blum and P. Chalasani. Learning switching concepts. In *COLT 92*, 1992.

[12] J. Burns, M. Fischer, P. Jackson, N. Lynch, and G. Peterson. Data requirements for implementation of n-process mutual exclusion using a single shared variable. *Journal of the ACM*, 29(1):183–205, January 1982.

[13] B. Chor and C. Dwork. Randomization in byzantine agreement. *Advances in Computing Research, JAI Press*, 5:443–498, 1989.

[14] S. Cohen, D. Lehmann, and A. Pnueli. Symmetric and economical solutions to the mutual exclusion problem in a distributed system. *Theoretical Computer Science*, 34:215–225, 1984.

[15] S. Ben David, A. Borodin, R. Karp, G. Tardos, and A. Widgerson. On the power of randomization in online algorithms. In *Proceedings of the* 22^{nd} *Annual ACM Symposium on Theory of Computing*, pages 379–386, 1990.

[16] E. W. Dijkstra. Solution of a problem in concurrent programming control. *Communications of the ACM*, pages 643–644, 1966.

[17] P. Feldman and S. Micali. Optimal algorithms for byzantine agreement. In *Proceedings of the* 20^{th} *Annual ACM Symposium on Theory of Computing*, pages 148–161, 1988.

[18] Y. Feldman. A decidable propositional probabilistic dynamic logic. In *Proceedings of the* 15^{th} *Annual ACM Symposium on Theory of Computing, Boston, Massachusetts*, pages 293–309, 1983.

[19] M. Fischer and L. Zuck. Reasoning about uncertainty in fault-tolerant distributed systems. In *Proceedings of the 1988-Annual Symposium on Formal Techniques on Real Time and Fault Tolerant Systems*, pages 142–157, 1988.

[20] P. Flajolet and N. Martin. Probabilistic counting algorithms for data base applications. *Journal of Computer and System Sciences*, 31:182–209, 1985.

[21] N. Francez and M. Rodeh. A distributed data type implemented by probabilistic communication scheme. In 21^{st} *Annual Symposium on Foundations of Computer Science,* Syracuse, New York, pages 373–379, 1980.

[22] R. Gawlick, R. Segala, J.F. Søgaard-Andersen, and N. Lynch. Liveness in timed and untimed systems. Technical Report MIT/LCS/TR-587, MIT Laboratory for Computer Science, November 1993.

[23] R.J. van Glabbeek, S.A. Smolka, B. Steffen, and C.M.N. Tofts. Reactive, generative, and stratified models of probabilistic processes. In *Proceedings of the 5th Annual Symposium on Logic in Computer Science*, pages 130–141, 1990.

[24] M. Goemans, N. Lynch, and I. Saias. Upper and lower bounds on the number of faults a system can withstand without repairs. In *Proceedings of the 4^{th} IFIP Working Conference on Dependable Computing for Critical Applications,* San Diego, California, 1994.

[25] R. Graham and A.C.-C. Yao. On the improbability of reaching byzantine agreements. In *Proceedings of the 21^{st} Annual ACM Symposium on Theory of Computing, Seattle, Washington*, pages 467–478, 1989.

[26] R. Gupta, S. Smolka, and S. Bhaskar. On randomization in sequential and distributed systems. *ACM Computing Surveys*, 26(1):7–86, 1994.

[27] H. Hansson. *Time and Probability in Formal Design of Distributed Systems.* PhD thesis, Department of Computer Science, Uppsala University, 1991.

[28] S. Hart and M. Sharir. Proceedings of the 16^{th} annual acm symposium on theory of computing, washington, d. c. In *Probabilistic Temporal Logics for Finite and Bounded Models*, pages 1–13, 1984.

[29] S. Hart, M. Sharir, and A. Pnueli. Termination of probabilistic concurrent programs. *ACM Transactions on Programming Languages and Systems*, 5(3):356–380, 1983.

[30] C.A.R. Hoare. *Communicating Sequential Processes.* Prentice-Hall International, Englewood Cliffs, 1985.

[31] J.E. Hopcroft and R.M. Karp. A $n^{5/2}$ algorithm for maximum matching in bipartite graphs. *SIAM Journal on Computing*, 2:225–231, 1973.

[32] C.C. Jou and S. Smolka. Equivalences, congruences and complete axiomatizations for probabilistic processes. In *Proceedings of CONCUR*, 1990.

[33] A. Karlin and A. Yao. Probabilistic lower bounds for byzantine agreement. *Unpublished document.*

[34] H. W. Kuhn and A. W. Tucker. *Contributions to the Theory of Games.* Annals of Mathematics Studies 24. Princeton University Press, 1950.

[35] E. Kushilevitz, Y. Mansour, M. Rabin, and D. Zuckerman. Lower bounds for randomized mutual exclusion. In *Proceedings of the 25^{th} Annual ACM Symposium on Theory of Computing*, pages 154–163, 1993.

[36] E. Kushilevitz and M. Rabin. Randomized mutual exclusion algorithms revisited. In *Proceedings of the* 11^{th} *Annual ACM Symposium on Principles of Distributed Computing,* Quebec, Canada, pages 275–284, 1992.

[37] D. Lehmann and M. Rabin. On the advantage of free choice: a symmetric and fully distributed solution to the dining philosophers problem. In *Proceedings of the* 8^{th} *Annual ACM Symposium on Principles of Programming Languages*, pages 133–138, January 1981.

[38] D. Lehmann and S. Shelah. Reasoning with time and chance. *Information and Control*, 53:165–198, 1982.

[39] L. Lovász and M. Plummer. *Matching theory.* Annals of Discrete Mathematics, 29. North-Holland, Mathematical Studies 121, 1986.

[40] N. Lynch, I. Saias, and R. Segala. Proving time bounds for randomized distributed algorithms. In *Proceedings of the* 13^{th} *Annual ACM Symposium on Principles of Distributed Computing,* Los Angeles, California, pages 314–323, 1994.

[41] N.A. Lynch and F.W. Vaandrager. Forward and backward simulations – part II: Timing-based systems. Technical Report MIT/LCS/TM-487, MIT Laboratory for Computer Science, September 1993.

[42] U. Mamber and M. Tompa. The complexity of problems on probabilistic, nondeterministic, and alternating decision trees. *Journal of the ACM*, 32:720–732, 1985.

[43] N. Maxemchuck and K. Sabnani. *Probabilistic Verification of Communication Protocols*, volume VII of *Protocol Specification, Testing, and Verification.* North-Holland, 1987.

[44] X. Nicollin and J. Sifakis. An overview and synthesis on timed process algebras. In K.G. Larsen and A. Skou, editors, *Proceedings of the Third Workshop on Computer Aided Verification,* Aalborg, Denmark, July 1991, volume 575 of *Lecture Notes in Computer Science*, pages 376–398. Springer-Verlag, 1992.

[45] O. Ore. Graphs and matching theorems. *Duke Math. J.*, 22:625–639, 1955.

[46] A. Pnueli and L. Zuck. Probabilistic verification by tableaux. In *Proceedings of the 1st IEEE Symposium on Logic in Computer Science*, 1986.

[47] A. Pnueli and L. Zuck. Verification of multiprocess probabilistic protocols. *Distributed Computing*, 1(1):53–72, 1986.

[48] M. Rabin. The choice coordination problem. *Acta Informatica*, 17:121–134, 1982.

[49] M. Rabin. N-process mutual exclusion with bounded waiting by $4 \log N$- shared variable. *Journal of Computer and System Sciences*, 25:66–75, 1982.

[50] M. Rabin. Randomized byzantine generals. In 24^{th} *Annual Symposium on Foundations of Computer Science,* Tucson, Arizona, pages 403–409, 1983.

[51] J.R. Rao. Reasoning about probabilistic algorithms. In *Proceedings of the* 9^{th} *Annual ACM Symposium on Principles of Distributed Computing,* Quebec, Canada, August 1990.

[52] J. Reif and P. Spirakis. Real-time synchronization of interprocess communication. *ACM Transactions on Programming Languages and Systems*, 6:215–238, 1984.

[53] I. Saias. Proving probabilistic correctness statements: the case of rabin's algorithm for mutual exclusion. In *Proceedings of the* 11^{th} *Annual ACM Symposium on Principles of Distributed Computing,* Quebec, Canada, pages 263–274, 1992.

[54] R. Segala and N. Lynch. A model for randomized concurrent systems. Manuscript under preparation, 1994.

[55] K. Seidel. *PhD Thesis: Probabilistic Communicating Processes.* Oxford University, 1993.

[56] A.N. Shiryayev. *Probability.* Graduate Texts in Mathematics 95. Springer Verlag, 1984.

[57] F.W. Vaandrager and N.A. Lynch. Action transducers and timed automata. In W.R. Cleaveland, editor, *Proceedings CONCUR 92,* Stony Brook, NY, USA, volume 630 of *Lecture Notes in Computer Science*, pages 436–455. Springer-Verlag, 1992.

[58] M. Y. Vardi. Automatic verification of probabilistic concurrent finite-state programs. In *Proceedings of 26th IEEE Symposium on Foundations of Computer Science*, pages 327–338, Portland, OR, 1985.

[59] Vorob'ev. *Game theory.* Applications of Mathematics. Springer Verlag, 1977.

[60] C. West. *Protocol Validation by Random State Exploration*, volume VI of *Protocol Specification, Testing, and Verification.* North-Holland, 1986.

[61] A.C.-C. Yao. Probabibilistic computations: Toward a unified measure of complexity. In 18^{th} *Annual Symposium on Foundations of Computer Science,* Providence, Rhode Island, pages 222–227, 1977.

[62] L. Zuck and A. Pnueli. Verification of multiprocess probabilistic protocols. *Distributed Computing*, 1(1):53–72, 1986.

Index